Server Driven UI für mobile Anwendungen

Sebastian Koller
Master Interactive Technologies
Masterclass Mobile
FH St.Pölten
St.Pölten, Österreich
it231522@fhstp.ac.at

Abstract

Diese wissenschaftliche Arbeit beleuchtet das Konzept des Server Driven UI (SDUI) in der Softwareentwicklung, insbesondere im Kontext mobiler Anwendungen. Im Zentrum steht die Verlagerung von UI-Logik von der Client- zur Serverseite, was zu flexibleren und schnelleren Anpassungen des User Interfaces führt, ohne aufwendige App-Releases. SDUI wird nicht als Framework, sondern als Design Pattern betrachtet, und ermöglicht eine konsistente User Experience über verschiedene Plattformen. Die Implementierung von SDUI, bei der das Layout dynamisch vom Server vorgegeben wird, wird anhand von Beispielen aus der Praxis und großen Technologieunternehmen wie Spotify und AirBnB veranschaulicht. Der begrenzte Forschungsstand wird durch positive Anwendungsbeispiele und das Interesse großer Unternehmen ausgeglichen, was auf vielversprechendes Forschungspotenzial hinweist. Der Ausblick in die Zukunft umfasst Optimierungen der Implementierungsmethoden, eine breitere Integration in verschiedene Branchen sowie die Bewältigung von Herausforderungen wie Abhängigkeit vom Backend und erhöhten Kosten. Insgesamt deutet diese Arbeit auf eine vielversprechende Zukunft für SDUI hin, die die Gestaltung und Aktualisierung von User Interfaces grundlegend transformieren könnte.

I. Einleitung

Das Problem, dass bei der herkömmlichen Entwicklung von Apps besteht, ist die Release Zeit. Für eine Änderung muss eine neuer Build erstellt werden, damit dieser dann bei den jeweiligen App Stores für die Überprüfung eingereicht wird. Bei Apple kann dieser Prozess bis zu einem Tag dauern [1]. Bei zeitkritischen Änderungen kann das jedoch deutlich zu lange sein. Grund genug, um über eine andere Art der Architektur nachzudenken.

Server Driven UI ist ein Konzept in der Softwareentwicklung, dass immer mehr Bedeutung für die Entwicklung von mobilen Anwendungen gewinnt. Der Kerngedanke ist dabei, Logik von der Client Seite auf die Server Seite auszulagern. Es begünstigt eine Konsistenz über mehrere Systeme, wie Web, iOS und Android. Es handelt sich dabei aber nicht um ein Framework, sondern um ein Design Pattern für die Architektur von Softwaresystemen, die auf unterschiedliche Weise implementiert werden kann [2].

II. Einführung SDUI

A. Herkömmliche UI Erstellung

Die Entwicklung einer mobilen Anwendung besteht aus zwei Komponenten: dem Frontend und dem Backend. Beim herkömmlichen Ansatz werden im Frontend die verschiedenen Komponenten, aus denen das User Interface besteht, angelegt, erstellt und statisch oder teilweise dynamisch in einem Layout platziert [3]. Eine Login Seite könnte in Dart demnach so aussehen:

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Login'),
    ),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        mainAxisAlignment:
             MainAxisAlignment.center,
        children: <Widget>[
          TextField(
            decoration:
             InputDecoration(labelText:
                          'Benutzername'),
          ),
          TextField(
            decoration:
             InputDecoration(labelText:
                          'Passwort'),
            obscureText: true,
          ),
          ElevatedButton(
            onPressed: () => _anmelden(),
            child: Text('Anmelden'),
          ),
        ],
      ),
    ),
  );
}
```

Listing 1: Dart Code einer Login Page [4]

Die _anmelden() Funktion würde einen Request an das Backend durchführen, um zu überprüfen ob die Daten korrekt sind, um damit die User*in zu authentifizieren.

B. Umsetzung mit SDUI

Beim Server Driven UI Ansatz wird das Layout nicht so statisch, wie im obigen Beispiel vordefiniert, sondern

wird vom Server vorgegeben. Navigiert ein*e User*in beispielsweise auf die login.dart Seite, wird zuerst ein Request an den Server geschickt. Dieser Retourniert dann eine Art Anleitung, die dem Frontend vorgibt, wie die Seite aufzubauen ist. Voraussetzung dafür ist, dass die Komponenten im Frontend darauf ausgelegt sind, anhand der Daten vom Server generiert zu werden [5]. Im Beispielcode ist eine Column, die als Kinder eine Liste von Widgets bekommt. Welche Kinder das sind, könnte zum Beispiel der Server vorgeben. Eine Response könnte demnach so aussehen:

```
{

  "FormFields": [
    {
      "widget": "TextField",
      "decoration": {"labelText":
             "Benutzername"}
    },
    {"widget": "SizedBox", "height": 20},
    {
      "widget": "TextField",
      "decoration": {"labelText":
                          "Passwort"},
      "obscureText": true
    },
    {"widget": "SizedBox", "height": 20},
    {
      "widget": "ElevatedButton",
      "onPressed": "anmelden",
      "child": {"widget": "Text",
                    "data":"Anmelden"}
    }
  ]
}
```

Listing 2: JSON API Response als UI Vorgabe [4]

Wie man sieht, korreliert die Vorgabe im Json 1:1 dem statischen Frontend Code. Auf dieselbe Weise könnten auch die anderen Widgets auf der Seite, wie die AppBar oder die Column selbst vorgegeben werden.

Für die Definition der anmelden-Funktion kann man verschiedene Herangehensweisen wählen. Entweder man bleibt bei der eher herkömmlichen Umsetzung im Frontend, was dem Server Driven UI Ansatz aber etwas Flexibilität nimmt, da alle Funktionen, die das Frontend können muss bei App-Release bereits vordefiniert sein müssen. Anders wäre es, wenn die onPressed Methode standardmäßig einen Backend Request sendet, und in diesem Request werden die Inhalte der Formularfelder, sowie ein Methodenname, in dem Fall „anmelden“, mitgeschickt, sodass das Backend die ganze Logik enthält. AirBnB hat zum Beispiel eine generische Methode im Frontend, die die User*in beim Click auf eine Seite bringt. Dieser Methode muss eine ID übergeben werden. Der Server schickt dann die Anleitung einen Navigationsbutton zu erstellen, dieser soll beim Click diese generische Methode aufrufen und die benötigte ID kommt ebenfalls vom Server [3].

Die Umsetzung einer Server Driven UI muss nicht bei Entwicklungsstart beginnen. Das Konzept kann Schritt für Schritt implementiert werden, indem man zuerst einzelne Komponenten umschreibt, oder man Features, die neu hinzukommen, mit dem Design Pattern umsetzt [2].

III. Forschungsstand

Nach einer Recherche über den Forschungsstand lässt sich feststellen, dass wenige wissenschaftliche Arbeiten zu dem Thema veröffentlicht wurden.

Kumar verfasste 2021 einen Artikel über die Reduktion der App Größe bei Android Applikationen, in der er, neben dem Nachladen von Bildern und Sprachfiles, auch die Verwendung von Server Driven UI diskutiert und schließlich vorschlägt. Neben den Problemen, für die er einfache Lösungen vorschlägt, werden vor allem die Vorzüge, nämlich das dynamische Darstellen von Screens und dem Umgehen eines neuen App Releases bei Änderungen, dargestellt.

Eine detailliertere Untersuchung wurde von Carvalho und Papst 2022 durchgeführt. Sie betrachteten den Nutzen von Server Driven UI in Bezug auf E-Commerce Anwendungen, wofür sie eine Bibliothek und ein Prototyp entwickelten. Dabei stießen sie auf das Problem, dass sich der Programmieraufwand drastisch steigerte, sobald bestehende Funktionalitäten um neue Funktionen erweitert wurden, da aufgrund der Komplexität der Architektur viele Codestellen geändert werden mussten. Die Bibliothek testeten sie mit fünf Expert*innen aus dem Bereich der mobilen Entwicklung. Obwohl nur einer der fünf praktische Erfahrungen mit Server Driven UI hatte, wurde der Großteil der Fragen zur Komplexität und Sinnhaftigkeit positiv bewertet. Sie merkten an, dass eine verständliche Dokumentation mit Beispielen bei der Verwendung einer Bibliothek für Server Driven UI notwendig wäre.

Trotz des geringen Forschungsstandes zeigt die Verwendung von Server Driven UI bei großen Technologieunternehmen, dass ein hohes Forschungspotential besteht. Spotify und AirBnB entwickelten beide eigene Implementierungen von Server Driven UI, beide betonen dass sie damit schneller und flexibler auf die Daten der User*innen reagieren können [3], [6].

A. Abgrenzung zur Model Driven UI

Da Model Driven UI ein Begriff ist, auf den man schnell bei der Recherche nach Server Driven UI stößt, wird er im Folgenden abgegrenzt. Die beiden klingen zwar ähnlich, stehen aber für zwei relativ gegensätzliche Paradigmen. Im Gegensatz zu hier beschriebenen Server Driven UI liegt die Kontrolle über die UI bei der Model

Driven UI beim Client, also im Frontend. Am Client befindet sich ein Model, das zum Beispiel eine UI-Komponente repräsentiert. Die UI wird aufgrund dieses Models erstellt und auch bei Änderungen des Models aktualisiert [7]. Das Model einer Aufgabe in einer Todo-App könnte zum Beispiel aus den Feldern „id“, „titel“ und „erledigt“ bestehen, aus denen als Komponente eine Liste mit Checkboxen erstellt wird. Beim Click auf eine Box wird das Model geupdatet, „erledigt“ auf true gesetzt, wodurch der Text durchgestrichen wird. Das Model steuert also die UI. Wie bei der Server Driven UI gibt es auch hier den Vorteil, dass ein Model für verschiedene Apps, z.B. Android und iOS, genutzt werden kann, jedoch benötigt es für Änderungen im Model oder der UI einen neuen App Release [8].

IV. SDUI – Was ist zu beachten

So wie bei jedem neuen Design Pattern gibt es auch bei Server Driven UI Punkte, die bei der Entscheidung berücksichtigt werden sollten, ob das Konzept das richtige für das jeweilige System ist.

Der Hauptgrund, weswegen man sich überlegen sollte, Server Driven UI in einem Projekt zu implementieren, ist die Möglichkeit, Änderungen im User Interface umzusetzen, ohne einen neuen App Release zu erstellen. Dieser müsste nämlich erst einen Review Prozess der App Stores durchlaufen werden, bevor die Updates veröffentlich werden. Kann man das umgehen, erspart man sich kostbare Zeit und Mühe und man kann viel schneller auf aufgetretene Probleme reagieren [5].

Des Weiteren trägt Server Driven UI zu einer erhöhten Applikationssicherheit bei. Die Daten können validiert werden, bevor sie an das Frontend geschickt werden, wobei sichergestellt wird, dass die Daten vertrauenswürdig sind und nicht durch Eingriffe beeinflusst wurden. Das ist vor allem bei persönlichen Daten oder Finanzdaten besonders wichtig. Zu guter Letzt ist der Server in der Lage, schnell auf Sicherheitsprobleme zu reagieren und den Zugriff zu blockieren, ohne dass eine neue App Version releast werden muss [5].

Durch das dynamische Laden der UI wird die schnelle Umsetzung von A/B-Tests ermöglicht. Außerdem können personalisierte Oberflächen, die auf Benutzer*inneneigenschaften wie Standort oder Sprache basieren, erstellt werden, was das User Experience und die Barrierefreiheit verbessert [9]. Server Driven UI fördert durch die Einrichtung einer "Single Source of Truth" die Wiederverwendbarkeit von Code über verschiedene Plattformen hinweg, wodurch Redundanzen reduziert und der Wartungsaufwand vereinfacht wird [5].

Nicht zu unterschätzen ist der gesteigerte Programmieraufwand. Vor allem wenn man beabsichtigt, ein bestehendes System vollständig zu refrakturieren, um den Vorgaben einer Server Driven UI zu entsprechen. Der zusätzliche Personalaufwand fokussiert sich dabei sowohl auf das Backend als auch auf das Frontend, da eine neue UI-Komponente von einem Feature in beiden Enden des Systems angelegt werden muss, was für doppelten Aufwand sorgt [10]. Abes es muss nicht alles auf einmal umgestellt werden. Schritt für Schritt neue und alte Features auf den neuen Ansatz umzustellen, verteilt den Programmieraufwand auf eine längere Zeitspanne, wodurch eine Umstellung des Systems realistischer wird [2].

Ein weiterer Punkt ist die Abhängigkeit vom Backend und damit von einer Internetverbindung. Natürlich besteht dieses Problem teilweise auch ohne Server Driven UI, doch mit ihr ist es allgegenwärtig. Ein Lösungsansatz ist es, die Responses vom Server zwischenzuspeichern, zu *chachen*, wodurch das Frontend auch ohne Internet etwas anzeigen kann [11]. Problematisch können die längeren Wartezeiten werden, da viel mehr mit dem Server kommuniziert wird und die Größe der Responses umfangreicher ist. Eine langsame Internetverbindung kann hier zu einem deutlichen Usability Problem werden, vor allem bei zeitkritischen Anwendungen wie Spiele oder Nachrichtendienste.

Mit den gesteigerten Anforderungen an den Server kommen auch erhöhte Kosten ins Spiel. Die häufigeren Requests bedeuten eine höhere Auslastung und benötigen eine bessere Erreichbarkeit. Allerdings sollte man sich bewusst sein, dass durch die langfristig gesehen geringere Entwicklungszeit diese Kosten wieder eingearbeitet werden. Hinzu kommen die Kosten für die Erweiterung des Teams an Entwickler*innen [10].

V. Fazit

Die Einführung von Server Driven UI (SDUI) markiert einen bedeutenden Schritt in der Evolution der mobilen Anwendungsarchitektur. Durch die Verlagerung von Logik auf die Serverseite ermöglicht SDUI eine flexiblere und schnellere Anpassung des User Interfaces ohne die Notwendigkeit eines zeitaufwändigen App-Releases. Dieser paradigmatische Ansatz, der nicht auf ein spezifisches Framework beschränkt ist, sondern als Design Pattern fungiert, verspricht eine konsistente User Experience über verschiedene Plattformen hinweg. Der vorgestellte Ansatz, bei dem das Layout dynamisch vom Server vorgegeben wird, zeigt sich besonders effektiv und praktikabel, wie durch die Implementierungen von großen Technologieunternehmen wie Spotify und AirBnB demonstriert wird.

VI. Ausblick

Der Forschungsstand zu Server Driven UI ist zwar noch begrenzt, doch die positiven Anwendungsbeispiele und das Interesse großer Unternehmen deuten auf ein vielversprechendes Forschungspotenzial hin. Zukünftige Entwicklungen könnten sich auf die Optimierung von Implementierungsmethoden konzentrieren, um den Programmieraufwand zu minimieren und die Anpassungsfähigkeit weiter zu verbessern. Die Integration von SDUI in verschiedenen Branchen, über die bisher erforschten Anwendungsbereiche hinaus, könnte die Vielseitigkeit und Relevanz dieses Architekturkonzepts weiter stärken. Die Herausforderungen hinsichtlich der Abhängigkeit vom Backend und erhöhten Kosten erfordern fortlaufende Untersuchungen, um innovative Lösungen zu entwickeln und die Anwendungsbereiche von SDUI zu erweitern. Insgesamt zeichnet sich eine vielversprechende Zukunft für Server Driven UI ab, die nicht nur die Entwicklung von mobilen Anwendungen, sondern auch die Art und Weise, wie wir User Interfaces gestalten und aktualisieren, grundlegend transformieren könnte.

VII. Referenzen

[1] A. Inc, „App Review - App Store“, Apple Developer. Zugegriffen: 22. November 2023. [Online]. Verfügbar unter: https://developer.apple.com/app-store/review/

[2] „Server-driven UI basics“, Apollo Docs. Zugegriffen: 22. November 2023. [Online]. Verfügbar unter: https://www.apollographql.com/docs/technotes/TN0032-sdui-basics/

[3] R. Brooks, „A Deep Dive into Airbnb’s Server-Driven UI System“, The Airbnb Tech Blog. Zugegriffen: 22. November 2023. [Online]. Verfügbar unter: https://medium.com/airbnb-engineering/a-deep-dive-into-airbnbs-server-driven-ui-system-842244c5f5

[4] Y. Mittal, „Server-Driven UI vs Traditional Flutter UI Development: Which is Right for Your Project?“, Medium. Zugegriffen: 25. November 2023. [Online]. Verfügbar unter: https://codecooker.medium.com/server-driven-ui-vs-traditional-flutter-ui-development-which-is-right-for-your-project-bf61d13f47dc

[5] R. Honório, „Server-Driven UI: the Ultimate Solution for Mobile Apps?“, Medium. Zugegriffen: 25. November 2023. [Online]. Verfügbar unter: https://betterprogramming.pub/server-driven-ui-does-it-solve-everything-for-mobile-apps-4b38397b2e04

[6] „Backend-driven native UIs“, At Scale Conferences. Zugegriffen: 22. November 2023. [Online]. Verfügbar unter: https://atscaleconference.com/videos/backend-driven-native-uis/

[7] P. A. Akiki, A. K. Bandara, und Y. Yu, „Adaptive Model-Driven User Interface Development Systems“, *ACM Comput Surv*, Bd. 47, Nr. 1, Mai 2014, doi: 10.1145/2597999.

[8] A. Pleuß, S. Wollny, und G. Botterweck, „Model-driven development and evolution of customized user interfaces“, S. 13–22, 2013, doi: 10.1145/2494603.2480298.

[9] C. M. E. T. Blog, „Server-Driven UI — Concept“, Medium. Zugegriffen: 25. November 2023. [Online]. Verfügbar unter: https://medium.com/@dfs.techblog/server-driven-ui-concept-db07d7946e94

[10] G. F. de Carvalho und N. do V. Papst, „Aplicação de server driven UI para interfaces mobile ios de e-commerce“, 2022.

[11] R. Kumar, „Android App Size Reduction: Analysis and different methodology“, in *2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Sep. 2021, S. 1–5. doi: 10.1109/ICECCT52121.2021.9616819.