Journal of Biomedical Informatics 105 (2020) 103420

Contents lists available at ScienceDirect

Journal of Biomedical Informatics

journal homepage: www.elsevier.com/locate/yjbin

Facilitating the development of cross-platform mHealth applications for chronic supportive care and a case study

Devasena Inupakutika[a,*], Sahak Kaghyan[a], David Akopian[a], Patricia Chalela[b], Amelie G. Ramirez[b]

[a] Department of Electrical and Computer Engineering, The University of Texas at San Antonio, TX, 78249, USA
[b] Institute for Health Promotion Research (IHPR), University of Texas Health San Antonio, San Antonio, TX, 78229, USA

ARTICLE INFO

Keywords:
Smartphones
Mobile devices
Mobile computing
Cross-platform
Mobile health
Applications

ABSTRACT

Mobile health (mHealth) apps have received increasing attention, due to their abilities to support patients who suffer from various conditions. mHealth apps may be especially helpful for patients with chronic diseases, by providing pertinent information, tracking symptoms, and inspiring adherence to medication regimens. To achieve these objectives, researchers need to prototype mHealth apps with dedicated software architectures. In this paper, a cloud-based mHealth application development concept is presented for chronic patient supportive care apps. The concept integrates existing software platforms and services for simplified app development that can be reused for other target applications. This developmental method also facilitates app portability, through the use of common components found across multiple mobile platforms, and scalability, through the loose coupling of services. The results are demonstrated by the development of native Android and cross-platform web apps, in a case study that presents an mHealth solution for endocrine hormone therapy (EHT). A performance analysis methodology, an app usability evaluation, based on focus group responses, and alpha and pre-beta testing results are provided.

1. Introduction

Mobile health (mHealth) applications (apps) are significantly transforming how healthcare services and information are being delivered, accessed and managed. The market share for healthcare apps peaked in 2017 and is expected to generate 111.1 billion by 2025 [1]. Nearly 83% of physicians currently utilize smartphones and medical apps to remotely monitor their patients' chronic conditions [2]. At a fundamental level, mHealth involves the use of mobile apps that distribute, deliver or collect healthcare information. mHealth is an emerging intersection among medical informatics, public health, and business, and refers to any health services or health-related information that is delivered or enhanced through the Internet and related technologies. The goal is to improve healthcare globally, using information and communication technologies (ICT) [3]. Studies have evaluated the use of mHealth apps from the perspective of both healthcare professionals and patients [4–6].

mHealth technologies over the last few years have resulted in the development of numerous platforms that offer opportunities for healthcare providers to deliver high-quality care remotely. Adherence to therapies in outpatient settings is crucial to reducing the widespread prevalence of chronic diseases, such as cancer, human immunodeficiency virus (HIV)/acquired immunodeficiency syndrome (AIDS), malaria, tuberculosis, and diabetes. The major challenges of chronic disease treatment include the maintenance of regular and continuous patient-physician communication, the lack of adherence, potential side effects, impact on quality of life and other outcomes reported by patients. The increasing number of mHealth solutions has allowed these challenges to be overcome and provided new opportunities [7–10]. Mobile technologies have begun to assume active roles in healthcare systems, increasing patient engagement as is necessary during chronic disease supportive care, which aims to provide side-effect relief for patients and are primarily used during disease treatment to support patient self-management activities in addition to disease prevention and diagnosis [11]. The large-scale adoption of mHealth apps for chronic disease management depends on the usefulness and

Abbreviations: mHealth, mobile health; App, application; CP, cross-platform; PEF, Patient Engagement Framework; AWS, Amazon web services; EC2, elastic cloud compute; REDCap, research electronic data capture; FCM, Firebase cloud messaging; APK, android package; CPU, central processing unit

* Corresponding author.
E-mail addresses: devasena.inupakutika@my.utsa.edu (D. Inupakutika), sahak.kaghyan@gmail.com (S. Kaghyan), david.akopian@utsa.edu (D. Akopian), chalela@uthscsa.edu (P. Chalela), ramirezag@uthscsa.edu (A.G. Ramirez).

https://doi.org/10.1016/j.jbi.2020.103420
Received 18 September 2019; Received in revised form 30 March 2020; Accepted 1 April 2020
Available online 07 April 2020
1532-0464/

effectiveness of the app and the inclusion of entertaining features capable of engaging users and promoting adherence to therapies. Patient awareness and care are necessary for successful self-management and adherence to treatment regimens. The combination of ICT with healthcare system support offers the potential to address concerns associated with patient self-management.

1.1. Importance of mHealth in chronic condition care

From a technological perspective, extensive reviews of the current state of mHealth services [15,16] examined the most significant research studies and presented detailed analyses of the existing, novel mHealth services and applications. The typical mHealth architecture uses the Internet and web-based services to enable authentic and pervasive patient-doctor interactions. Furthermore, the mHealth apps currently in use include access to patient's health records via web portals for healthcare providers and doctors, patient communication, access to medical education via internet educational content, videos, disease-specific documentation and clinical trials for patients [17,18]. These apps enable chronic disease survivors to maintain close and personal relationships with patient navigators (PNs) and offer immediate access to providers in case of emergencies or other issues during long-term therapies [19]. Many other studies have also demonstrated the effectiveness and the potential of offering increased access to comprehensive cancer-related care to patients in various geographical locations using remote consultations with PNs, doctors and healthcare providers.

1.2. mHealth characteristics and challenges

As mobile technology continues to evolve, ensuring that the characteristics of mHealth apps remain relevant to all mobile device platforms becomes increasingly important. The core characteristics of mHealth are as follows: (1) the adoption of apps use among human communities; (2) the availability and form of apps; (3) the availability and configuration of Internet access and network connections; and (4) the devices tethered to individuals [20]. The suitability and significance of these mHealth characteristics must continue to be relevant as technology advances. Considering these issues is essential during the design, implementation, and selection of mHealth-based solutions for healthcare systems and providers. Several qualitative [21–23] and empirical [24] studies have examined mobile app development, to investigate the challenges faced by researchers and practitioners. Mobile device and platform variability, app fragmentation, and the lack of interoperability across platforms are major challenges that must be overcome for the broader deployment of mHealth solutions and can influence the characteristics of mHealth apps. Fragmentation is the most commonly reported challenge, encountered in 58%, 8%, and 4% of the identified literature examining native, web and hybrid mobile apps, respectively [24]. Fragmentation can cause portability issues for mobile apps, which is prevalent among the previously mentioned mobile app types, due to platform heterogeneity. The lack of portability has resulted in considerable efforts being spent to transform apps or data from one platform/system into a format that is usable by other platforms/systems. Data portability is addressed by existing standards, such as the International Organization for Standardization (ISO)/The Institute of Electrical and Electronics Engineers (IEEE) 11073 [31] and Health Level Seven (HL7) [32]. However, standards can be used to standardizing tasks when data (clinical and administrative) cannot be exchanged between the involved mobile devices or platforms using common serialization formats, such as XML or JavaScript Object Notation (JSON). In typical developmental settings, a common practice is to develop native mobile apps for a single platform and then attempt to re-implement the app for all other platforms, starting from scratch, to address the portability challenges caused by platform fragmentation [25]. In research-oriented settings, such as academic studies, project-associated time and cost constraints often require accelerated prototyping strategies, and frequent app adjustments may be necessary due to continuous feedback from patients, healthcare promotion researchers, and the observations from intervention studies. The additional requirement for scalability when adapting mobile apps to such evolving needs becomes clear, and provide researchers and patients with solutions to these problems is preferable to switching them to a new, specific app, allowing the apps to continue providing patients with new services over time. To solve these problems, the mHealth system architecture must be flexible and modular, with loosely coupled components.

In this study, we present a cloud-based mHealth system architecture and the development of native and web apps that utilize this architecture, which addresses both portability and scalability aspects. The architecture enables the integration of existing, heterogeneous software components, which can be reused for different platforms, facilitating portability. A partial, cloud-based, cross-platform (CP) approach is employed for use during app development, in which some services are cloud-based, whereas others operate on the terminal device. Inter-cloud collaborations between different cloud providers is an essential requirement when addressing mobile-cloud interoperability challenges across diverse devices and platforms, which is the focus of this paper. Data portability aspects are thoroughly addressed in the literature and are, therefore, not addressed in this paper [26]. To demonstrate our concepts, native Android and CP web apps are developed for a case study of this mHealth solution to support endocrine hormone therapy (EHT). We developed prototype hormone therapy (HT) Patient Helper native apps for the most dominant mobile OSs, i.e., Android and iOS. However, we will restrict our discussion in this paper to the native Android version and the CP web app, due to space limitations.

Scalability aspects are addressed by the integration of heterogeneous components (corresponding to the different services that each module in the system has to offer) as separate modules, using centralized data models that are not platform-dependent. Because of the loose coupling between components, the prototypes can be better tailored to various changes.

Using the case study solution, a performance analysis methodology and usability evaluations are conducted for the apps, through focus groups, alpha, and pre-beta testing.

The rest of the paper is organized as follows. Section 2 presents an overview of the three types of mobile apps and state-of-the-art CP developmental approaches. Section 3 presents a description of the system architecture. Section 4 describes the native Android and web prototype apps that are facilitated through the architecture along with the requirements that the integrated system and the apps have to fulfill, discusses the technology components that are integrated into the system, their functionality in the current study and explains why these components are chosen. Section 5 discusses the technical system performance methodology, metrics and assessment of both apps to analyze the differences between them and presents results from the usability evaluation. Section 6 provides an analysis of the challenges of the presented architecture. Finally, Section 7 concludes this paper and provides directions for future work.

2. State-of-the-art cross-platform development approaches for mobile apps

2.1. Mobile app development approaches

In general, mobile apps can be developed as web, native, or hybrid apps. Web apps run on web-based servers, are accessible via desktop/mobile web browsers, and are highly portable across multiple platforms. Because these apps can be used across diverse platforms, the time needed for development and deployment cycles and the costs associated with app development are reduced. However, mobile web apps cannot access device-specific hardware (HW) features, such as cameras and other sensors. Native apps utilize the device capabilities and offer

better performance than web apps. These apps are available for download via mobile OSs or platform-specific app stores. However, the development of native apps for multiple platforms requires expertise in the relevant programming languages, Software Development Kits (SDK), and APIs, which ultimately requires extra resources in terms of time, skills, and costs [21]. Hybrid apps can be installed on a device, similar to a native app, but are written using the same technology as web apps. They behave like native apps and can, therefore, access device-specific HW features and be downloaded from platform-dedicated app stores [33].

2.2. State-of-the-art cross-platform development approaches

In recent years, efforts have been made to address the portability issues caused by mobile device/platform/app fragmentation. These efforts have been directed towards CP development tools and frameworks. These frameworks allow apps to be developed and distributed across multiple platforms, reducing the costs of required resources and maximizing code reuse. An extensive study [34] initially classified several CP approaches that have emerged, evaluating web apps developed using the web-technology-based PhoneGap [35] and Titanium Mobile [36] and compared these apps with natively developed mobile apps. They presented a comprehensive set of criteria for evaluating these approaches. Mobile web apps represent quicker and simpler methods for app development because they do not require native development; however, web-based apps perform poorly with regards to speed and functionality. This study in Ref. [34] offers general guidelines, but specific projects would require further tuning and customization. The study also highlighted the tradeoffs that should be considered when attempting to develop CP solutions for existing mobile apps. More recent studies [37] have also performed comprehensive analyses of innovative CP development frameworks, based on *React Native, the Ionic Framework,* and *Fuse.* Furthermore, technology has been moving towards the development of special languages for mobile app development that include model-based development (MDD), model-based languages (MDLs) [38], or domain-specific languages (DSLs) [39,40] and the application of product-line model-driven engineering to mobile app development [41] (specifically for industry-grade, feature-based, mobile apps) and component-based frameworks (CBFs) [42]. The key to these approaches is driven by modeling activities and we can describe the app on a high-level which translates to a native code based on the target platform. We can categorize the adopted CP mobile app development approaches as follows: 1) *Compilation (Cross-compiler and Trans-compiler)*; 2) *Component-Based*; 3) *Interpretation (Virtual Machine, Web-Based and Runtime Interpretation);* 4) *Modeling (Model-Based UI development and MDD)*; 5) *Cloud-Based* where app processing is delegated to the cloud [43]; and 6) *Merged* [44], which leverages all the benefits from the above approaches but requires huge efforts for development and requires great maintenance efforts for continued proper functioning. Moreover, the *Merged* approach is only supported on Android and Windows Phone 8. We observed that no approach offers a clear advantage and that some frameworks are associated with performance penalties. Despite the advances and research regarding CP techniques, the apps that are developed using these frameworks require serious tailoring for use on different mobile platforms, including translation into native programming languages, UI updates, and source code reuse, to develop fully functional mobile apps, making app development cumbersome and requiring developers to understand a variety of technologies.

3. Cloud-based mHealth system architecture

This section presents the application of a cloud-based mHealth architecture for the facilitation of native and CP web apps for chronic condition supportive care, as shown in Fig. 1. The presented architecture is intended to reduce the impacts of fragmentation, caused by diversity among devices (ranging from tablets to smartphones) and OS platforms, and provide services that can accommodate changing user preferences and needs. The system is based on cloud storage, consisting of a private database (SQL or NoSQL) that is specific to each component or service. Amazon web services (AWS) or Google Cloud Platform (GCP) can also be used as central databases (DBs) for all system components. All data that moves through the mHealth system should end up in this database. This architecture allows the use of different types of databases, based on the mHealth system component storage.

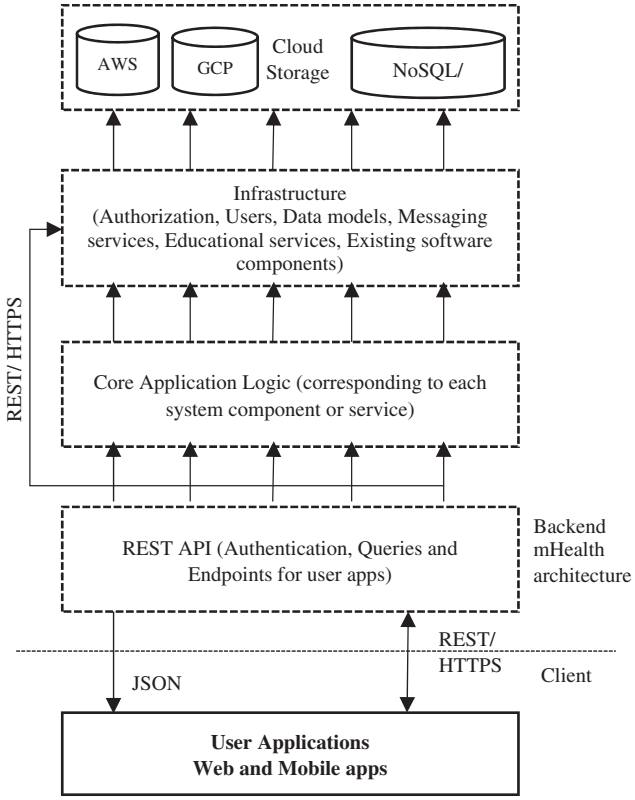

Fig. 1. Cloud-based mHealth system architecture.

The next layer is a set of infrastructural components that impose the key system features. Data models, educational, messaging, user authentication and authorization services, and user management components that are common to the system, are provided by this layer. This layer also provides data separation, on a per-application (existing software components integrated into the system) basis. This layer also ensures that the common components required by any user app are reusable.

The core application logic layer follows the infrastructure layer. This layer consists of the logic and functionality of each involved system component. Communications occur between each component, based on the actions triggered by the user in the mobile or web app, via a representational state transfer (REST) API, as shown in Fig. 1. The components use JSON to exchange data.

The final layer is a REST API. User apps (native mobile or web) user interfaces (UIs) connect with this layer for authentication, data querying, and triggering actions. This architecture is optimized in three directions: simplified development, by reusing existing software components; portability, through the use of common components that offer multiple features; and scalability, through loose coupling between involved components. New apps can be developed by instantiating or including only those components that the app requires.

4. Case study: Prototype development

This section aims to demonstrate the feasibility of applying the mHealth system architecture to the development of a supportive app in

both native Android and web formats, designed to improve EHT adherence among breast cancer patients. Poor adherence to long-term therapies, such as EHT, can compromise the effectiveness of these therapies. To alleviate this issue and to improve breast cancer survivorship care and the patients' overall wellbeing, we utilized the Patient Engagement Framework (PEF) [13] during the design and implementation of case study mHealth apps. The PEF model consists of app engagement strategies. Researchers have expended serious efforts [14] to maintain patient engagement, to maximize long-term adherence to therapies. As described in the study protocol in Ref. [12], the University of Texas, San Antonio (UTSA) and University of Texas Health Science Center at San Antonio (UTHSCSA) teams both performed usability studies, received constant feedback, and performed focus group interviews with actual breast cancer patients who were undergoing EHT therapy, which influenced the iterative development of the native Android mobile and CP web apps.

4.1. EHT requirements

The research team conducted a thorough requirements analysis before designing and implementing the system. We have documented the requirements in a structured and comprehensive manner, in the form of specifications documents starting from the formative research phase. These documents guided the entire design and development process. The iterative design and development cycle consisted of specifications, app mockups, and prototypes. We obtained healthcare researchers' reviews and user feedback and performed usability studies for the prototypes. This iterative process ensured that the requirements, user (both healthcare researchers and patients) expectations, and system development remained aligned. In Ref. [12], we provided an overview of the research setting, intended to support a patient population with a large minority component, and described the protocol of a randomized, controlled study, with the aim of developing and pilot-testing a bilingual, culturally tailored, personalized, interactive mHealth app to improve EHT adherence |among breast cancer patients. The functional requirements that the integrated mHealth platform and the apps were designed to fulfill include the following: 1) facilitate patient education, the identification and reporting of side effects, and the delivery of self-care advice; 2) empower patients to self-monitor and self-manage; and 3) simplify communications between the patient and the oncology team. The features discussed in Section 4.2 satisfy each of these functional requirements.

4.2. EHT features

This section summarizes the features of the prototype apps that were facilitated using the proposed architecture. The apps include the following features:

1. Provide basic information regarding how to locate services, clinical facilities, and PNs and provides therapy/medication-related education materials using *Komen*-authorized information sources.
2. Track symptom status and symptom progression during therapy.
3. Facilitate the management of symptom histories and summaries by patients.
4. Send push notifications and reminders associated with health, daily care, medication, appointments, and symptom assessment.
5. Offer social media access (through a closed *Facebook* group) to provide a network for support and information exchange with other patients who are undergoing or have undergone therapy.
6. Provide a supportive environment, through interactions with the *Facebook* group and with PNs (administrator), on a regular basis.
7. Provide technical assistance for app use and technical glitches, for both existing features and new functionalities.
8. Provide a web-based dashboard for PN use, allowing easy DB CRUD (Create, Read, Update and Delete) operations for patient records, without requiring PNs to visit the *Firebase* [27] JSON structured DB.
9. Facilitate the transfer of patient-specific app data (specifically symptom data) to *REDCap* [30], facilitating the integration of app data with the patient's baseline data, for further analysis and processing by the healthcare research team.

4.3. Technology description

This section describes the software components utilized, the rationale underlying the choice of each component, and the combination of favorable features and services from each technology that was used to facilitate the development of the CP mHealth prototype apps.

4.3.1. Cloud-based mHealth solution

The decision between cloud-based and on-premise infrastructure depends on the particular project or organizational goals. Significant effort, time, and expertise are necessary for the management and maintenance of an on-premises server, whereas the cloud-based approach offers more flexibility and is less expensive to scale if increased complexity or more space is necessary. Therefore, based on the goals of the current study, we chose a cloud-based solution. mHealth apps are constantly developing and integrating third-party services. The availability of a cloud-based service platform can ease the development and integration of data and services from heterogeneous components, as demonstrated.

This study utilized cloud-based services, hosting the webserver on *AWS EC2* [28] and using Google *Firebase* as the cloud-based DB, for both web and native apps. Each cloud provider utilizes associated web services that provide relevant API information, which we can leverage to increase the interoperability of the data across the constituent software components and apps. This approach enables developers to drastically reduce their app development efforts and time by using the platform-specific services offered by each API. Additionally, we developed an API (last layer) for the integrated system, which was exposed to the web services and operate on top of this cloud infrastructure; this layer will be discussed at the end of this section. With a centralized web server that uses the same backend DB *Firebase* for all apps, the portability issues associated with cloud use, including programming language/framework, platform-specific services, data storage, and platform-specific configuration files, could be resolved by using the specific service APIs offered by each provider.

4.3.2. Google Firebase

Firebase is a cloud-based Backend-as-a-Service for online data storage. It consists of a real-time DB and a Cloud Firestore. Therefore, *Firebase* is a fully-fledged platform for building Android, iOS, and web-based apps, providing authentication services, file storage, analytics, messaging, and data synchronization. The current study leveraged the real-time DB, which is a NoSQL cloud-based DB that syncs data across all clients (native mobile and web apps), in real-time, and provides offline functionality. Data is stored in JSON format, and all clients share the same DB, which allows the automatic reception of updates containing the latest data. All of the data models accessed by the mHealth apps reside on the *Firebase* DB. This DB can scale horizontally and allows updates to the schema, which results in faster iteration cycles that can overcome latency and return optimized results to the mHealth app in a faster, more efficient manner. The app also used *Firebase* to handle user authentication. We selected *Google Firebase* because it provides real-time DB data storage, *Google Cloud Storage,* which can store and serve user-generated content, and *FCM,* for push notifications. *Firebase* is considered to be a fast Publisher/ Subscriber, is easy to use, and deploys a framework that is well-suited for the prototyping of similar mHealth-based systems.

4.3.3. Firebase cloud messaging (FCM)

FCM is a CP cloud notification service and a component of *Google*

Firebase and the *Google Cloud Platform*. *FCM* provides connections to devices, through notifications and messages. The app developed in the current study used *FCM* to send patients reminders regarding medications and doctor or healthcare professional appointments, motivational messages, and reminders regarding symptom submissions. A minimum of 98% of all messages are delivered to devices within 500 ms or less, making this system very reliable. Increased control over the delivery of messages/notifications can be utilized to avoid the unnecessary spamming of unintended users, which makes it a suitable choice for the current study. For example, consider a scenario in which all of the participants in a breast cancer study installed the same app, on different days and at different times. Each patient would receive a reminder to submit their symptoms, exactly 2 weeks after joining the study. *FCM* offers flexible message targeting, allowing notifications to be sent to a single patient or to a group of patients who enrolled on the same date.

4.3.4. Amazon elastic cloud compute (EC2) (node server)

Amazon EC2 provides scalable, pay-as-you-go computing capacity for the AWS cloud, which makes web-scale computing easier and provides fully-fledged control of the EC2 server instances. For the current study, we established a *Node.js* [45] server on EC2 Linux instance. This server serves two functions: hosting a web app for the mHealth platform and acting as an app server that stores the tokens associated with the various devices running the Android and web-based apps. Token storage on the EC2 server allows increased flexibility and scalability when sending notifications to selected users or groups of devices. Normally, we would store the tokens directly on *Firebase* and manually schedule notifications for selected users. However, as the platform complexity increases, using a server to store and send notifications using the *Firebase* API automates the process, reducing the requirements of human interventions and easing future enhancements. Data synchronization scripts that collect patient-specific app data from *Firebase* also reside on the server. These scripts are triggered by any updates to any data and transfer the data to *REDCap*, using respective APIs, to ensure that patients' records are up-to-date for healthcare professionals. Given the current use-case, which leverages most of the Google environment (Android as the mobile OS, Google *Firebase* as the cloud-DB, and *FCM* for push notifications), the Google Cloud Platform (GCP) was an obvious first choice for the app server. However, despite the similar service offerings between Google and Amazon, we chose to use AWS for this study, due to its dominance of the public cloud market [47], the great scope of its operations, a worldwide network of data centers and is the most mature cloud-based service for managing large numbers of resources and users.

4.3.5. GitHub (GitHub Pages) for educational services

The complete library of therapy and medication-related content that can be accessed by the app is static, and *Komen* has authorized its distribution to the public. Because the content is likely to change and be updated, regardless of whether any changes are made to the app, we deployed the content library separately, using a static, site-generator framework, called *GitHub Pages* [29], which provides completely free hosting and allows healthcare professionals and non-developers to submit content changes, as necessary. *GitHub* hosts the content directly from the repository, is easy to set up, and supports custom domains. *Firebase* also supports secure hosting for web-based apps, microservices, and static and dynamic content. However, *Firebase* only provides a command-line interface and is, therefore, less user-intuitive than *GitHub Pages* for frequent, seamless updates to content, especially for the non-developer members of the research study team.

4.3.6. Redcap

REDCap is a translation-based platform that collects data for research purposes and can be used to store clinical trial data and create/send surveys. The mHealth platform collects data (specifically symptom-related data) from patients that are stored in *Firebase,* which can only be accessed by the platform development team. However, the patients' baseline study data resides on. To ensure that all relevant patient healthcare data for the study are standardized and are accessible to all project teams, we leverage the *REDCap* API to import data from other components of the mHealth platform.

4.3.7. Restful API for the last layer

We built a REST API for the cloud infrastructure backend, to reduce the gap between the low-level implementation details of the platform and the distribution of high-level functionalities. This API also assists with the coupling of new components or apps into the system and mitigates the effects of downtime for existing components. We designed a set of web services, as a viable back-end, by embedding an authentication protocol to identify users, a NoSQL DB to track users and apps, and unit tests that address both of the clear-use cases, in addition to edge cases, for each API endpoint. *Resources* are entities that are exposed to the web, such as images, videos, and educational content topics. The service interfaces are based on Unique Resource Identifiers (URIs or endpoints) that identify resources. Each service implements one or more CRUD operations on a resource. The following semantics of the HTTP methods were used to access the resources: a) GET, to access resource states; b) POST, to create a new resource; c) PUT, to modify an existing resource; and d) DELETE, to delete a resource. For more flexibility in the API, we leveraged the REST approach of hypermedia controls [48] to associate one resource with another, which allows for better navigational control between resources.

To simplify the use of APIs across platforms and programming languages, we described the resources in a lightweight, JSON data format, composed of property-value pairs. Furthermore, as a scalable solution, the URIs for resources represent their hierarchies. For example, */symptoms* represents a collection of symptom names associated with breast cancer patients undergoing HT, */educontent* represents the collection of complete educational content, */educontent/{topicKey}* describes specific HT-related content by topic name, */symptoms/{sympName}/{sympDef}* represents information for a specific symptom, and */symptoms/{sympName}/users* lists all users that submitted ratings for a specific symptom. Fig. 2 shows extracted data for the specific symptom resources, including user-submitted ratings in response to a GET request. The thirteen-digit number in the representation is the timestamp, in milliseconds, associated with the submission of a rating by a specific user.

4.4. Implementation of native Android and web applications

Fig. 3 shows the presented architecture, tailored for the EHT case study, illustrating the connections among the various components and indicating the prototype implementations. Google *Firebase* was used as the cloud DB, which was common among both native mobile and web apps. The web app was deployed using *Node.js* as the web server on Amazon web services (*Amazon EC2*), containing the same functionality as the mobile app. The educational content was deployed centrally, using *GitHub* pages to provide easy access to breast cancer-relevant information within the app, including medication and therapy side-


```
"Decreased range of motion in arm on surgery side": {
"XaRoJ6U2O2Q2lFYcTGM1RLAfKHM2" : {
      "1561442778607" : {
        "sympAnswer" : "Extremely",
      },
      "1561478572769" : {
        "sympAnswer" : "Moderately",
      }
    },
    "Y9jjl6Bx6qUs7CcKkEMzXiPSRVp1" : {
      "1550084969953" : {
        "sympAnswer" : "Slightly",
      }
    }
}
```


Fig. 2. JSON representation of a symptom.

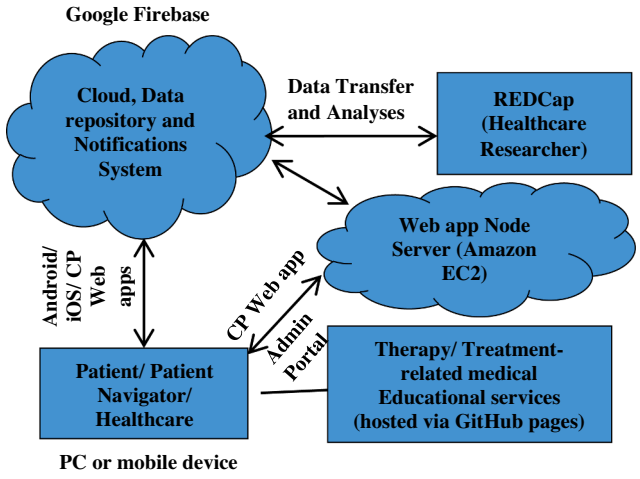

Fig. 3. mHealth architecture, tailored for EHT.

effects, lifestyle tips, and contact information for local healthcare centers.

The system also uses FCM to send push notifications and reminders to patients, directly through the apps. The apps send regular medication reminders, motivational messages, and facilitated direct communications with the medical team, in addition to helping users build support networks. Regular motivational messages and social media group communications can educate and motivate patients, who can share similar experiences with fellow patients and healthcare professionals. The app uses Google account-based authentication. The first time the user accesses the app, she is prompted to input a Gmail ID, which is recorded on the Firebase user data model. PNs can use this ID to send relevant information to the patient, such as emails, reminders, and notifications. *REDCap* is accessible to both healthcare providers and the public health researchers associated with this study, allowing patients to be monitored, based on their submitted symptom data. The system collects health-related data submitted by app users into *Firebase* and transfers the data to *REDCap*.

Due to the integration of heterogeneous components in the mHealth platform, the addition, testing, and release of new features can be complex. The apps utilize a layered architecture pattern [46] that modularizes each app into three layers: presentation (UI), core research or domain logic, and data access, with each layer forming an abstraction around each task that must be completed to fulfill a particular request.

The Android app was built to include multiple activities. Each activity includes one or more features (discussed in Section 4.2) of the app. Various activities correspond with the app's functionality, as follows: clinical use and diagnostic assistance (through symptom-reporting activity, allowing healthcare personnel to review symptoms; personal health record access, based on a weekly symptom summary; and contact activity that facilitates communications with PNs, if necessary); reminders and motivational messages (scheduling of appointments; management of appointments and events through calendar activity; and messaging through push notifications); and easy medical referencing (reference guides and other breast cancer-specialized medical referencing, through educational content and video activity).

The web app runs on a *Node.js* web server, hosted on *AWS EC2*, and can be accessed through a web browser on mobile devices. We leveraged the Progressive Web App (PWA) [49] methodology, which enhances mobile web apps by supporting a more native-like presence on the device's home screen, push notifications (which are a major component of the proposed mHealth system for patient re-engagement), and offline support. Although the web app can be accessed by using the web browser, the user can install the web app on the device. Service workers handle app behavior, when the user is offline or when the app is being used on low-quality networks, by caching the UI and assets (JavaScript or CSS bundles). Web content in the app was adapted to increase accessibility for mobile users and to enhance the user-intuitive features. The apps' web pages correspond to each activity/ feature. We utilized JavaScript (Embedded JavaScript (EJS), Session for secure and persistent logins, and Express and Passport authentication middleware frameworks) as the programming language for the web app, and the node modules were specific to certain functionalities.

The app source code resides on the server, facilitating the easier and quicker application of app updates, without requiring the process of submission and approval that mobile platform-dedicated app stores typically require. The complete web app is a replica of the Android version and performs similarly. The dedicated web pages and client-specific functionality code bases differ between the Android and web versions of the app. The *Firebase* data models, cloud messaging, and educational services functionalities are common to both the Android and the CP web-based apps. To set up the base app, we configured *npm packages*, *node application*, *configuration files* (including *Firebase* authentication-related specifications), database *structure* (based on the Firebase data modeling), and *routes*. *Routes* are the app endpoints, which determine how the app responds to user requests from the client.

4.5. The complete interactive HT patient Helper application design

The apps have been developed through an iterative process, which was heavily influenced by our discussions with the healthcare research team and by usability studies that recruited potential users (i.e., breast cancer survivors who have received EHT).

4.5.1. App mockups

The creation of app mockups was the next stage after the specifications were approved, based on the qualitative information collected, as stated in Ref. [12]. We used Balsamiq Mockups to sketch UI ideas and to create mockups because this software utilizes a simple drag and drop interface for placing graphical elements on a page. Multiple pages/activities in the app can also be linked using graphical elements, such as buttons. The evolving series of mockups served as a walk-through for the complete app and proved to be easier for app development than programming app prototypes. Furthermore, we obtained rapid feedback from the healthcare research team and other involved stakeholders, which led to frequent iterative modifications of the app design before conducting usability studies with the very first version of the functional app prototype.

4.5.2. UI design

The UIs for both the native Android and web apps were designed by considering users' technical abilities, the type of data being collected and displayed, and the various display sizes of mobile devices. Because women who suffer from breast cancer and receive EHT tend to be older, they may have minimal smartphone experience and are unlikely to have previous experience using smartphone apps for the collection of health data or the promotion of adherence to medication regimens for chronic disease treatment. Thus, the use of brighter background images and button colors (as shown in Figs. 4–6) and the incorporation of icons that conveyed the functionality of each button on the home page (as shown in Fig. 4.1), was preferred, to make the app appear attractive and cheerful. Buttons were designed to be larger, contain simple text, and be easier to press on smaller mobile device screens.

The displays have been simplified to reduce the data entry requirements for users. Thus, the intensity ratings for each symptom consists of a slider, which displays a gradual color change from green to red, allowing users to report a range of no side effects (green) to extreme side effects (red), corresponding with a rating scale from 0 to 4 (as shown in Fig. 6.1). This minimizes the users' efforts for describing the intensity of a particular symptom to the simple sliding of a scale bar, instead of requiring text or number entry.

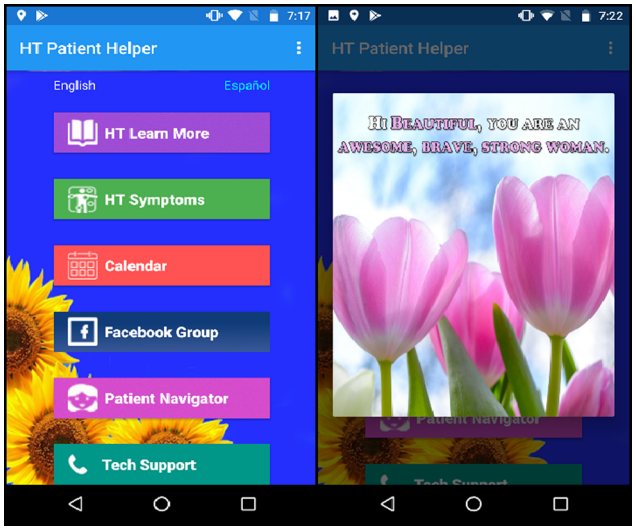

Fig. 4. 1) Menu screen, with access to all the app's features. 2) Received notification screen.

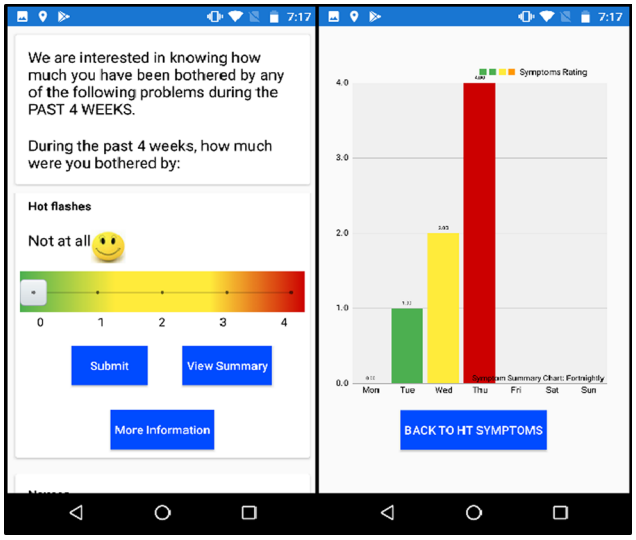

Fig. 5. Symptoms screen, showing 1) submission and tracking and 2) summary.

The degree of informational content that was included for patient education was based on patients' needs. For example, the *More Information* button on the symptoms page (as shown in Fig. 5.1) was designed to show accurate and brief information regarding side effects, whereas the educational content (*HT Learn More → Content)* (as shown in Fig. 6.1) included more in-depth information for those who wanted to learn more.

Thus, during development, the features of the apps were mapped to the following PEF categories: Information and Way-finding, Patient-Specific Education, Patient-Generated Data, e-Tools, Interactive Forms, Integrated Forms, Collaborative Care, and Community Support. These categories supported the following functions of the apps in this research setting.

1. *HT Learn More,* shown in Figs. 4.1 and 6, provided information regarding local support groups, assistance programs, PNs, services (promoting patient-provider communication), technical support, and various patient education information sources and content (text and videos).
2. *HT Symptoms,* shown in Fig. 5 facilitates the entry and tracking of symptoms and allows patients to examine their symptom history and current status.
3. Appointment and medication reminders are provided through push notifications and calendar invites, as shown in Fig. 4.
4. Text messages and notifications reminding patients of daily care, providing motivational messages, and requesting health and symptom assessment (both self-reporting through *HT Symptoms* and interactive online forms, sent as surveys), are sent.
5. The provision of a supportive environment, such as closed peer support groups on Facebook, allowing patients to engage with other patients undergoing therapy and providing a platform for information exchange.

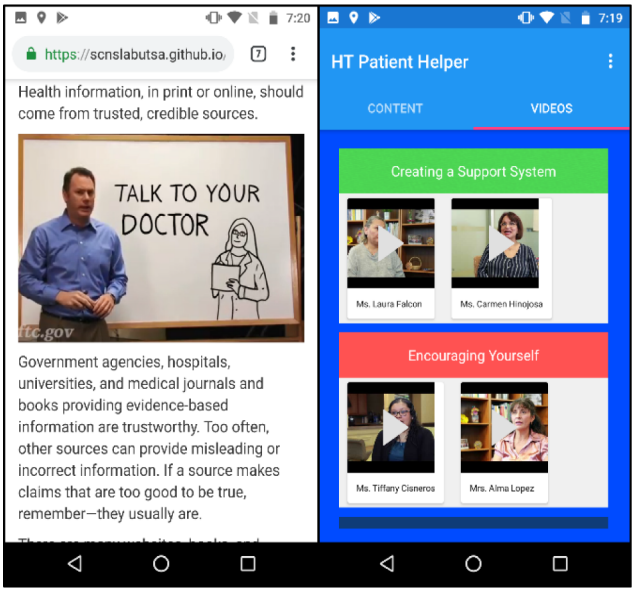

Fig. 6. Educational screens for 1) content and 2) videos.

The features provided in our mHealth solution apps developed for breast cancer survivors receiving EHT, which are based on PEF, include features that support the *Inform me, Engage me, Empower me, Partner with me,* and *Support my e-community* PEF phases. Figs. 4–6 show screenshots of the HT Patient Helper prototype app, including the screens for the main page, received push notifications, symptom tracking, symptom summary, educational content, and videos. Section 5.3.5 describes other changes that were made to the app design, which were influenced and motivated by the feedback received from app users and usability studies.

4.6. Data collection and synchronization

User information, including symptom data, were collected by both apps and stored on *Firebase,* in respective nodes. Synchronization occurs with the *REDCap* project, using the API. Both *Firebase* and *REDCap* provide easy data transfer to a central server/location, allowing data to be synchronized between multiple devices. Data synchronization with *REDCap* is necessary to enable healthcare professionals or PNs to monitor patients during therapy. Scripts are triggered following an update in the patient-specific *Firebase* symptom data to handle conflicts during data synchronization. At a minimum, patients were required to submit symptom information upon receiving a push notification reminder, based on the study schedule. Each submission was modeled with and identified by a timestamp. The *Firebase* project also records the submission logs from the apps, which can be useful for tracking historical patient-specific data (especially when assessing untimely or multiple submissions). This mechanism ensures that updated patient records are readily available for monitoring and analyses by healthcare professionals.

Table 1
Test device specifications.

	Android	iOS
Device	Moto G^5 Plus	iPhone Xs
Operating System	Android 7.0	iOS 12.2
RAM Memory	4 GB	2 GB
CPU	Octa-core, 2 GHz	Hexa-core (2 × 2.5 GHz Vortex + 4 × 1.6 GHz Tempest)

5. Evaluation metrics and results

This section defines the parameters that were used to assess the performance of the implementations that utilized the proposed mHealth architecture and presents the measurement methods, the functionality testing, and the usability evaluation results. Table 1 reports the specifications of the test devices that were used during the performance analysis, to verify the compatibility of the apps when used by different platforms. We reset all test devices to standard factory settings and performed experiments in an isolated environment, without any apps running in the background. The test computer (PC) used to measure the test devices was a 2015 MacBook Pro 15″, running macOS Mojave, with a 2.5 GHz i7 processor and 16 GB 1600 MHz DDR3. Additionally, to create a real-world scenario for functionality testing, we measured response times under conditions of both LTE and Wi-Fi network connectivity. The available bandwidth, uplink, and downlink data rate specifications for the LTE and Wi-Fi networks used were as follows

1. LTE: 12–30 mbit/sec download, 1–5 mbit/sec upload, using AT&T as the service provider.
2. Wi-Fi: 200 mbit/sec download, 10 mbit/sec upload, using Spectrum as the service provider.

5.1. Evaluation criteria

This section describes the measured metrics, including the rationale behind each choice, based on the relevance of each metric to the overall performances of the apps, and how the results of these metrics demonstrate the usefulness of the proposed method.

5.1.1. Installation size

Installation size refers to the disk space required by the installed app on the test device, as well as the size of the APK (the downloadable installer for the Android app). The size of the installer is important because apps can be installed over mobile internet connections, and users may have either limited data plans or limited device storage. Additionally, the small app size indicates the quality of the architecture utilized during the development of the app. Because we considered both web and native Android apps in the current study, we examined the APK size for the Android app on an Android test device and examined web app size on both Android and iOS test devices.

5.1.2. CPU usage

CPU usage refers to the percentage of total CPU capacity that is used by the app during a measured time period. Apps that are CPU-intensive can affect other device processes, which can negatively affect the user experience. This metric also provides insight on potential configuration improvements that can be made to separate modules of the utilized architecture. The current study examined CPU usage during educational video retrieval from *Firebase*, under the *HT Learn More* functionality, to more closely match a real-world scenario.

5.1.3. Memory usage

Memory usage refers to the amount of RAM allocated by the app. If an app allocates a higher percentage of RAM, the performance becomes degraded. The current study considered the following two measurements for this metric: a) the amount of RAM used when the app is completely launched, and b) the maximum RAM usage when navigating each activity/web page, in order of their functionality.

5.1.4. Response times

Response time is an important factor for user experience and is a major contributor to user engagement with self-management mHealth apps. This metric measured the following two scenarios, based on the proposed cloud-based mHealth solution: a) the app's launch time, and b) the time required to retrieve educational videos from Firebase, under the *HT Learn More* functionality. The response time for the second scenario was measured as the difference between the time at which Firebase responded to the user request and the time at which the user requested video retrieval. We also observed variations in response times associated with varying *Firebase* loads because slower response times negatively affect the user experience.

5.2. Metric measurement tools

Table 2 lists the tools used to measure the performance metrics for the web and native Android apps, on each of the test devices.

We executed the "*dumpsys meminfo",* and "*top"* commands, through the Android Debug Bridge (ADB) shell, to perform CPU and memory usage measurements, respectively, for the native Android app on the Android test device. For the web app, we used the Chrome Dev Tools Task Manager. Additionally, we used *Lighthouse* [50] to observe the overall PWA performance, through *audits.*

Chrome Dev Tools and *Lighthouse* were used to measure the launch time for the web app, whereas, for the native Android app, the Android test device automatically logged timestamps when the app was launched. To measure the response times under varying *Firebase* loads, we developed a custom java app and automation shell script to load *Firebase* and simulate user interactions that result in educational video retrieval (Sign in → Open *HT Learn More* → Click Videos tab → Scroll for educational videos). We collected the timestamps that corresponded with the times that requests were sent and the times when the videos were retrieved from *Firebase* and calculated the response time as the difference between the two timestamps.

5.3. Results

This section reports and analyzes the results of the measurement-

Table 2
List of measuring tools for performance analysis.

Metrics	Web App	Android App
CPU Usage	Chrome Dev Tools and Node.js Keymetrics PM2, Android ADB, iOS Instruments	Android studio profiler, ADB (top)
Memory Usage	(Time profiler and Allocations)	ADB (dumpsys meminfo)
Installation Size	Size on test device (Android Chrome and iOS Safari browsers) when PWA web app added to Home Screen	Size on Android test device visible from settings
Response Times	Chrome Dev Tools, Apache JMeter	Android device monitor (DDMS), Custom java app during load on *Firebase*
Loading Firebase Cloud DB	Custom Java app and Shell script for loading *Firebase*	

Table 3
Installation Sizes of Android and Web Apps (in MB).

	Android Test Device	iOS Test Device
Native Android APK Installer (Download size)	7.97	–
Native Android Installed App	36.27	–
Web App	1.04	1

Table 4
CPU Utilization of Android and Web Apps (in %).

	Android Wi-Fi	iOS Wi-Fi	Android LTE	iOS LTE
Native Android App	9	–	11	–
Web App	15.77	11.93	19.32	14.44

based performance study of the mHealth web and native Android apps. We performed each test ten times for each app, and the average values of each measured metric are shown in Tables 4–9. Furthermore, we measured the response times using both LTE and Wi-Fi network connections, under varying loads. For all other measurements, we performed tests in a controlled setting, using either LTE or Wi-Fi and no additional network traffic.

5.3.1. Installation size

Table 3 shows the sizes of the installed apps and the installer APK Android bundle. The size of the web app reflects the memory footprint of the web app when accessed using the Chrome browser, on Android devices, and the Safari browser, on iOS devices. Furthermore, we added the web app to the Home Screen of both browsers. The native Android app consumed more persistent device memory than the web app because the code and files for the native Android app reside in the device's memory, unlike the web app, which uses the PWA methodology, where all the files reside on our server instead of on the user's device. The web app used device storage primarily for the cache, utilizing 1.04 MB and 1 MB of memory on the Android and iOS test devices, respectively, indicating similar memory usage, regardless of the device used. Users are more likely to install apps that utilize less disk space. Thus, reducing the APK download size by publishing the native Android APK as a bundle enables users to download the app, regardless of network connectivity limitations. As discussed, the current study demands feature-rich apps. Thus, the android version consists of additional assets (UI elements), which increase the app size, despite both implementations utilizing the same cloud-based architecture and each functionality being implemented as separate modules and services across the technological components of the system.

5.3.2. CPU usage

CPU usage refers to the percentage of the total available CPU capacity that is utilized (CPU time) when performing a task in the app. In the current study, we measured CPU utilization when navigating through the *HT Learn More* educational video tab and retrieving videos from *Firebase.* CPU profiling works by sampling this process at set intervals. We used 1-millisecond sample intervals. Table 4 shows the CPU utilization, as a percentage of the total CPU capacity, for the native Android and web apps during video retrieval. The web app server logic resides on Amazon EC2, but it utilizes the *Firebase* API to retrieve videos from the *Videos* node, similar to the native Android app. The CPU utilization for the web app was slightly higher than that for the native Android app. Due to the low computational complexity required by the current mHealth platform, the choice of AWS EC2 server configuration was T2.small. While retrieving videos from *Firebase,* the system may not be able to store additional app data, due to a lack of system resources, even when tests were performed in an isolated environment with either Wi-Fi or LTE connectivity. This performance could be improved, however, by optimizing the server configuration and resource use. Performance differences may also be due to differences in the designs of the *Firebase* API for Android and web apps. Wi-Fi is generally more efficient

Table 5
Memory usage of Android and Web Apps (in MB).

	Android Launch	Android Nav Sequence	iOS Launch	iOS Nav Sequence
Native Android App	39.5	81	–	–
Web App	61.11	140.41	55.04	132.02

Table 6
Launch times for the Android and Web Apps (in ms).

	Android Test Device	iOS Test Device
Native Android App	493	–
Web App	304	200

Table 7
Mean response times for the native Android App with LTE and WI-FI modes (in ms).

Load	Android LTE	Android Wi-Fi
0%	121.45	66.51
25%	144.96	101.39
50%	170.32	143.75
75%	234.37	186.49
100%	358.96	286.84

Table 8
Mean response times for the Web App with LTE and WI-FI modes on an android test device (in ms).

Load	LTE	Wi-Fi
0%	163.25	99.49
25%	230.88	203.17
50%	300.96	229.23
75%	400.11	349.37
100%	516.88	460.51

Table 9
Mean response times for the Web App with LTE and WI-FI modes on an iOS test device (in ms).

Load	LTE	Wi-Fi
0%	140.35	78.51
25%	200.62	180.24
50%	290.92	219.23
75%	390.74	309.18
100%	496.88	400.85

for retrieving data than LTE. The web app showed slightly more efficient CPU usage on the iOS device than on the Android device.

5.3.3. Memory usage

Table 5 shows the measured values for memory usage, in MB, when navigating through the apps, based on the order of functionality The following protocol was followed to examine memory use: open home page, open *HT Learn More,* click on *Videos* tab, scroll to the bottom, go back to the homepage, open *HT Symptoms*, submit the rating for the first symptom, and go back to the home page. We measured memory consumption during the app's complete launch and the maximum RAM usage while performing the described navigation sequence. The measured values represent total values, including the memory use corresponding to graphics, code, and stack.

The observed difference in the values for the web app between the Android and iOS test devices is likely due to the different memory management strategies utilized by Android and iOS devices. The web app leverages the iPhone's memory slightly more effectively than the Android's memory. The native Android app has a lower memory requirement than the web app when launched completely.

5.3.4. Response times

A high-quality user experience requires response times to be faster than 100 ms (ms) for mobile apps. Delays greater than a few seconds might degrade the perceived user experience.

5.3.4.1. App launch times. Table 6 shows the launch times for the native Android app, on the Android test device, and for the web app, on both the Android and iOS test devices. The web app, which was designed with the PWA methodology, does not appear to add any overhead and, therefore, required a shorter launch time than the native Android app during tests run under isolated conditions, with no background apps running. However, unlike the native app, the app rendering time for the web app on both test devices is likely to depend on whether the browser is running in the background, due to the dependency of the PWA methodology on the browser.

5.3.4.2. Impact of varying Firebase loads on response times. In this section, we applied increasing loads on the *Firebase* cloud DB. The video data model consists of educational videos that are relevant to breast cancer patients who are undergoing therapy and are enrolled in the study. We performed a delay analysis by calculating response times as the difference between the timestamp associated with the video retrieval request and the timestamp associated with the response from *Firebase*. Initially, we uploaded approximately 3,000 dummy videos, in JSON format, with YouTube links and titles [51] to the *Firebase* video node. We then simulated multiple users and their interactions, based on the navigation sequence designed for this video retrieval scenario. Additionally, we observed user behavior and the effects of retrieval on the UI when the load increased from 0 to 100%, in 25% increments. The UI became sluggish for both the native Android app and the web app (on both Android and iOS devices) with increasing loads. However, the impact varied for both apps, depending on the device. Tables 7–9 show the mean response times for this scenario for the native Android app, the web app on an Android device, and the web app on an iOS device, respectively.

Clearly, for both apps, the response times increased as the cloud DB load increased. However, the web app experienced slightly more sluggish behavior in the UI, due to higher response times, compared with the native Android app, which is likely to degrade the user experience. The CPU utilization (Table 4) and server-cloud communication delays in this scenario for the web app are likely contributors to higher response times. Wi-Fi connectivity was able to retrieve larger chunks of data faster than LTE connectivity when the signal strength was higher.

5.3.5. Usability evaluation

The target users of the apps include the healthcare researchers (through direct interactions), clinicians and oncologists (indirectly through their interactions with the healthcare researchers), and breast cancer patients who have received EHT. The inclusion of these users during the development of the app, including specifications, design, and the integration of the involved technological components, and the performance of usability studies has been crucial to the creation of easy-to-use apps and the facilitation of the overall system. The studies also provided information regarding how to further improve the interface and data flow.

5.3.5.1. User interface. In the very initial stages of the development process, the healthcare research team evaluated the user interface, and they found it to be somewhat non-intuitive. We further refined the apps based on discussions with the healthcare team. The refined version was previously discussed in Section 4.5.2. The healthcare research team was consulted and asked to use the app informally, testing every functionality addition during the development phase. The apps evolved as a result of these testing phases and demonstrated improvements in the ease of use.

For intermediate steps, as in the study protocol in Ref. [12], a total of 8 breast cancer patients were invited for two focus group discussions (2 in English and 2 in Spanish). During the first focus group, the app mockups included various emoticons (ranging from a delighted face to a crying face) corresponding with each rating for the symptoms under the slide and the first version of the app mockups also included single, colored bars under the symptom summary chart. The focus group results revealed that users would prefer if the emotions changed with the movement of the slider, as opposed to the static placement of emoticons. Furthermore, although the summary graph for symptom tracking was reported to be useful, the focus group indicated that the color legend should match the submitted rating for each particular symptom (as shown in Fig. 5.2). The patients appreciated the summary functionality, stating that they could review their health information dating back several years. They also expressed that the information topics available within the app content were important and helpful to them personally. The second focus group was conducted after implementing the suggested improvement from the first focus group, using updated prototypes. Reminders and notification features were added to this version. Patients found app navigation to be more intuitive and they were excited to see the motivational messages and reminders, as text messages. However, they revealed that the messages disappeared quickly and might not be useful if they missed reading them. We made the notification window more intuitive, to allow users more control over the received push notifications. Feedback was also provided regarding the display and arrangement of educational videos, which consisted of tips from doctors for symptom management and the experiences of other patients. Patients expressed interest in linking the symptoms reported on the *HT Symptoms* page with *HT Learn More* videos. Thus, we enhanced user interaction and accessibility by categorizing videos based on the symptoms reported during the intervention.

Additionally, we conducted one internal alpha and one pre-beta round of testing of app functionality, both with 4 breast cancer patients (2 in English and 2 in Spanish), and 5 healthcare researchers (2 in English and 3 in Spanish). The patients were sent invitations to download improved versions of the apps and were asked to use the apps daily, for 2 weeks. In addition, the app manual was developed using graphics and text, with step-by-step information describing how to download the app, key app features, and troubleshooting and contact information for assistance and technical support. At the end of the alpha and pre-beta rounds of testing, the participants were asked to complete the post-study system usability questionnaire (PSSUQ) [52], version 3, based on a 5-level Likert grading scale, with answers ranging from 1 for "*Strongly Disagree*" to 5 for "*Strongly Agree*". We sent these

questions in the form of an online survey through Google forms, to assess the system usefulness (SYSUSE), information (INFOQUAL) and interface quality (INTERQUAL), and overall scores. Approximately 80% of the users marked 5 (*Strongly Agree*) and the other 20% marked 4 (*Agree*) for all of the questions that received higher scores and, thus, indicated higher perceived satisfaction. Overall, the users were pleased with the apps and found them to be usable. They also believed that the apps would enhance their quality of life, improve their ability to monitor reduce the recurrence of the disease, and allow them to educate themselves.

5.3.5.2. System integration and data tracking. The patient data, which was collected through the apps, was made accessible to the healthcare research team on the *REDCap* dashboard to allow researchers to conveniently compare patient responses with medical records (baseline data). During the functional app internal alpha testing trial, the healthcare research team verified the patient-specific self-reported symptom data gathered through the apps, at the backend. Some inconsistencies in data were discovered, which were the result of user misinterpretation. The pattern and frequency of data submissions were random, resulting in multiple rating entries on a single day or at particular times of the day, which invalidated the purpose of the app. For these reasons, the apps now include notes and directions for data entry. Additionally, users receive reminders through push notifications, asking them to submit data on particular times, to ensure more accurate data. These improvements to the UI and backend logic also resulted in more accurate histories of patient progression shown on the symptom summary page during the pre-beta testing phase.

6. Discussion

The presented architecture addresses some aspects and challenges associated with mHealth app design, which can be classified as simplified development, portability, and scalability.

1. Simplified Development: Each service in the presented architecture was built and aligned around a healthcare function, to reduce the complexity of the app feature change-management process. Because each new service can be individually updated or changed, tested, and deployed, without affecting other components, the develop-deploy cycle can be accelerated. Furthermore, by integrating and reusing existing or off-the-shelf software components, such as leveraging *FCM* for push notifications instead of building this functionality from scratch, the creation of prototypes can occur more rapidly.
2. Portability: In the proposed architecture, each component is associated with a service that offers a particular app functionality. For example, reminder messages and notifications are handled through the *Firebase* push notification service and its related API. The boundary of this service (what a specific app feature offers to users) is important for such architecture, allowing a level of portability to be achieved. Moreover, the apps facilitated through this architecture use common components, such as common backend data models and educational services, and interact with web services, ensuring that these features are not tied to a particular platform, which, in turn, improves portability.
3. Scalability: Each app functionality is offered to users through an app home page, where each menu button presents relevant options, based on functionality. New functionality can be added separately to the architecture, either by utilizing the existing software components, through the addition of corresponding logic, or by integrating a new software component associated with its own services, independently. This architecture enables features associated with the different components to act independently, within the complete mHealth system, while bridging these components through loose coupling. Scalability (adapting the boundary of an existing service) aspect is addressed in the case, where we utilized the same AWS web server to extend the functionality of the notifications, allowing specific groups of patients enrolled in the study to be messaged based on their starting date.

Additionally, data interoperability between different system components is important, especially when data is being collected, stored and used. Standardization during the deployment of the architecture is necessary for these scenarios. The case study represents a standalone application for use in a health promotion study. Thus, the presented architecture utilized simple web technologies, JSON schema, and service interfaces, through REST APIs, to standardize the communications between component services that require interactions with the user, without the implementation of standards, such as HL7. Fig. 2 shows a JSON representation of the symptom for the *RESTful* API, in which *Firebase* utilizes JSON by default, and *REDCap* records can be updated with the same schema.

Thus far, two focus groups and two internal rounds of testing have been conducted using these apps, without any reported technical issues, enabling healthcare researchers to gather patient-reported symptom data and to educate, motivate, and encourage the patients, with regards to medication adherence. The data collected from the usability studies, however, have certain limitations that include small sample sizes and the bias and subjectivity associated with evaluating survey responses from the same healthcare research team that has been constantly providing feedback during overall system development. As described in the protocol from Ref. [12], this study will be expanded during future phases to include other oncologists and healthcare professionals of varying levels of expertise and more breast cancer patients, to evaluate the system more objectively. The results of the focus group discussions and the PSSUQ survey showed that, for the most part, the participants were satisfied with the general usability of the apps, in terms of ease of use, intuitiveness, app content, UI (screen navigation), and app usefulness.

From the quantitative data collected during the technical system evaluation, the measured performance metrics did not show significant differences between the native Android and web-based apps that might affect user experience, in the absence of a load on the cloud DB. This finding indicates that for the current level of complexity, the Android and CP web apps performed equally well, despite the inherent architectural differences in their designs. Web apps occupy far less disk space than native Android apps and require shorter launch times. However, increased complexity, increased data retrieval from *Firebase,* and increased DB loads can all affect response times, which, in turn, can affect the user experience, resulting in a poor patient engagement. Larger app installation sizes might cause patients to refrain from installing apps on their devices. Higher CPU and memory usage might result in draining the power of their devices, which, in turn, might result in patient drop out during the intervention. General connectivity issues that cause blank screens (which was later resolved to display a default motivational message) when opening push notifications and the delayed loading of videos were reported, due to poor Wi-Fi or LTE connectivity. Additionally, Table 10 highlights the strengths and limitations of the native and CP web apps, for the current mHealth setting. Table 10 also shows how these two apps can complement each other, by leveraging both *AWS*, as an Infrastructure-as-a-Service cloud server, and *Firebase*, as a Backend-as-a-Service cloud DB. Furthermore, Table 10 highlights the scalability and portability of the proposed architecture, demonstrating how independently deployable services (such as the advanced push notification functionality) can be easily added to the system.

A study [53] examining the overall behavioral performance of apps implemented using 10 common CP development tools showed that apps built using CP frameworks can add overhead and have significant performance penalties, in terms of CPU and memory usage and response times. The process of developing CP versions of apps using these frameworks is not completely straightforward and requires setting up

Table 10
Strengths and limitations of native Android and Web App prototype implementations.

Apps	Strengths and limitations
Native Android only	1. All system features from Section 4.2 with access to native HW features. 2. *Firebase* Push notifications for selected users/ groups and token management without a server is cumbrous and requires an app server for flexibility. (Less scalable)
Cross-platform Web App only	1. All system features from Section 4.2, with access to limited native HW features. With PWA, native experience is similar to native apps, easily scalable, agile, open and flexible, handles selective *Firebase* push notifications. 2. PWA service worker feature however, are not available completely across all CP browsers. 3. REST API integration
Both	1. All system features with access to capabilities of mobile devices and access to device computing resources. 2. Easily scalable, agile, open and flexible, handles selective *Firebase* push notifications. 3. Adds the ability to dynamically shift responsibilities between mobile and cloud with increasing complexity of the system and DB.

platform-specific SDKs, editing configuration files, and adding relevant plug-ins. This study, however, did not consider network conditions, mobile-cloud communications, or context-aware functionality (push notifications, access to native HW features, offline access). Developing apps using CP frameworks also requires specific skillsets, beyond those required for standard web technologies, and prominent mobile platform-specific knowledge (Android and iOS) can be cumbrous, even for simpler functionality apps, requiring longer develop-deploy cycles.

7. Conclusions and future work

The purpose of this study is to present a cloud-based mHealth system designed to provide chronic condition supportive care functionalities. The architecture used to develop this solution allows the addition of new modules or services over time, facilitating the development of a wide range of healthcare features. This architecture has been presented to address the challenges of portability and scalability in mHealth scenarios. We described how to use available software components (*Google Firebase, Amazon AWS, REDCap, GitHub*) and related services to create an integrated mHealth platform and app prototypes, sufficiently fulfilling the requirements necessary for a case study research setting.

In addition, usability and functionality were assessed, for the use of the mHealth apps during research studies aiming to improve chronic disease medication adherence. Specifically, we developed a cloud-based mHealth architecture, with bilingual, interactive CP mobile apps, that was implemented in the EHT setting to empower patients to self-monitor and manage their disease and symptoms, ensuring the development of a scalable and evidence-based intervention. Our architecture is validated and tailored based on the study protocol described in Ref. [12]. The apps that we developed for Android and PWA are nearly identical in appearance and functionality. Building UIs with standard, native-specific, and web-based technology greatly eases the efforts of researchers looking to assess new technology, by leveraging existing resources. Moreover, the components of the system, which offer notification or messaging services, patient-specific educational content, and video library services, can be re-used for different mHealth chronic disease-based supportive care systems. However, most of the UI design and implementation methods used here are generic, and could be easily used to customize this architecture for similar research settings. Most of the app content (user, notifications, videos, symptoms and configuration settings data) is present as data models on the *Firebase* centralized cloud DB, and educational content on *GitHub* can be transferred to similar settings, with minimal effort. Our choices of database (*Google Firebase),* content library hosting platform (*GitHub),* and web server (*AWS)* provided all the necessary functionality for this proof-of-concept study. However, the complete functionality and services corresponding to features could be adapted on a single cloud platform, for easier maintenance.

In future studies, we plan to evaluate the performance of the native iOS app that utilizes the presented mHealth architecture. A future extension to the architecture may also be the addition of a mobile device's virtual assistant to cater to the needs of users with different learning abilities and assist them in navigating through apps. We also intend to investigate non-cloud-based solutions and other cloud providers in comparison with the current implementation. Future work can also include evaluation of the effects of geographical locations, varying network bandwidth, increasing traffic, other performance indicators, and battery conditions on the app performance.

CRediT authorship contribution statement

Devasena Inupakutika: Investigation, Methodology, Software, Data curation, Writing - original draft, Writing - review & editing. **Sahak Kaghyan:** Conceptualization, Data curation, Supervision. **David Akopian:** Conceptualization, Supervision, Project administration, Writing - review & editing, Funding acquisition, Resources. **Patricia Chalela:** Conceptualization, Supervision, Project administration, Funding acquisition, Resources, Validation. **Amelie G. Ramirez:** Conceptualization, Supervision, Project administration, Funding acquisition, Resources.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The study is supported in part by Susan G. Komen (Award No. SAB160005), the Mays Cancer Center (Grant No. P30 CA054174) and Redes En Acción (Grant No. U54 CA153511).

References

[1] Global mHealth Market, Global mHealth Apps Market Will Reach USD 111.1 Billion By 2025: Zion Market Research, 2019 (online), available: https://www.globenewswire.com/news-release/2019/01/24/1704860/0/en/Global-mHealth-Apps-Market-Will-Reach-USD-111-1-Billion-By-2025-Zion-Market-Research.html (accessed August 22, 2019).
[2] C. Tran, A.P. Dicker, H.S.L. Jim, The Emerging Role of Mobile Health in Oncology, J. Target. Therapies Cancer (2017).
[3] G. Eysenback, What is e-health? J. Med. Internet. Res. 3 (2) (2001), https://doi.org/10.2196/jmir.3.2.e20.
[4] C.L. Ventola, Mobile devices and apps for health care professionals: uses and benefits, P T 39 (5) (2014) 356–364.
[5] P.R. Sama, Z.J. Eapen, K.P. Weinfurt, B.R. Shah, K.A. Schulman, An evaluation of mobile health application tools, JMIR mHealth uHealth 2 (2) (2014) e19, , https://doi.org/10.2196/mhealth.3088.
[6] A.S. Mosa, I. Yoo, L. Sheets, A systematic review of healthcare applications for smartphones, BMC Med. Inform. Decis. Mak. 12 (2012) 67, https://doi.org/10.1186/1472-6947-12-67.
[7] G. Castelnuovo, G. Mauri, S. Simpson, A. Colantonio, S. Goss, New technologies for the management and rehabilitation of chronic diseases and conditions, Biomed. Res. Int. 2015 (2015), https://doi.org/10.1155/2015/180436.
[8] N. Kearney, L. McCann, J. Norrie, L. Taylor, P. Gray, M. McGee-Lennon, M. Sage,

M. Miller, R. Maguire, Evaluation of a mobile phone-based, advanced symptom management system (ASyMS) in the management of chemotherapy-related toxicity, Support Care Cancer 17 (4) (2009) 437–444, https://doi.org/10.1007/s00520-008-0515-0.
[9] A.V. Bennett, R.E. Jensen, E. Basch, Electronic patient-reported outcome systems in oncology clinical practice, CA: A Cancer J. Clinicians 62 (5) (2012) 336–347, https://doi.org/10.3322/caac.21150.
[10] P. Klasnja, W. Pratt, Healthcare in the pocket: mapping the space of mobile-phone health interventions, J. Biomed. Inform. 45 (1) (2012) 184–198, https://doi.org/10.1016/j.jbi.2011.08.017.
[11] G. Nasi, M. Cucciniello, C. Guerrazzi, The role of mobile technologies in health care processes: the case of cancer supportive care, J. Med. Internet. Res. 17 (2) (2015) e26, , https://doi.org/10.2196/jmir.3757.
[12] P. Chalela, E. Munoz, D. Inupakutika, S. Kaghyan, D. Akopian, V. Kaklamani, K. Lathrop, A. Ramirez, Improving adherence to endocrine hormonal therapy among breast cancer patients: study protocol for a randomized controlled trial, Contemp. Clin. Trials Commun. 12 (2018) 109–115.
[13] Y. Geng, S. Myneni, Patient engagement in cancer survivorship care through mHealth: a consumer-centered review of existing mobile applications, AMIA Annu. Symp. Proc. 2015 (2015) 580–588 (PMID: 26958192).
[14] K.Ç.Serdaroğlu, G.Uslu, Ş.Baydere, Medical drug adherence monitoring with real time activity recognition: an experimental study, in: IEEE International Workshop on e-Health, Pervasive Wireless Applications and Services (eHPWAS 2015) in Conjunction with IEEE Wimob, UAE, 2015.
[15] Bruno M.C. Silva, Joel J.P.C. Rodrigues, Isabel de la Torre, Miguel Lopez-Coronado Diez, Mobile-health: a review of current state in 2015, J. Biomed. Inform. 56 (2015) 265–272, https://doi.org/10.1016/j.jbi.2015.06.003.
[16] Alessia Paglialonga, Alessandra Lugo, Eugenio Santoro, An overview on the emerging area of identification, characterization, and assessment of health apps, J. Biomed. Inform. 83 (2018) 97–102, https://doi.org/10.1016/j.jbi.2018.05.017.
[17] W. Kuijpers, W.G. Groen, N.K. Aaronson, W.H. van Harten, A systematic review of web-based interventions for patient empowerment and physical activity in chronic diseases: relevance for cancer survivors, J. Med. Internet Res. vol. 15(2): e37, (2014), https://doi.org/10.2196/jmir.2281 (PMID: 23425685).
[18] A. Moumtzoglou, M-Health innovations for patient-centered care, IGI Global (2016), https://doi.org/10.4018/978-1-4666-9861-1.ch016.
[19] A. Cox, G. Lucas, A. Marcu, M. Piano, W. Grosvenor, F. Mold, et al., Cancer Survivors' experience with telehealth: a systematic review and thematic synthesis, J. Med. Internet Res. 19 (1) (2017) e11, https://doi.org/10.2196/jmir.6575.
[20] T.L. Davis, R. DiClemente, M. Prietula, Taking mHealth forward: examining the core characteristics, JMIR mHealth uHealth 4 (3) (2016) e97, , https://doi.org/10.2196/mhealth.5659.
[21] M. Erfani, A. Mesbah, P. Kruchten, Real challenges in mobile app development, in: Proc. ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2013, pp. 15–24. doi: 10.1109/ESEM.2013.9.
[22] I. Malavolta, S. Ruberto, T. Soru, V. Terragni, End Users' perception of hybrid mobile apps in the google play store, in: Proc. IEEE International Conference on Mobile Services, New York, NY, 2015, pp. 25–32. doi: 10.1109/MobServ.2015.14.
[23] I. Malavolta, S. Ruberto, T. Soru, V. Terragni, Hybrid mobile apps in the google play store: an exploratory investigation, in: Proc. 2nd ACM International Conference on Mobile Software Engineering and Systems, Florence, 2015, pp. 56–59. doi: 10.1109/MobileSoft.2015.15.
[24] A. Ahmad, K. Li, C. Feng, S.M. Asim, A. Yousif, S. Ge, An empirical study of investigating mobile applications development challenges, IEEE Access 6 (2018) 17711–17728, https://doi.org/10.1109/ACCESS.2018.2818724.
[25] S. Xanthopoulos, S. Xinogalos, A comparative analysis of cross-platform development approaches for mobile applications, in: Proc. BCI, 2013. doi: 10.1145/2490257.2490292.
[26] Z. Sanaei, S. Abolfazli, A. Gani, R. Buyya, Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges, IEEE Commun. Surv. Tutor. 16 (1) (2014) 369–392, https://doi.org/10.1109/SURV.2013.050113.00090.
[27] Google, Firebase, 2018 (online), available: https://firebase.google.com/ (accessed March 2, 2017).
[28] Amazon EC2, Amazon Elastic Cloud Compute, 2017 (online), available: https://aws.amazon.com/ec2 (accessed Feb 13, 2017).
[29] GitHub, GitHub Pages, 2018 (online), available: https://pages.github.com/ (accessed Feb 10, 2018).
[30] P.A. Harris, R. Taylor, R. Thielke, J. Payne, N. Gonzalez, J.G. Conde, Research electronic data capture (REDCap) – a metadata-driven methodology and workflow process for providing translational research informatics support, J. Biomed. Inf. 42 (2) (2009) 377–381, https://doi.org/10.1016/j.jbi.2008.08.010.
[31] HealthInformatics—Point-of-careMedicalDeviceCommunication—Part 10101: Nomenclature, ISO/IEEE 11073-10101:2004(E), 2004.
[32] Health Level Seven (HL7), (1987) (online), available: http://www.hl7.org (accessed July 20, 2019).
[33] M. Ali, A. Mesbah, Mining and characterizing hybrid apps, in: Proc. WAMA International Workshop on App Market Analytics, Seattle, WA, USA, 2016, pp. 50–56. doi: 10.1145/2993259.2993263.
[34] H. Heitkotter, S. Hanschke, T.A. Majchrzak, Evaluating cross-platform development approaches for mobile applications, in: Proc. 8th Int. Conf. Web Inf. Syst. Technol. WEBIST 2012, Lecture Notes in Business Information Processing, vol. 140, 2013, pp. 120–138. doi: 10.1007/978-3-642-36608-6_8.
[35] Phonegap, 2019 (online), available: https://phonegap.com/ (accessed May 15, 2019).
[36] Titanium, 2019 (online), available: https://www.appcelerator.org/ (accessed May 10, 2019).
[37] T. Majchrzak, T.M. Gronli, Comprehensive analysis of innovative cross-platform app development frameworks, Proc 50thHawaii International Conference on System Sciences, 2017, https://doi.org/10.24251/HICSS.2017.745.
[38] M. Latif, Y. Lakhrissi, E. H. Nfaoui, N. Es-Sbai, Cross platform approach for mobile application development: a survey, in: Proc. International Conference on Information Technology for Organizations Development (IT4OD), Fez, 2016, pp. 1–5. doi: 10.1109/IT4OD.2016.7479278.
[39] O.L. Goaer, S. Waltham, Yet another DSL for cross-platforms mobile development, in: Proc. 1st Workshop on the Globalization of Domain Specific Languages, 2013, pp. 28–33. doi: 10.1145/2489812.2489819.
[40] D. Kramer, T. Clark, S. Oussena, MobDSL: a domain specific language for multiple mobile platform deployment, in: Proc. IEEE International Conference on Networked Embedded Systems for Enterprise Applications, Suzhou, 2010, pp. 1–7. doi: 10.1109/NESEA.2010.5678062.
[41] M. Usman, M.Z. Iqbal, M.U. Khan, A product-line model-driven engineering approach for generating feature-based mobile applications, J. Syst. Softw. 123 (2017) 1–32, https://doi.org/10.1016/j.jss.2016.09.049.
[42] J. Perchat, M. Desertot, S. Lecomte, Component based framework to create mobile cross-platform applications, Procedia Comput. Sci. 19 (2013) 1004–1011, https://doi.org/10.1016/j.procs.2013.06.140.
[43] W.S. El-Kassas, B.A. Abdullah, A.H. Yousef, A.M. Wahba, Taxonomy of cross-platform mobile applications development approaches, Ain Shams Eng. J. 8 (2) (2015) 163–190, https://doi.org/10.1016/j.asej.2015.08.004.
[44] W.S. El-Kassas, B.A. Abdullah, A.H. Yousef, A.M. Wahba, Enhanced code conversion approach for the integrated cross-platform mobile development (ICPMD), IEEE Trans. Softw. Eng. 42 (11) (2016) 1036–1053, https://doi.org/10.1109/TSE.2016.2543223.
[45] Node.js., Node.js, 2017 (online), available: https://nodejs.org/ (accessed Feb 10, 2017).
[46] M. Richards, Software Architecture Patterns, O' Reilly Media Inc, 2015.
[47] AWS Cloud Market, Public cloud market, 2019 (online), available: https://www.sdxcentral.com/articles/news/aws-remains-dominant-player-in-growing-cloud-market-srg-reports/2019/02/ (accessed July 20, 2019).
[48] R.T. Fielding, Architectural styles and the design of networkbased software architectures (Ph.D. dissertation), Inf. Comput. Sci., Univ. California, California, Irvine, USA, 2000.
[49] Progressive Web Apps, PWA development, 2019 (online), available: https://developers.google.com/web/progressive-web-apps/ (accessed May 15, 2019).
[50] Chrome Dev Tools Lighthouse, Lighthouse, 2019 (online), available: https://developers.google.com/web/progressive-web-apps/ (accessed May 15, 2019).
[51] Youtube, YouTubeLinksImporter, 2019 (online), available: https://github.com/ytdl-org/youtube-dl (accessed Apr. 10, 2019).
[52] J.R. Lewis, IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use, Int. J. Human-Computer Interact. 7 (1) (2009) 57–78, https://doi.org/10.1080/10447319509526110.
[53] M. Willocx, J. Vossaert, V. Naessens, Comparing performance parameters of mobile app development strategies, in: IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), Austin, TX, 2016, pp. 38–47. doi: 10.1109/MobileSoft.2016.028.