Overview on Composable UI

نگاهی کلی به رابط کاربری افزودنی

حسین خادمیان[۱]

[۱] دانشجوی کارشناسی، مهندسی کامپیوتر، دانشگاه شیراز ، شیراز، me@hkhademian.ir

۱۳۹۹/۰۴/۱۸

چکیده

در این مقاله، به معرفی روابط کاربری ترکیب پذیر ، مزایا و معایب این سیستم، علت چرخش شرکت های بزرگ فناوری به استفاده از این شیوه ساخت روابط کاربری و همچنین نگاه و طریقه آنها در پیاده سازی می پردازیم.

کلمات کلیدی

UI, Composable UI, Flutter, Jetpack Compose, Swift UI, React JS, React Native.

۱ مقدمه

شیوه کنونی تولید روابط کاربری بر مبنای تعریف عناصر صفحه صفحه نمایش به صورت اشیایی شناخته شده و با جداسازی widget های گوناگون مانند Button، ListView، DropBox، CheckBox، ... استوار است.

این شیوه توصیف لایه های کاربری در پلتفرم های گوناگون، جهت پیاده سازی رابط کاربری برنامه ها در سیستم عامل ها دستگاه های مختلف (از جمله کامپیوتر شخصی، تبلت، موبایل، صفحات وب و ...) پیاده سازی می شد.

در این شیوه معمولا بخشی جهت اعمال تغییرات در بخش لایه کاربری مثل Editor Layout وجود دارد که در کنار کد منطق برنامه این لایه های کاربری منتظر می شوند.

این لایه های کاربری کاملا به سیستم و پلتفرم وابسته هستند و معمولا از پیش تعریف شده (استاتیک) توصیف می شوند و هرگونه تغییر در این لایه ها به صورت مخفی کردن بخش های خاص یا حتی کپی چند باره از بخش مد نظر صورت می پذیرد.

۲ تکنولوژی های پیشین

جهت پرداختن به مزایا و معایب لایه های کاربری ساختنی و همچنین برسی تفاوت های آن با شیوه های پیاده سازی کنونی ابتدا باید آشنایی کلی با وضعیت فعلی داشته باشیم. این تکنولوژی ها معمولا در سطح سکو یا سیستم عامل نهادینه شده و هرگونه تغییر در آنها نیازمند بروز رسانی سکو اجرایی است. در ادامه به معرفی چند مورد از شیوه های پر استفاده که با آنها کار کرده ام جهت بررسی میپردازم:

۲-۱ HTML در صفحات وب

صفحات وب مجموعه ای از المنت هاست [۲] که هرکدام مشخص کننده قسمتی از ویژگی های صفحه درحال نمایش هستند. صفحات وب از چندین تگ کلی:

۱. <HTML>: این تگ در برگیرنده کل اطلاعات صفحه است
۲. <BODY>: این تگ در برگیرنده کل محتوای قابل نمایش صفحه است
۳. <HEAD>: در این بخش اطلاعات صفحه مانند عنوان و استایل صفحه معرفی می شوند

از خواص این نوع تعریف لایه کاربری: تعریف قسمت های مختلف صفحه (دکمه ها، فرم ها، زیرصفحه و ...) در کنار مشخصات و ویژگی های آنها (رنگ، فونت، ...) و حتی کد های رویداد های مختلف در کنار هم است.

مزایا: [۴]

۱. این مدل توصیفی است، یعنی برنامه نویسی بدون نیاز به کد نویسی به توصیف لایه کاربری می پردازد.
۲. زبان ساده و خوانا، قابل استفاده برای نا-برنامه نویسان (طراحان، دانش آموزان و ...) [۱]

```html
<html id="rtl">
  <head>
    <title>HTML layout using Table (deprecated)</title>
    <link rel="stylesheet" href="styles.css">
    <style>
    </style>
  </head>
  <body>
    <div id="page-wrap">
      <div id="header" colspan="2">Header</div>
      <div id="main">
        <div id="sidebar" rowspan="2">Sidebar</div>
        <div id="content">
          <div id="sub-nav">sub menu</div>
          <div id="article">
            <form id="login" method="post">
            Username: <input type="text" name="user" value="" />
            Password: <input type="password" name="pass" value="" />
            <input type="submit" name="login" value="Login"/>
            </form>
          </div>
        </div>
        <div style="clear: both;"></div>
      </div>
      <div id="footer" colspan="2">Footer</div>
    </div>
  </body>
</html>
```

شکل ۱: نمونه کد HTML

۳. چشم پوشی از خطا: اگر در توصیف لایه کاربری خطایی (در بارگزاری یا حتی سمت برنامه نویس) رخ دهد کل برنامه بی استفاده نمی شود.

معایب:[۳]

۱. فقط امکان تولید صفحات استاتیک به تنهایی دارد

۲. مخلوطی از تمامی منطق برنامه، ظاهر کاربری، کنترل و اطلاعات. باعث عدم امکان توسعه خطی در پروژه های بزرگ و سازمانی می شود.

۳. نیازمند دیگر ابزار جهت پویا سازی مثل جاواسکریپت، activeX و ...

۴. صاحب و مدیر StateMachine است و برنامه نویس فقط با event ها از تغییرات وضعیت با خبر می شود. و کنترل کاملی روی آن ندارد. این مسئله باعث به وقوع پیوستن States nonDeterministic می شود.

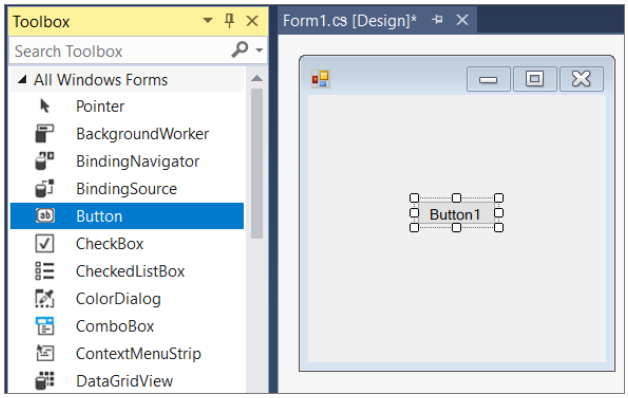
۲-۲ WindowsForms, UWP, WPF

پلتفرم WindowsForms تکنولوژی قدیمی بر بستر WindowsAPI ارایه می شود و به طور کلی ابزاری جهت پیاده سازی رابط کاربری گرافیکی برای برنامه های ویندوزی (GUI) است. در این پلتفرم جهت توصیف لایه کاربری مستقیما با LayoutEditor نسبت به تعریف رابط کاربری اقدام می کنیم. ایده پشت این پلتفرم Form-Based بودن است. به این شیوه که تمامی صفحاتی که کاربر با آن تعامل می کند در حقیقت فرمی جهت ورود اطلاعات و دریافت بازخورد از اوست.

نسخه بروزتر از این پلتفرم UWP و WPF که به ترتیب برای توسعه برنامه های چند سکویی و دسکتاپ طراحی شده اند دارای زبانی جهت توصیف لایه کاربری به نام XAML هستند. این شیوه شباهت بسیاری به صفحات HTML داشته و بیشتر مزایا و معایب آن در این شیوه هم صادق است. [۵]

۳-۲ Android

لایه های کاربری در اندروید، به صورت از پیش تعریف شده در فایل XML هایی به عنوان layout برنامه ایجاد می شوند. برنامه نویس امکان ویرایش این فایل را چه به صورت مستقیم و متنی و یا با کمک LayoutEditor اقام به تولید این لایه های کاربری می کند. در حین اجرای هر Activity (که نمایانگر قسمتی از منطق برنامه) است، این لایه کاربری بارگزاری می شود و نمایش داده می شود. برای ایجاد



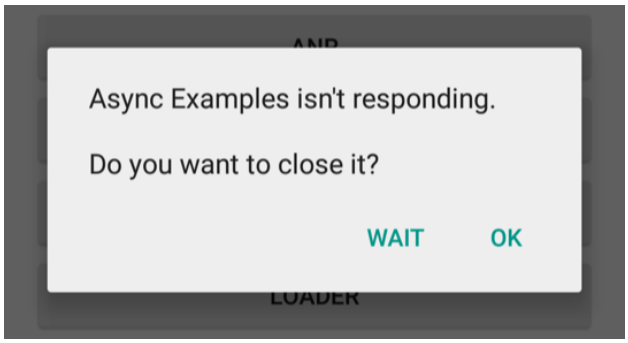
شکل ۲: نمونه طراحی با ویرایشگر WinForm

هرگونه تغییرات (از جمله افزودن مدیریت رویدادی مثل کلیک) باید ابتدا آن عنصر لایه کاربری را پیدا کرده، رفرنسی از آن را نگه داری و اقدام به اعمال تغییرات مد نظر در آن رفرنس کنیم.[۶]

```
// Imperative style
b.setColor(red)
b.clearChildren()
ViewC c3 = new ViewC(...)
b.add(c3)
```

شکل ۳: نمونه کد برنامه Android

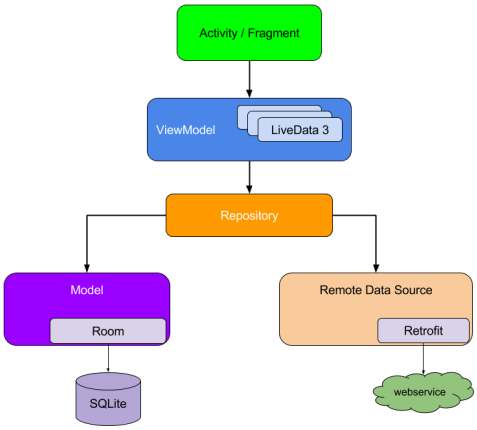
لازم به ذکر است که این نگه داری رفرنس در ادامه باعث بروز MemoryLeak ها می شود. این خطا یکی از رایج ترین خطاهای برنامه های اندرویدی است که نیاز به بررسی جداگانه یا توجه مضاعف برنامه نویس جهت مدیریت آنها دارد. [۷][۸]



شکل ۴: پیام خطایی آشنا، که MemoryLeak یکی از دلایل بروز آن است

سیستم لایه کاربری اندروید، یکی از معیوب ترین سیستم ها در پلتفرم های با بیشترین استفاده را دارد. یکی از این دلایل قدیمی و متفاوت بودن ورژن های مختلف اندروید روی میلیون ها دستگاه فروخته شده در دهه گذشته است. طراحان جهت سازگاری برنامه های تولید شده با ورژن های قدیمی و همچنین معرفی قابلیت های جدید مجبور به پیاده سازی Support Library های مختلف برای قسمت های مختلف محیط کار شدند. به طوری که استفاده از این کتابخانه های پشتیبانی

بدون افزودن قابلیتی جدید به برنامه ها در اندروید های بروز به حجم فایل نصبی (APK) برنامه چندین مگابایت می افزودند و حتی مصرف رم و حافظه دستگاه افزایش پیدا میکرد. از طرفی هاستفاده از همین کتابخانه های پشتیبانی در دستگاه های قدیمی تر باعث کاهش سرعت و کارایی و حتی مصرف باتری بیشتر یا داغ کردن دستگاه می شد. در ادامه با معرفی دسته کتابخانه JetPack گوگل تمرکز خود را برروی ارایه راهکار های خارج از پلتفرم به جای ارایه آنها درون هسته سیستم عامل اقدام کرد. [۹]



شکل ۵: شمایی از ساختار جدید ارایه شده گوگل برای برنامه ها

۳ زمینه

تا کنون با تعدادی از شیوه های تولید رابط کاربری قدیمی آشنا شدیم. در اکثر این شیوه ها برنامه نویس هیچ گونه مدیریتی بر وضعیت ماشین (برنامه) و محتوای نمایش داده شده به کاربر به طور مستقیم ندارد. بلکه با استفاده از ویجت های ارایه شده هر پلتفرم و با اتکا به رویداد های گزارش شده آنها به بروزرسانی وضعیت مد نظر می پردازد.

این شیوه از مدیریت وضعیت و اتکا به پلتفرم های گوناگون (با امکان آپدیت و تغییرات) باعث مشکلات بسیاری در برنامه ها می شود.

اکثر ما تا کنون با برنامه هایی که برخلاف انتظار برنامه نویس به وضعیتی نامطلوب رفته اند (فرضا یک دیالوگ "درحال بارگزاری" در یک برنامه اندرویدی که با چرخش صفحه ناپدید می شود اما از دید برنامه هنوز باید نمایش داده می شد). یکی از مهمترین دلایل این تجارب، نه از عدم آگاهی توسعه دهنده بلکه از **غیر قابل پیش بینی بودن پلتفرم** ایجاد می شود.

جهت مدیریت اینگونه وضعیت های از قبل پیشبینی نشده که اکثرا به علت نقص در پلتفرم، بروز نبودن پلتفرم در دستگاه کاربر (مثل نسخه اندرویدی دستگاه)، عدم شناخت کافی توسعه دهنده از ویژگی ها و عکس العمل های هر کامپوننت در شرایط گوناگون نشات می گیرد، مستلزم به بکارگیری تیم آزمایش محصول و همچنین فرایند پیچیده تر دریافت گزارش های خطا از کاربر می شود. حتی گاهی مدیریت یک وضعیت ناخواسته از خود منطق برنامه پیچیدگی بیشتری پیدا میکند.

یکی دیگر از موارد مهم، Reusability کد ها و کامپونت هایی که یک برنامه نویس تولید و استفاده می کند است. در شیوه سنتی به دلیل وابستگی کامل برنامه نویس به زمینه ای که در آن یک کامپونت نوشته می شود (از جمله موقعیت قرار گیری در صفحه، المنت های دیگر صفحه و ...) امکان استفاده از یک کامپونت در دیگر جاهای برنامه کمتر و فرایند آماده سازی یک کامپونت برای استفاده در پروژه های دیگر خیلی هزینه زیادی دارد.

اکثر مشکلات اشاره شده در این قسمت و قسمت های قبل از عدم کنترل برنامه نویس بر وضعیت کنونی برنامه و همچنین متمرکز نبودن مرجعیت درستی وضعیت در کل برنامه است. برای مثال: به این صورت که از دید برنامه نویس چک باکس شماره ۱ فعال و چک باکس ۲ غیر فعال است. با تغییر این انتخاب توسط کاربر در اکثر پلتفرم ها ابتدا وضعیت تغیر پیدا کرده و سپس تغییر جدید به برنامه نویس از طریق Event ها اطلاع داده می شود. و برنامه نویس کنترلی روی این تغییر ندارد.

۴ راه کار

راه حل این مسئله فرستادن کل فرایند رسیدگی و مدیریت وضعیت ها به برنامه نویس و رندرینگ و ... به پلتفرم است. همچنین کامپونت های تشکیل دهنده لایه کاربری هیچ گونه وضعیتی آشکار (قابل مدیریت توسط برنامه نویس) نگه داری نکنند. به این ترتیب حتی تغییر وضعیت یک دکمه رادیویی، تایپ در یک تکست باکس هم با اطلاع و موافقت برنامه نویس صورت میگیرد.

برای پیاده سازی این شیوه تولید کامپونت های لایه کاربری نیاز است که تغییری در نحوه نگاه به لایه های کاربری به طور کلی بیاندازیم. از جمله:

۱. هر کامپونت فقط باید به نقش انحصاری خود بپردازد و در کار بقیه قسمت ها تغییری ایجاد نکند.
۲. اطلاعات مورد نیاز جهت پیاده سازی به کامپونت به آن داده می شود. همچنین کامپونت از هرگونه تغییری در اطلاعات باید آگاه شود.
۳. کامپونت ممکن است جهت اجرای نقش مشخص شده خود، دارای وضعیتی داخلی باشد که البته هیچ تاثیری در وضعیت کلی برنامه نباید بگذارد.
۴. کامپونت ها نباید وابستگی به والد یا فرزندان خود داشته باشند و به تنهایی و در هر محیط و usecase بتوانند وظیفه مربوطه را انجام دهند.

یکی از دیدگاه های مفید در این زمینه به این صورت مطرح می شود که:

شکل ۶: UI is result of application function on state

به این معنی که هرگونه تغییر در وضعیت برنامه در ظاهر کاربری اثر میگذارد و نه برعکس. ظاهر کاربری هیچگونه مسیولیتی در زمینه صحت سنجی، بروز رسانی، مدیریت، بارگذاری و پردازش اطلاعات ورودی و خروجی و همچنین وضعیت فعلی برنامه ندارد و تنها مسئولیت آن به نمایش گذاشتن وضعیت فعلی اطلاعات برنامه است.

۵ پلتفرم های امروزی

در قسمت قبل به بررسی راه حل کلی این مدل پیاده سازی آشنا شدیم. حال در ادامه چندین پیاده سازی این راه کار را بررسی می کنیم:

۵-۱ React

توسعه دهنده: فیسبوک ، مجوز: .MIT این فریم ورک در اصل یک موتور نمایش اطلاعات در DOM است. به همین علت هرگونه مدیریت وضعیت و ... نیازمند کتابخانه های کمکی جهت مدیریت وضعیت برنامه از جمله Redux نیازمند است. کامپوننت های تولید شده توسط این فریم ورک به صورت تگ های HTML درون برنامه استفاده می شوند. همچنین تمامی وضغیت مورد نیاز کامپوننت در ابتدای ایجاد آن کامپوننت به آن ارسال می شود.

```
class App extends React.Component {
  render() {
    return (
      <div>
        <p>Header</p>
        <p>Content</p>
        <p>Footer</p>
      </div>
    );
  }
}
```

شکل ۷: A simple application component in ReactJS

از مزایای سیستم ComponentBased فیسبوک قابلیت جابجایی آن است. اکثر کامپونت های طراحی شده در پروژه شما بدون نیاز به تغییری قابلیت ایمپورت و استفاده درجا دارند. این قابلیت از آنجا که تمامی دیتای مورد نیاز و وابسته به کامپوننت حین استفاده از آن تخصیص داده میشود و هیچ گونه وابستگی به دیتا استاتیک یا گلوبال وجود ندارد امکان پذیر شده است. [۱۰]

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(state => ({
      seconds: state.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }
```

شکل ۸: an Stateful component in React

۵-۲ Flutter

توسعه دهنده: گوگل ، مجوز: BSD ۳. این فرم ورک جهت تولید لایه های کاربی چند سکویی با هدف کنترل تمام پیکسل های صفحه با ایده گیری از React تولید شد. زبان این فریم ورک Dart است که شباهت بسیار زیادی به جاوا دارد.

در فلاتر مثل React ما به توصیف لایه های کاربری می پردازیم. همچنین کد منطق نرم افزار می تواند (توصیه نمی شود) در کد لایه کاربری ترکیب شود. از مزیت این فریم ورک سرعت فوق العاده بالای آن، چند سکویی (اندروید، آی او اس، وب، ویندوز، مک، ...) بوده و امکان کنترل در سطح پیکسل به پیکسل برنامه را به برنامه نویس می دهد. کامپوننت های نوشته شده در فلاتر به نام ویجدت شناخته می شوند و در دو حالت نوشته می شوند: حالت Stateless به معنی اینکه ویجدت خود حاوی هیچ گونه وضعیتی نیست و حالت Stateful به معنای اینکه ویجت برای عملکرد صحیح نیاز به نگداری تعدادی وضعیت داخلی دارد.

HelloWorld.dart

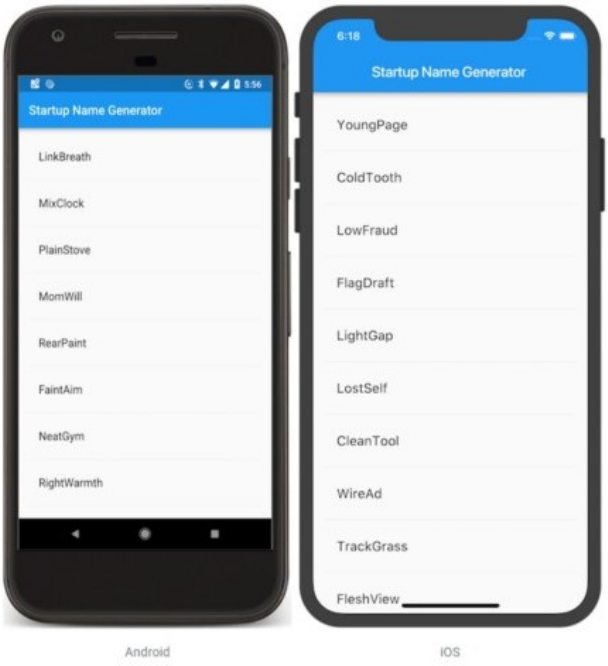
```
import 'package:flutter/material.dart';

void main() => runApp(new MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Welcome to Flutter',
      home: new Scaffold(
        appBar: new AppBar(
          title: new Text('Welcome to Flutter'),
        ),
        body: new Center(
          child: new Text('Hello World'),
        ),
      ),
    );
  }
}
```

شکل ۹: Simple hello world application in flutter

مطابق شکل ۹ قابل مشاهده است که حتی اپلیکشن تعریف شده در فلاتر به صورت یک ویجت تعریف و مصرف می شود. این یکسان سازی کاربری در کل پلتفرم باعث سادگی و همگرایی کلی برنامه نویس و طراح می شود. لازم به ذکر است فلاتر یکی از محبوب ترین فریم ورک های برای نابرنامه-نویسان و طراحان جهت آشنایی با برنامه نویسی، طراحی و ارایه پروتوتایپ های سریع از برنامه است.

از مزایای دیگر این فریم ورک چند سکویی بودن برنامه و متحد الشکل بودن نتیجه بین همه سکو های اجرای برنامه است. این قابلیت در خطوط تولید برنامه ها در شرکت های استارت آپی با تیم توسعه کوچک و همچنین شرکت ها با برنامه های بزرگ بسیار کاربردی است. به این علت که مدیریت و نگهداری یک Code Base و ارایه قابلیت های جدید و رفع باگ های گزارش شده همزمان صورت می گیرد. شرکت های مختلف پس از مهاجرت به فلاتر اکثرا تجربه ای واحد در بهبود

تیم توسعه و افزایش سرعت و بهبود تجربه کاربری داشته اند. [۱۱]



شکل ۱۰: a Flutter App run in both android and iOS

۵-۳ تازه وارد ها

پس از معرفی و موفقیت Flutter و React کتابخانه های گوناگونی با همین ایده شکل گرفته اند. از جمله UI Swift مختص تولید لایه های کاربری نوین برای iPadOS ،iOS ،macOS و tvOS یا Compose Android ویژه تولید لایه های کاربری جدید اندروید.

این کتابخانه های جدید اکثرا در مرحله پیش نمایش یا بتا هستند و به طور قطعی نمی توان از کارایی و عملکرد آنها اطلاع پیدا کرد، اما می توان این نتیجه را گرفت که: **بازار آینده تولید روابط کاربری به سمت این ایده حرکت می کند.**

۶ بهبود تجربه

استفاده از مدل برنامه نویسی جدید علاوه بر مزایا، به دلیل جدید بودن شیوه کار با این تکنولوژی ها بعضی سختی هایی هم دارد. در اکثر این تکنولوژی ها به توصیف لایه کاربری به همان شیوه ای که در صفحات HTML به صورت درختی تعریف میشدند، می پردازیم. این شیوه در لایه های کاربری غنی و پر جزئیات بعضی باعث تورفتگی های با درجه زیاد می شود. راه کار هایی برای این گونه مشکلات ارایه شده از جمله تعریف تابع به تابع قسمت های مختلف و لینک به یکدیگر یا در زبان هایی مثل Swift، Kotlin و Dart جدیدا با تعریف function extension های کاربری این تو رفتگی ها را از دید کاربر محو می کنند.

۷ نتیجه گیری

با فراگیری استفاده از تکنولوژی های جدید فوق، تیم های توسعه دهنده برنامه ها به شکل چشم گیری تغییر شکل پیدا خواهند کرد. روش ها و نگاه ها به فرایند توسعه کاملا عوض شده و در ادامه با افزایش جمعیت جامعه و ابزار های متن باز و

```
Padding(pad,
    child: ChildView(...),
)


ChildView(...)
  .padding(pad)
```

شکل ۱۱: convertion of hierachy to extension function

... شاهد تحولی در برنامه های موجود در بازار خواهیم بود. شایسته است که برنامه نویسان فعال در این حوزه، پیشاپیش نسبت به فراگیری ، تست و کسب تجربه با این تکنولوژی ها آشنا شوند و در آینده بتوانند در رقابت با بازار جهانی سطح بالایی از محصولات نرم افزاری به بازار ارایه دهند. همچنین گسترش ابزار ها، کامپونت های فارسی، ترجمه یا همگام سازی موارد فعلی نیز در گسترش این تکنولوژی ها در برنامه های داخلی کمک شایانی می کند.

مراجع

[1] Official *HTML 4.0 Specification*. W3C Recommendation, revised on 24-Apr-1998.

[2] Wikipedia: Hypertext Markup Language.

[3] Cons of HTML

[4] Meduim Article, Pros of HTML, olivia sand, Jan 2018.

[5] Choose your app platform, Article, Microsoft, 2019.

[6] Android Layouts, Android Developers Document, Google.

[7] Article, 9 ways to avoid memory leaks in Android, Anitaa Murthy, May 2018.

[8] ProAndroidDev Article, Everything you need to know about Memory Leaks in Android, Ali Asadi, Jun 2019.

[9] Android Jetpack, Android Developers Document, Google.

[10] A JavaScript library for building user interfaces, ReactJs, Facebook.

[11] Flutter Application Showcase, Flutter.