Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2021, 8(10):199-211**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Developing Mobile Applications with Salesforce Mobile SDK: The present work represents a complete manual

Venkat Sumanth Guduru

Venkatguduru135@gmail.com

Abstract: This report is a documentation of the process of creating mobile applications using the Salesforce Mobile SDK. I consider it as a manual which will provide any developer having no previously existed experience in working with Salesforce, with all the necessary knowledge and instruments for developing the powerful, scalable mobile applications which are freely integrated with Salesforce [1]. To begin with, as a contextual background, this report presents a brief overview of the Salesforce Mobile SDK about what it is, its framework, and the key constituents of it. This is followed by a step-by-step guide on how to create the development environment, the SDK and creating a mobile application from scratch. Some of the aspects of the software development methodology focus on best practices in developing mobile applications for creating secure authentication, strategies for synchronizing data to enhance efficiency, and ways of optimizing performance in the applications. Overall, the actual contents of the report and various sample applications that demonstrate the suitable use of the SDK correspond to the set objectives of the report in the best possible manner [1]. Among some of the main findings it was realized that the Salesforce Mobile SDK has not only its benefits in easing in the development process but also great benefits in enhancing the status and functionality of the final app as well as the usability and experience of the end user.

Keywords: Salesforce, Mobile SDK, Mobile Development, Architecture, Pseudocode, Flowchart, Data Synchronization, Authentication, Performance Optimization, Testing.

Introduction

Nowadays, mobile application development is a prime field that is influencing the world of information technology. With the rapid growth in consumer and business smartphone penetration, companies are feeling the urge more than ever to build mobile apps that attract users in some engaging manner and provide a seamless experience for consumers. The increased complexity in mobile application development, especially with cloud platforms, has made it obligatory to have specialized development tools [2]. One of the tools is Salesforce Mobile SDK, which empowers developers to create mobile apps that have the full power of the Salesforce platform. This makes such a kind of SDK an undeniable pivotal resource for developers who would like to build enterprise-grade mobile apps because of the pre-built components, secure authentication mechanisms, and data synchronization features.

It is primarily aimed at serving as a comprehensive guide for developing mobile applications using the Salesforce Mobile SDK. The goal is to equip any developer, new and seasoned, with knowledge and tools to build solid, scalable, and secure mobile apps seamlessly integrated with the Salesforce platform. This will consist of the essential SDK concepts, setting up the development environment in a step-by-step approach, and finally, best practices to build and optimize a mobile application [3]. This report will technically focus on mobile application development using the Salesforce Mobile SDK. The main areas of focus would be SDK architecture, authentication, data synchronization, and performance optimization. It will further give in-depth

coverage on the implementation details with the aid of pseudocode, flowcharts, and architecture diagrams. While the report will be comprehensive, some understanding of the basics of mobile application development and familiarity with Salesforce are assumed. The extent of technical detail herein will be suitable for developers who solely need to improve their skills in the area of mobile applications that integrate with Salesforce.

Understanding Salesforce Mobile SDK

The Salesforce Mobile SDK offers a robust development framework that assists in smoothing out the process of mobile application development. It facilitates developers to create apps connected with the Salesforce platform by providing tools and libraries to build on native, hybrid, and web apps for iOS and Android devices [4]. Finally, using the Salesforce Mobile SDK means that one is in a position to take advantage of the value of the Salesforce platform; namely, access to Salesforce data as well as integration with Salesforce services while at the same time achieving enterprise-level security. They encompass all the factors of development complication with neatly packed modules for authentication, data synchronization and user interface letting the developer focus solely on designing an application that is strong and scalable.

The key components of the Salesforce Mobile SDK include: The key components of the Salesforce Mobile SDK include:

- Smart Store is safe, encoded, and works only when the app is offline; it enables mobile apps to caching data from any app.
- Mobile Sync: A framework to replicate the mobile application's data synchronization with Salesforce CRM for the same.
- OAuth 2. 0 Authentication: This is an authentication mechanism that offers comfort and security in signing an application through incorporation of the identity's services by Salesforce.
- REST API: This consists of set of APIs that provide access to or the ability to interact with the data or services in Salesforce from a mobile application [5].
- It supports both hybrid development – which is done by web technologies including HTML5, CSS and JavaScript – and native development in languages specific to the platform like Swift for iOS and Kotlin for android.

A. Features:

The Salesforce Mobile SDK contains features that ease the development process but at the same time provide secure and high-performance mobile applications:

1. Cross-Platform Support: Software Development Kit provides tools and libraries for developing apps on iOS and Android. Native support with handover to ground-to-platform-specific traits, and yet there is continuity across different devices.
2. Offline Data Access: Through Smart Store, it would be possible to enable offline capabilities so that users could access and manipulate data even offline. This is particularly useful in scenarios where network connectivity is spotty or unreliable.
3. Data Sync: The Mobile Sync engine does data synchronization between the Salesforce cloud and the mobile app. It resolves the conflict and provides a smooth user experience with all data always updated.
4. Secure Authentication: The SDK incorporates OAuth 2.0 authentication, making it absolutely secure for users to log in to the application using their Salesforce credentials, ensuring that sensitive data remains protected and follows enterprise security standards.
5. Customizable User Interface Components: It is associated with readily available, pre-built UI components in the SDK that are easy to individualize towards an application's brand identity and design. It is performance-optimized with consistency in user experience [6].
6. Integration with Salesforce Services: The SDK makes it easier than ever to integrate mobile applications developed with the SDK with Salesforce services like Salesforce1, Chatter, and custom objects. Business applications become quite powerful during such extensions.

B. Use Cases:

The Salesforce Mobile SDK comes in handy when a company needs to extend their Salesforce environment functionality to mobile devices. The most common use scenarios of this solution include:

1. Field Service Applications: Companies whose people need to work in the field can use this SDK to build applications that enable access to customer information, service requests, and inventory data. Field technicians can update records, capture signatures, and synchronize data back to Salesforce from areas with poor connectivity [7].
2. Sales Enablement: The SDK will offer the sales teams mobile applications to be used in looking through real-time data of customers on-the-go, tracking opportunities, and managing sales pipelines. This would increase the productivity and responsiveness of the sales representative, increasing the effectiveness of sales in this way.
3. Customer Engagement: Use these SDKs to build customer-facing applications that enable individualized experiences using Salesforce data. These can include appointment scheduling, order tracking, and customer support among many other capabilities, all fully integrated with Salesforce [8].
4. Event Management: The SDK can be used to build event management applications where attendants can register, get schedules, interact with content, capture attendee information by organizers, and then sync it with Salesforce for follow-up and analysis.

Architecture of Mobile Applications with Salesforce SDK

A. Architecture Overview:

The architecture of mobile applications developed with the Salesforce Mobile SDK is modular, scalable, and secure. It enables smooth integration between the mobile app and the Salesforce backend for effective data management and rich user experiences. Most of the time, it will be based on a client-server model wherein the mobile application acts as the client and the Salesforce platform as the server, with each communicating with the other through a set of well-defined APIs and services [9]. This architecture is going to enable a mobile app to realize the power of Salesforce's CRM, thus bringing forth native or hybrid user experiences on mobile devices.

At a high level, the architecture can be divided into three layers:

1. Client Layer: The client layer is the mobile app itself and may be developed using either native or hybrid technologies. It is responsible for user interface, local data storage, and logic for interaction with the Salesforce backend. This includes modules such as Smart Store for offline data storage, Mobile Sync for data synchronization, and pre-built UI components so that uniformity of user experience is guaranteed [10].
2. Middleware Layer: Controls the communication process between the mobile app and Salesforce. This layer consists of all the Salesforce Mobile SDK components, namely handling the authentication process through OAuth 2.0, making requests to the API via the REST API, syncing data. This layer clearly abstracts the complexity of the interaction with the Salesforce services, thus leaving a developer to concern themselves only with the business logic of the application.
3. Server Layer: The server layer consists of the Salesforce backend, including Salesforce's CRM platform, custom objects, and services like Chatter and Salesforce1. This layer handles the processing and storage of data, as well as the execution of business logic through Apex classes, triggers, and workflows. The server layer also provides the APIs that the middleware layer uses to communicate with Salesforce [10].

The flowchart below illustrates the architectural structure of a typical mobile application utilizing the Salesforce Mobile SDK. It showcases the flow of data and interaction between the different architectures.

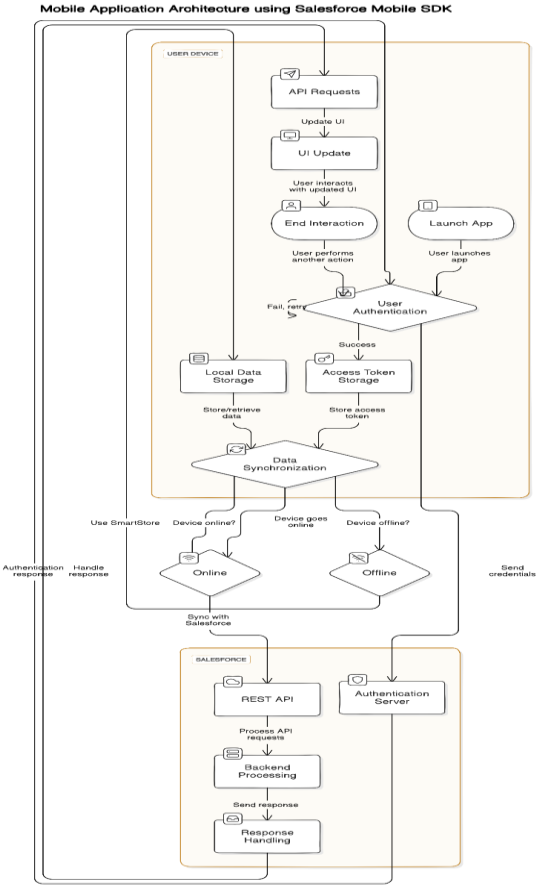

Figure 1: architectural structure of a typical mobile application

1. User Authentication: A user opens a mobile application and is asked for log-in. The mobile app will authenticate the user with Salesforce using the OAuth 2.0 protocol. It gets back an access token to be used in subsequent API requests upon authentication [11].
2. Data Synchronization: An application triggers a data synchronization operation by the Mobile Sync engine. The latter can talk to Salesforce, using the REST API, to fetch or update data. If an application goes offline, data will be stored locally in Smart Store and then synchronized after re-establishment of the connection.
3. Data Storage and Access: The mobile application communicates with the data stored locally in Smart Store. In doing so, this allows the end-user to be able to view and update data in an offline mode. When the device is online, the app syncs data with Salesforce so that the most current information resides on the device and in the cloud.
4. API Requests: Every time the app needs to read/write Salesforce data or call Salesforce services, API requests are sent via the middleware layer. It manages communication with the REST API of Salesforce, processing access tokens and handling errors and retries.
5. Processing on Server-Side: The Salesforce backend processes the incoming API requests for processing and retrieves or updates data based on the request. Any business logical needs are executed. After that, it responds with the server's answer back to the mobile app through the REST API [11].
6. UI Rendering: Processes data returned from Salesforce and refreshes the user interface; it displays records, updates list, or even performs other actions depending on the activity of the user.

B. Components and Interactions:

Several key components interact within this architecture to ensure seamless operation:

1. Smart Store: It is a plug-in that provides encrypted, offline storage to mobile apps. It talks to the local database on the device; hence, it stores Salesforce data when the app is in an offline mode and synchronizes with Salesforce when online [12].

2. Mobile Sync: It's a synchronization engine that takes care of the automatic or manual synchronization of data between the local device and Salesforce. It takes care of the consistency of data and allows for conflict resolution in case of updates done offline.
3. OAuth 2.0 Authentication: This component manages secure user authentication, ensuring that nobody can access the Salesforce data without due authorization. It handles the login process and maintains the access tokens used during the API's interactions.
4. REST API: This is the bridge to Salesforce from a mobile app. All communication concerning data retrieval, updating, and interaction with services on Salesforce lies in the hands of this component. The REST API acts as a gateway for apps so that the powerful features and data existing within Salesforce can be utilized [12].
5. Salesforce Backend: The backbone houses Salesforce's cloud-based CRM platform, which does process on the data and business logic and provides integration APIs. It interacts with middleware to address the requests from the mobile app and returns the relevant responses.

Development Process

A. Setting Up the Environment:

Setting up the development environment of the Salesforce Mobile SDK involves a couple of steps. The SDK provides several tools and libraries for developing mobile applications, from native apps in either iOS or Android to hybrid ones using Web technologies.

Step 1: Install Prerequisites

Before setting up Salesforce Mobile SDK, you need to install the following tools:

1. Node.js: An external prerequisite that is required to run Salesforce Mobile SDK commands.
2. npm: Node Package Manager – Comes with Node.js; it is used to install the Salesforce Mobile SDK CLI.
3. Git: It is a version control system, which is needed to clone the SDK repository.
4. Xcode—Required for iOS development. Available only on a Mac.
5. Android Studio— This will be needed for the development of Android.

Step 2: Install Salesforce Mobile SDK CLI

Open your terminal and run the following to globally install the Salesforce Mobile SDK CLI:

```
npm install -g forceios forcereact forcedroid
```

Figure 2: For Installing Salesforce Mobile SDK CLI

These are the commands to create a new project for iOS, Android, and React Native respectively.

Step 3: Create a New Project

Depending on the kind of platform to be developed, create a new project with the following command:

iOS Native App:

```
forceios create
```

Figure 3: Command for iOS Native App

Android Native App:

```
forcedroid create
```

Figure 4: Command for Android Native App

Hybrid App (React Native):

```
forcereact create
```

Figure 5: Command for Hybrid App (React Native)

You will be asked for information such as the name of your app, package name, and organization name, and then it will create the project structure based on the target platform.

Step 4: Configure Salesforce Connected App

In order to have integration with the mobile app, you need to create a connected app within your Salesforce organization. It involves the following steps:

1. Go to Salesforce Setup and search for "App Manager."
2. Next, click on "New Connected App" and enter all the required details.
3. Turn on the OAuth settings so that the application can communicate with Salesforce. In this case, the callback URI will be sfdc://success, and in the OAuth Scopes, you can select api, refresh_token and offline access.
4. Save the Connected App. This will generate a Consumer Key and Consumer Secret which will be required on the authentication process.

Step 5: Integrate Connected App with the Mobile App

Open the project generated with your favorite IDE (Xcode for iOS, or Android Studio for Android), and configure the app to use both Consumer Key and Callback URL from the connected app. This is usually done in either the bootconfig.json or www/config.xml file.

B. Building the Application:

Now the environment is ready, and you may start developing your mobile application. More precisely, the development steps are as follows:

Step 1: Implement Authentication

First, implement the authentication functionality. Using the built-in OAuth 2.0 flow from the Salesforce Mobile SDK, you can authenticate users to obtain an access token for sending API requests.

```
Function AuthenticateUser()
    Start OAuth Authentication Process

    IF Authentication Successful THEN
        Obtain Access Token
        Store Token(accessToken)
    ELSE
        Display Error(auth.error)
    ENDIF
End Function
```

Figure 6: Implement Authentication code

Step 2: Data Synchronization

The Mobile Sync engine ensures the app and Salesforce hold each other's data. Here's a sample of how to implement the process of synchronization in pseudocode.

```
Function SyncData()
    IF User Is Online THEN
        Fetch Data From Salesforce
        Update Local SmartStore With New Data(remoteData)
        Synchronize Changes With Salesforce(SmartStore.getChanges(), "Contact")
    ELSE
        Use Local Data From SmartStore
        Display Data(localData)
    ENDIF
End Function
```

Figure 7: Data Synchronization code

Step 3: CRUD Operations

The next pseudocode shows how CRUD—Create, Read, Update, delete—on Salesforce objects will be conducted within the mobile app.

```
Function CreateRecord(objectType, data)
    Result = SalesforceAPI.create(objectType, data)

    IF Result.Success THEN
        Store Data In SmartStore(result.recordId, data)
    ELSE
        Display Error(result.error)
    ENDIF
End Function

Function ReadRecord(objectType, recordId)
    Record = SalesforceAPI.fetch(objectType, recordId)
    Return Record
End Function

Function UpdateRecord(objectType, recordId, updatedData)
    Result = SalesforceAPI.update(objectType, recordId, updatedData)

    IF Result.Success THEN
        Update SmartStore(recordId, updatedData)
    ELSE
        Display Error(result.error)
    ENDIF
End Function

Function DeleteRecord(objectType, recordId)
    Result = SalesforceAPI.delete(objectType, recordId)

    IF Result.Success THEN
        Remove From SmartStore(recordId)
    ELSE
        Display Error(result.error)
    ENDIF
End Function
```

Figure 8: CRUD Operations code

1. User Authentication Flowchart:

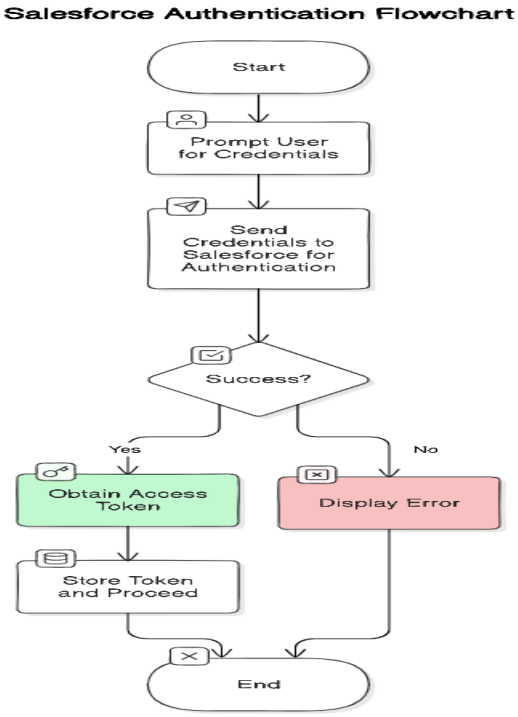

Figure 9: Sales Authentication Flowchart

2. Data Synchronization Flowchart:

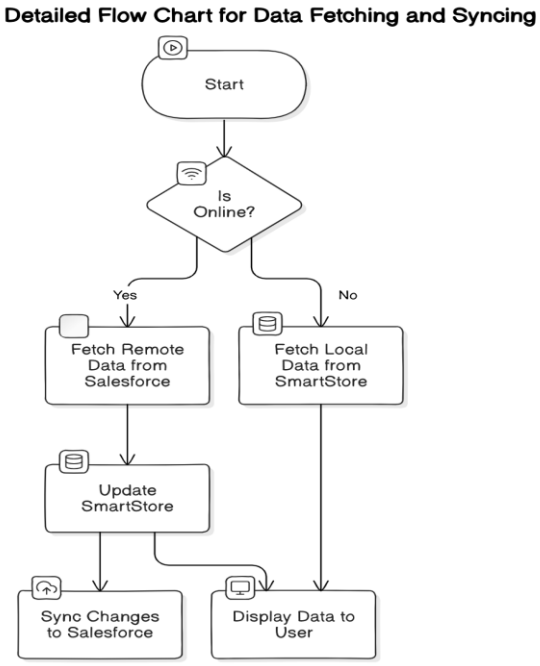

Figure 10: Data Synchronization Flowchart

In the development process of mobile applications, set up a development environment and build key functionalities to ensure that the app is very well integrated to the Salesforce backend using the Salesforce Mobile SDK. The provided pseudocode and flowcharts give the outline of the logical flows of data and user interactions within an app [13]. This allows developers to construct robust, scalable mobile applications.

Advanced Features and Customizations

A. Custom Components:

On the subject of custom components, Salesforce Mobile SDK empowers the developer to stretch further advanced functionality beyond what it has in its standard features. Custom components can be tailored as per any business needs, providing a very unique UI or specialized workflows for the end-user.

You can create a custom component by initiating development for a custom UI element in React Native or native code, say, Swift for iOS and Java/Kotlin for Android. This might be ported into the environment of the Salesforce Mobile SDK.

For instance, in case you need some custom data visualization widget, you could implement it with a charting library like D3.js for React Native or MPAndroidChart for Android. Later on, when the component is built, it will be possible to embed it into the application, hook it up to Salesforce data via the SDK's data services.

```
Function CustomChart(data)
    Return Chart
        Set Data = data
        Set Options
            Type = 'bar'
            Colors = ['#FF5733', '#33FF57']
        ENDSET
    End Chart
End Function

Function Dashboard()
    SalesData = Fetch Sales Data From Salesforce

    Return View
        Use CustomChart With Data(salesData)
    End View
End Function
```

Figure 11: data visualization widget code

B. Integration with Other Services:

The Salesforce Mobile SDK will help in easy integration of the mobile app with external services, such as third-party REST APIs, thereby giving it the functionality to leverage data and functionality from multiple sources.

For example, if you want to integrate the weather forecast into your Salesforce app, it will call a REST API on the weather service and then render the returned data inside your mobile app. In this case, making the HTTP request at hand is facilitated by many of the out-of-the-box networking libraries available in the Salesforce Mobile SDK, such as forcenet.

```
Function FetchWeatherData(location)
    Set URL = "https://api.weather.com/v3/weather/conditions?location=" + location

    Make API Request Using Salesforce Mobile SDK(url, response)

    IF Response.Success THEN
        Set WeatherData = response.data
        Update UI With Weather Data(weatherData)
    ELSE
        Display Error(response.error)
    ENDIF
End Function
```

Figure 12: REST API on the weather service

It is the place where the mobile application fetches weather data for a specified location and refreshes the user interface to show current conditions.

C. Performance Optimization:

Optimizing performance is most important in a mobile application, more so with large data sets from Salesforce. Here are some tips and techniques to ensure smooth performance:

1. Efficient Management of Data: Use Smart Store to store data locally, retrieving it from there to reduce the number of requests to Salesforce. Run sync filters to make sure that only relevant data gets into synchronization with Salesforce [14].
2. Lazy Loading: Implement lazy loading of data-intensive screens; that is, load the data only when it is required—for example, loading a list only while scrolling through it. This will reduce memory usage and increase responsiveness.
3. Caching: It is data that is normally accessed and cached locally via Smart Store or some other local database. This will avoid unnecessary calls to Salesforce and increase the speed of data retrieval.
4. Optimized API calls: Combine as many API calls as possible into a single composite request. Doing so does eliminate multiple HTTP requests' associated overheads, which goes on to have positive implications on the overall performance of an app [14].
5. Minimize Re-renders: A large number of re-renders make your application slow in React Native. Therefore, always use React. Memo and use Callback to avoid extra re-renders of the component.

```
Function MyComponent(data)
    // Component will only re-render if 'data' changes
    Return Text(data)
End Function

Function FetchDataAndRender()
    Data = Fetch Data From SmartStore
    Return MyComponent(data)
End Function
```

Figure 13: REST API on the weather service

Through these optimization strategies, the mobile app will provide smoother, quicker, and more responsive interactions to its users.

Testing And Debugging

A. Testing Strategies:

Testing is a crucial phase in the development lifecycle, ensuring that your mobile application functions as expected across different scenarios and devices. For applications built with Salesforce Mobile SDK, the testing strategies should include unit testing, integration testing, and user acceptance testing (UAT).

- Unit Testing: It is concerned with the independent testing of each component or function. It ensures that every single part of the code does what it is expected to do. At this level, in a Salesforce Mobile SDK environment, it would mean testing custom components, data retrieval methods, or API interactions using such frameworks as Jest for JavaScript-based apps, or XCTest for iOS [15].
- Integration Testing: This helps in making sure that several modules and or services within an application are integrated in the right manner. For instance, you have to assure yourself how your individual UI components call Salesforce data services to verify that the data is effectively transferred between these two elements.

- UAT refers to User Acceptance Testing, which is the final phase of testing where an app is given to the real user to interact with it and validate that the app serves all their needs and expectations. It becomes very critical to validate all the business requirements while giving a seamless user experience through the app.

B. Debugging Tools:

In the development process, there will be issues that have to be identified and fixed. In this respect, efficient debugging tools are important. Most of the standard debugging tools or platforms are pretty well integrated with the Salesforce Mobile SDK.

- React Native Debugger: If you're developing in React Native, this tool will allow inspection of the state of your app, network requests, and debug UI components right from its window. It comes with an integration to Chrome Dev Tools and feels quite familiar while debugging.
- Xcode and Android Studio are the two primary integrated development environments used to develop native iOS and Android apps. They have full-featured debugging tools with support for breakpoints, memory management, and performance profiling. Using these, you can step through your code line by line and examine variables to diagnose issues on real devices or emulators [16].
- Salesforce Mobile SDK logs: The SDK itself provides a significant number of detailed logs that can be used to isolate any issues related to Salesforce data interactions or authentication processes. One can access the logs and analyze them for insights on the flow of data and where problems may potentially occur.

C. Automated Testing:

This will be uppermost in a complex mobile app where the need for automated testing, to ensure the quality and efficiency of code, becomes paramount. Among many other automated testing frameworks, Salesforce Mobile SDK supports the following:

- Jest and Mocha are just a few of the popular choices for any JavaScript-based application. These frameworks let you write test cases for your components, run them automatically to ensure your codebase is stable upon making any changes in it.
- Appium is an open-source, cross-platform mobile automation test framework that helps automate testing for iOS and Android apps. Additionally, it supports Salesforce Mobile SDK and can be used to simulate user interaction, verify UI elements, and test application functionality in a variety of contexts [16].
- Test Framework: Salesforce CLI has a test framework inside the Salesforce CLI that you will be able to integrate with your CI/CD pipeline. This will help you run automated tests of Salesforce-specific functionalities—for instance, data synchronization and authentication—to see that these very critical components work seamlessly.

Such test strategies, supplemented by the correct debugging tools and embedded automated testing, can substantially raise your assurance level for the quality and reliability of your Salesforce Mobile SDK app.

Conclusion

The report has elaborately discussed the development of mobile applications using the Salesforce Mobile SDK. We begin with an overview of mobile application development and the importance of the Salesforce SDK, then delve into the architecture, features, and development processes involved in creating robust mobile applications [17]. All the requisite components have been covered at length, including custom components and their integration with any external service, techniques of performance optimization, and so on. We have looked further at the role of rigorous testing and debugging, including tools and techniques for ensuring a smooth user experience.

Considering the way mobile technology is developing at this point, there exist a lot of possibilities for further enrichment of Salesforce Mobile SDK applications in such areas as deeper integration with AI and machine learning for predictive analytics, enhancing user experience with the provision of more advanced UI/UX components, and the extension of SDK functionality to enable support for emerging technologies like AR and VR. Besides, optimizations that would have allowed running apps on 5G networks to open a wealth of new opportunities in terms of real-time data processing and improved performance. The Salesforce Mobile SDK exposes a solid framework that provides an opportunity to develop qualitative, scalable, and class-leading

mobile applications, which can be integrated into a powerful ecosystem available at Salesforce [17]. Given that businesses are becoming increasingly dependent on mobile solutions, this makes developing and maintaining such applications in an effective manner very important. With these techniques and strategies, it is possible for developers to harness the full potential of the Salesforce Mobile SDK. In the future, the ability to adapt to the changes and new trends in technologies, as well as in software development will be vital to generate effective and useful mobile applications.

Acknowledgment

On our behalf, the authors would want to take this opportunity to appreciate all subjects who have in one way or the other contributed to making this report what it is today. There are also people who cannot be named without gratitude, and those are [Sponsor Name] for their constant support and sponsorship. We would like to thank the Salesforce Developer Community for helping provide us with ideas and other approaches to things. Consequently, it was virtually inconceivable to accomplish this work through individual endeavor and encouragement from peers and advisors. The opinions and recommendations made in this report are those of the authors and not of the organizations that provided the support for this work.

References

[1]. A. Vahdat, A. Alizadeh, S. Quach, and N. Hamelin, "Would you like to shop via mobile app technology? The technology acceptance model, social factors and purchase intention," Australasian Marketing Journal, vol. 29, no. 2, pp. 187-197, 2021.

[2]. D. Jyoti, J. A. Hutcherson, "Salesforce Identity and Access Management Architecture," in Salesforce Architect's Handbook: A Comprehensive End-to-End Solutions Guide, 2021, pp. 223-256.

[3]. Garg, H. (2017). An Introduction to World Leading CRM: SalesForce. com. i-manager's Journal on Cloud Computing, 4(2), 9.

[4]. Davis, A. (2019). Mastering Salesforce DevOps: A Practical Guide to Building Trust While Delivering Innovation. Apress.

[5]. Bae, D., Han, K., Park, J., & Yi, M. Y. (2015, February). AppTrends: A graph-based mobile app recommendation system using usage history. In 2015 International Conference on Big Data and Smart Computing (BIGCOMP) (pp. 210-216). IEEE.

[6]. Samy, D., Hegazy, A., & Kadry, M. (2017, March). Propose New SDLC Practices Model for Mobile Native Application. In First EAI International Conference on Computer Science and Engineering (pp. 25-34).

[7]. Q. He, B. Li, F. Chen, J. Grundy, X. Xia, and Y. Yang, "Diversified third-party library prediction for mobile app development," IEEE Trans. Softw. Eng., vol. 48, no. 1, pp. 150-165, 2020.

[8]. R. E. Saragih, "Development of interactive mobile application with augmented reality for tourism sites in Batam," in 2020 Fourth World Conf. Smart Trends Syst., Secur. Sustain. (WorldS4), 2020, pp. 512-517.

[9]. M. N. Islam, I. Islam, K. M. Munim, and A. N. Islam, "A review on the mobile applications developed for COVID-19: an exploratory analysis," IEEE Access, vol. 8, pp. 145601-145610, 2020.

[10]. A. E. Roberts, T. A. Davenport, T. Wong, H. W. Moon, I. B. Hickie, and H. M. LaMonica, "Evaluating the quality and safety of health-related apps and e-tools: Adapting the Mobile App Rating Scale and developing a quality assurance protocol," Internet Interventions, vol. 24, p. 100379, 2021.

[11]. B. Aljedaani and M. A. Babar, "Challenges with developing secure mobile health applications: Systematic review," JMIR mHealth uHealth, vol. 9, no. 6, p. e15654, 2021.

[12]. E. M. Messner et al., "The German version of the Mobile App Rating Scale (MARS-G): development and validation study," JMIR mHealth uHealth, vol. 8, no. 3, p. e14479, 2020.

[13]. A. Biørn-Hansen, C. Rieger, T. M. Grønli, T. A. Majchrzak, and G. Ghinea, "An empirical investigation of performance overhead in cross-platform mobile development frameworks," Empirical Softw. Eng., vol. 25, pp. 2997-3040, 2020.

[14]. Harrison, R., Flood, D., & Duce, D. (2013). Usability of mobile applications: literature review and rationale for a new usability model. Journal of Interaction Science, 1, 1-16.

[15]. B. Ncube and H. A. Koloba, "Branded mobile app usage intentions among generation Y students: A comparison of gender and education level," Int. J. eBusiness eGovernment Stud., vol. 12, no. 2, pp. 91-106, 2020.
[16]. A. Tashildar, N. Shah, R. Gala, T. Giri, and P. Chavhan, "Application development using flutter," Int. Res. J. Modernization Eng. Technol. Sci., vol. 2, no. 8, pp. 1262-1266, 2020.
[17]. X. Zhang et al., "Screen recognition: Creating accessibility metadata for mobile applications from pixels," in Proc. 2021 CHI Conf. Human Factors Comput. Syst., 2021, pp. 1-15.