دانشگاه قم
دانشکده فنی و مهندسی
پایان‌نامه کارشناسی رشته مهندسی کامپیوتر- نرم‌افزار

عنوان

طراحی و پیاده‌سازی سامانه‌ی پیام‌رسان تحت اندروید

استاد راهنما
دکتر حسین امیرخانی

نگارنده
مهدی جواهری‌صابر

پاییز ۱۳۹۶


بسم الله الرحمن الرحیم

تقدیم به:

استاد بزرگوار و فرهیخته، دکتر حسین امیرخانی که با دلسوزی بی‌نظیر و راهنمایی‌های گرانقدر ایشان این اثر گردآوری گردید.

چکیده:

ابزارهای ارتباط جمعی روز به روز دستخوش تغییر می‌گردند. در جامعه‌ی امروزی شبکه‌های اجتماعی نقش پررنگی در ارتباطات میان فردی بازی می‌کنند. لذا دغدغه‌ی راهکاری ایمن و یکپارچه برای این ارتباطات منجر به خلق چنین اثری گردید. این پروژه با هدف رویارویی با چالش‌های مهندسی نرم‌افزار در طراحی، پیاده‌سازی و به‌کارگیری اپلیکیشن پیام‌رسان در بستر سیستم‌عامل اندروید به‌عنوان کلاینت و سرور با زبان جاوا انجام گردید. از آنجایی که مسئله‌ی پیام‌رسان ذاتا با پردازش لحظه‌ای همراه‌است، مباحثی چون شبکه، پایگاه‌داده، چندنخی[۱] IPC[۲]، الگوی طراحی[۳] SOC[۴] و لایه‌بندی[۵] از الزامات این پروژه هستند. مسائلی که توسعه‌ی اپلیکیشن را با چالش‌های زیادی مواجه کرد معماری، الگوی طراحی، رفع خطاهای شبکه و همگام‌سازی اطلاعات بود. در این مقاله سعی بر آن شده تا راهکارهای عملی و مطابق با استانداردهای ارائه‌شده توسط مراجع معتبر، در قالب کدهای نمونه و توضیحات دقیق، گردآوری گردد.

واژه‌های کلیدی:

Android، Java، MVP، SQL، Socket، Thread

آدرس پست الکترونیک نویسنده: m.javaherisaber@gmail.com

---

[۱] Threading

[۲] Inter-Process-Communication

[۳] Design Pattern

[۴] Separation of concerns

[۵] Layering

فهرست مطالب

عنوان **صفحه**

فهرست جداول

عنوان **صفحه**

فهرست شکل‌ها

عنوان **صفحه**

فصل ۱:

مفاهیم برنامه نویسی اندروید

۱-۱- مفاهیم پایه

اپلیکیشن‌های اندروید به‌زبان جاوا نوشته شده‌اند. ابزار توسعه‌ی اندروید[۱] کدهای شما به‌همراه منابع و فایلهای پروژه را دریافت کرده و یک فایل با پسوند apk ایجاد می‌کند که فایل نصبی برنامه می‌باشد.

هر برنامه‌ی اندرویدی داخل یک محیط امنیتی محافظت‌شده به‌نام sandbox[۲] اجرا می‌گردد [1] که این محیط ویژگی‌های زیر را دارد:

۱. سیستم‌عامل اندروید یک سیستم لینوکسی چند کاربره است، به این معنا که هر برنامه به‌عنوان یک کاربر تلقی می‌شود.

۲. سیستم‌عامل به‌طور پیشفرض برا هر برنامه یک شناسه‌ی کاربری انتساب می‌دهد (که فقط توسط سیستم قابل خواندن است و از دید برنامه‌ها مخفی است). سیستم‌عامل دسترسی‌های لازم را به تمامی فایلهای مربوط به یک برنامه را انتساب کرده و فقط به همان برنامه (با شناسه کاربری یکتای خود) اجازه‌ی دسترسی می‌دهد.

۳. هر فرآیند[۳] در اندروید، ماشین مجازی[۴] خودش را دارد، بنابراین کدهای یک برنامه در محیطی ایزوله اجرا می‌گردد که از دید بقیه‌ی برنامه‌ها مخفی است.

۴. به‌طور پیشفرض هر برنامه‌ای داخل فرآیند لینوکسی خودش اجرا می‌شود. هنگامی که یکی از اجزای برنامه اجرا شود سیستم‌عامل این فرآیند را راه‌اندازی کرده و زمانی که کار برنامه تمام شود یا نیاز به حافظه باشد، آن را نابود می‌کند.

---

[۱] Android SDK Tools

[۲] در این محیط داده‌های اپلکیشن به صورت خصوصی ذخیره می‌گردد.

[۳] Process

[۴] Virtual Machine

سیستم‌عامل اندروید اصل «حداقل امتیاز» را پیاده سازی کرده، بدین معنا که هر برنامه به‌طور پیشفرض تنها به اجزایی دسترسی دارد که برای کارکرد خود نیاز دارد. این روش محیطی فوق العاده امن را ایجاد می‌کند که برنامه‌ها نمی‌توانند به قسمت‌هایی از سیستم‌عامل که دسترسی داده نشده، راه یابند. به هر حال سیستم‌عامل راهکارهایی را برای اشتراک گذاری داده با بقیه برنامه‌ها و دسترسی به سرویسهای سیستمی در اختیار برنامه‌ها قرار داده که به‌عنوان نمونه:

۱. این امکان وجود دارد که دو برنامه شناسه‌ی کاربری لینوکسی یکسانی را به اشتراک بگذارند، تا بتوانند به فایلهای یکدیگر دسترسی داشته باشند. برای حفظ منابع سیستمی، برنامه‌های با شناسه‌ی کاربری یکسان می‌توانند حتی در یک فرآیند لینوکسی یکسان اجرا شده و ماشین مجازی یکسانی داشته باشند. این حالت تنها زمانی می‌تواند رخ دهد که هر دو برنامه با یک فایل گواهی امضا شده‌باشند.

۲. برنامه‌ها می‌توانند برای دسترسی به داده‌های دستگاه از قبیل مخاطبین، پیام‌ها، حافظه‌های جانبی، دوربین و بلوتوث درخواست مجوز کنند. در این صورت کاربر به‌صورت آشکار یا ضمن نصب برنامه این دسترسی‌ها را به برنامه می‌دهد.

در ادامه مفاهیم زیر را بررسی می‌کنیم:

۱. اجزای[۲] اصلی اندروید که هسته‌ی عملکرد برنامه‌های شما را تشکیل می‌دهند.

۲. فایل manifest که شناسنامه‌ی برنامه‌ی شماست و اجزا و قابلیت‌های برنامه را در آن تعریف می‌کنیم.

۳. منابع برنامه که از سورس کد برنامه جدا هستند و به برنامه اجازه‌ی بهبود عملکرد بر اساس پیکر بندی‌های متنوع دستگاه‌ها می‌دهند.

[۱] Least Privilege

[۲] Component

۲-۱- اجزای اصلی

اجزای اصلی اندروید مهره‌های کلیدی را در یک اپلیکیشن اندرویدی تشکیل می‌دهند. هر جزء نقطه شروع تعامل کاربر یا سیستم با برنامه‌ی شماست. برخی از اجزا به دیگری وابستگی دارند.

در سیستم‌عامل اندروید چهار جزء اصلی وجود دارد:

۱. Activity
۲. Service
۳. Broadcast receiver
۴. Content provider

هر نوع برای مقاصد خاصی بوده و طول عمر مختص خودش را دارد که مشخصه‌ی آغاز و پایان هر جزء در حین فراخوانی می‌باشد. در زیر مختصری از این اجزا را توضیح می‌دهیم. (جزئیات بیشتر به‌تفصیل در بخش‌های بعدی مطرح شده)

۲-۱-۱- Activity

Activity نقطه‌ی شروع برای تعامل با کاربر است. این جزء مسئول نمایش یک صفحه‌ی گرافیکی است. به‌عنوان مثال برنامه‌ی ایمیل ممکن است صفحه‌ای برای نمایش لیست ایمیل‌های جدید و صفحه‌ای دیگر برای نوشتن ایمیل داشته باشد. اگرچه activityها با یکدیگر کار می‌کنند تا یک تجربه‌ی یکپارچه برای کاربر مهیا کنند، با این حال آنها مستقل از یکدیگرند. ویژگی استقلال صفحات از یکدیگر این امکان را به برنامه‌ها می‌دهد که بتوانند صفحات یکدیگر را فراخوانی کنند و نتیجه تعامل کاربر با آن صفحه را به اشتراک یکدیگر بگذارند. به‌عنوان نمونه یک برنامه عکاسی می‌تواند یک activity از برنامه‌ی ایمیل را فراخوانی کرده و تصویر گرفته‌شده را از طریق ایمیل به اشتراک دیگران قرار دهد.

activity تعامل‌های کلیدی زیر را بین سیستم و اپلیکیشن فراهم می‌کند:

۱. پیگیری فعالیت فعلی کاربر با دستگاه (که با یک صفحه یا activity است) برای تضمین اینکه سیستم بتواند فرآیندی که آن activity را اجرا می‌کند در حال اجرا باقی بماند.

۲. دانستن فرآیندهای پیشین که کاربر قبلا با آنها در تعامل بوده (صفحات متوقف‌شده) و اولویت بندی منابع سیستم برای نگه داری این فرآیندها، چرا که کاربر با احتمال بالاتری دوباره به این صفحات رجوع می‌کند.

۳. مهیاکردن امکان پیاده سازی جریان تغییر صفحه بین اپلیکیشن‌ها و هماهنگی سیستم‌عامل با این تغییر صفحات.

هر صفحه یا activity در اندروید یک کلاس جاوا است که از کلاس Activity ارث بری می‌کند.

Service -۱-۲-۲

Service یا خدمتگزار یک نقطه شروع از اپلیکیشن برای نگه داری حالت اجرای برنامه در پشت زمینه، به دلایل مختلف می‌باشد. این جزء که در پشت زمینه اجرا می‌شود به‌منظور انجام عملیات‌های طولانی و یا کار با فرآیندهای راه دور[۱] به‌کار می‌رود. سرویس رابط کاربری ندارد. به‌عنوان مثال، ممکن است یک سرویس حین تعامل کاربر با برنامه‌ی دیگر در پشت زمینه موزیک پخش کند و یا داده‌هایی را بدون مختل‌کردن تعامل کاربر با activity، از طریق شبکه ارسال کند. اجزای دیگر مثل activity می‌تواند سرویس راه‌اندازی کند و به آن سرویس دستور اجرا داده یا حتی می‌تواند با متصل شدن به آن تعاملاتی را انجام دهد.

دو الگوی کاملا مجزا برای سرویس‌ها وجود دارد که به سیستم چگونگی مدیریت اپلیکیشن را اعلام کنند:

۱. سرویس‌های فراخوانی‌شده (Started services): به سیستم می‌گوید که تا زمان اتمام کار مورد نظر، سرویس را در حال اجرا نگه دارد. این حالت می‌تواند به‌منظور همگام سازی داده در پشت زمینه یا پخش موزیک باشد. حتی زمانی که کاربر اپلیکیشن را ترک می‌کند. همگام سازی و پخش موزیک نیز دو نوع متفاوتی از سرویس‌های فراخوانی‌شده می‌باشند که چگونگی کنترل آنها توسط سیستم متفاوت است:

- پخش موزیک عملی است که کاربر به‌طور مستقیم از آن اطلاع دارد. بنابراین اپلیکیشن با نمایش اعلان مناسب در نوار بالای صفحه[۲] کاربر را از جریان پخش موزیک مطلع نگه می‌دارد. در این

[۱] Remote

[۲] Notification Bar

حالت سیستم می‌داند که باید به سختی تلاش کند فرآیند مربوط به این سرویس را در حال اجرا نگه دارد. چرا که اگر سرویس متوقف شود نارضایتی کاربر را در پی دارد.

o سرویس همگام سازی دوره‌ای عملی است که کاربر به‌طور مستقیم از آن اطلاع ندارد. بنابراین سیستم آزادی عمل بیشتری در مدیریت فرآیند مربوط به آن دارد. یعنی ممکن است در صورت نیاز اجزاء دیگر به منابع، سیستم این سرویس را نابود کرده و در زمان مناسب دیگری اقدام به راه‌اندازی مجدد کند.

۲. سرویس‌های مقید (Bound services) به این دلیل اجرا می‌شود که جزء دیگری از برنامه یا حتی خود سیستم فرمان اجرا را صادر کرده و تصمیم دارد که نتیجه‌ای را از این سرویس دریافت کند. اساسا این سرویس یک API برای فرآیند دیگر فراهم می‌کند. در نتیجه سیستم می‌داند که وابستگی بین این فرآیندها وجود دارد، بنابراین اگر فرآیند A به سرویس حاضر در فرآیند B متصل شده‌باشد، می‌داند که باید فرآیند B (و سرویس آن) را بایستی برای استفاده در فرآیند A در حال اجرا نگه دارد.

به خاطر انعطاف پذیری بالا، سرویس‌ها به یکی از مهره‌های کلیدی برای هر نوع سیستم سطح بالا به‌کار گرفته می‌شود. تصویر زمینه‌های زنده[۱] محافظ صفحه نمایش[۲] متدهای ورودی و بسیاری از ویژگی‌های هسته‌ی سیستم‌عامل همگی بر پایه‌ی سرویس ساخته شده‌اند.

سرویس در اندروید یک کلاس جاوا است که از کلاس Service ارث بری می‌کند

Broadcast Receiver -۱-۲-۳

broadcast receiver بخشی است که سیستم را قادر به پخش رویدادهایی به اپلیکیشن میکند که خارج از فعالیت‌های کاربر اتفاق می‌افتد. از آنجایی که broadcast receiverها نقطه آغازین[۳] دیگری از یک برنامه به شمار می‌روند، سیستم می‌تواند رویدادهای broadcast را حتی به اپلیکیشن‌هایی که در

[۱] Live Wallpaper

[۲] Screen Saver

[۳] Entry Point

حال اجرا نیستند، انتقال دهد. بنابراین به‌عنوان مثال، یک برنامه می‌تواند ثانیه شماری برای ارسال اعلان به نوار اعلانها تنظیم کند که کاربر را درباره‌ی رویداد پیش رو مطلع گرداند، با ارسال این ثانیه شمار به قسمت broadcast receiver برنامه، تا زمانی که ثانیه شمار به موعد میرسد، دیگر نیازی به این نیست که برنامه مذکور در حالت اجرا باقی بماند. بسیاری از پیغام‌های broadcast توسط سیستم تولید می‌شوند، مثلا از خاموش شدن صفحه، ضعیف بودن باتری، روشن شدن وایفای، وصل شدن جک هنذفری و... اطلاع میدهند. برنامه‌ها نیز می‌توانند آغازگر پیام broadcast باشند، مثلا به برنامه‌های دیگر اطلاع دهند که یک سری داده دانلود شده و در دسترس آنها قرار دارد.

اگرچه broadcast receiverها رابط کاربری ندارند، ممکن است نوار اعلان ثابتی را در بالای صفحه ایجاد کنند و حین رخ دادن یک رویداد کاربر را از آن مطلع کنند.

broadcast receiver در اندروید یک کلاس جاوا است که از کلاس BroadcastReceiver ارث بری می‌کند و هر پیام broadcast داخل یک شی Intent به سیستم پاس داده می‌شود.

Content Provider -۱-۲-۴

وظیفه‌ی این جزء، اشتراک گذاری مجموعه داده‌هایی از برنامه که می‌توانند در قالب فایل، پایگاه داده SQLite، روی وب و یا هر مکان ذخیره سازی دیگری که برنامه‌ی شما می‌تواند دسترسی داشته باشد. از طریق content provider دیگر برنامه‌ها می‌توانند از داده‌ها query گرفته یا آنها را تغییر دهند، در صورتی که content provider اجازه‌ی دسترسی را بدهد. به‌عنوان نمونه سیستم اندروید یک content provider برای مدیریت اطلاعات مخاطبین ارائه داده‌است. به اینصورت که هر برنامه‌ای با دسترسی‌های مناسب می‌تواند از content provider پرس و جو کند، مثلا از طریق ContactsContract.Data[1] می‌تواند درباره‌ی یک شخص خاصی، اطلاعاتی را خوانده و یا بنویسد.

اغواء کننده است که به content provider از دید یک انتزاع روی پایگاه داده نگاه کنیم، چرا که واسط‌ها و پشتیبانی پیشفرض زیادی برای آنها در موارد عمومی وجود دارد. به هر حال آنها هسته هدف متفاوتی از دید طراحی سیستمی دارند.

[1] نام جدولی در پایگاه‌داده‌ی مخاطبین دستگاه

برای سیستم، یک content provider مدخلی به برنامه است که به‌منظور انتشار داده‌های اسمی، شناسایی‌شده با یک URI scheme می‌باشند. بنابراین یک برنامه می‌تواند تصمیم بگیرد که چگونه میخواهد داده‌هایی را که دارد به یک فضای نام URI[1] انتساب دهد و آنها را طوری روی موجودیت‌های دیگر هدایت کند که این موجودیت‌ها در عوض بتوانند به داده‌ها دسترسی پیدا کنند.

یکسری موارد وجود دارند که به سیستم امکان انجام دادن چنین مدیریتی روی یک برنامه را میدهند:

۱. انتساب یک URI نیازی به درحال اجرا بودن برنامه ندارد، بنابراین URIها می‌توانند حتی بعد از اینکه برنامه‌ی صاحب آنها از حالت اجرا خارج می‌شوند، باقی بمانند. سیستم تنها زمانی نیاز دارد بداند که برنامه‌ی صاحب URI در حال اجرا است که مجبور باشد داده‌ها را با استفاده از URI مربوطه از برنامه استخراج کند.

۲. همچنین این URIها یک مدل امنیتی دقیق را ارائه میدهند. به‌عنوان مثال یک برنامه می‌تواند URI مربوط به دسترسی به یک تصویر را داخل clipboard[2] نگه دارد ولی content provider مربوط به آن را قفل کند، بنابراین دیگر برنامه‌ها نمی‌توانند آزادانه به آن تصویر دسترسی یابند. زمانی که برنامه‌ی دومی تلاش می‌کند که به یک URI که روی clipboard قرار دارد، دسترسی پیدا کند، سیستم می‌تواند به آن برنامه از طریق یک URI دسترسی موقت، اجازه‌ی دسترسی دهد، بنابراین دسترسی به آن داده تنها پشت آن URI اجازه داده‌شده، نه چیز دیگری که مستقیما از داخل برنامه‌ی دوم باشد.

content provider در اندروید یک کلاس جاوا است که از کلاس ContentProvider ارث بری می‌کند و بایستی یکسری رابط‌های استاندارد را پیاده سازی نماید که برنامه‌های دیگر را قادر به انجام تراکنش‌ها می‌کند.

یک جنبه‌ی منحصر بفرد در طراحی سیستم‌عامل اندروید این است که در آن هر برنامه‌ای می‌تواند

[1] Uniform Resource Identifier

[2] حافظه‌ای در دستگاه که متن کپی‌شده را نگه‌داری می‌کند

اجزای برنامه‌ی دیگری را فراخوانی نماید. به‌عنوان مثال اگر شما میخواهید، کاربر تصویری را با دوربین بگیرد، احتمالا برنامه‌ی دیگری وجود دارد که این کار را برای شما انجام دهد بدون آن که نیاز باشد خودتان یک activity برا گرفتن آن تصویر بنویسید. نیازی نیست که کدی را از برنامه‌ی دوربین داشته باشید و یا به کدهای آن لینک شوید. در عوض میتوانید به سادگی activity برنامه‌ی دوربین را راه‌اندازی نمایید. زمانی که گرفتن تصویر کامل شد، تصویر گرفته‌شده به برنامه‌ی شما برگردانده می‌شود و میتوانید از آن استفاده کنید. از دید کاربر اینطور به نظر میرسد که برنامه‌ی دوربین واقعا جزئی از برنامه‌ی شماست.

زمانی که سیستم یک جزء (component) را راه‌اندازی می‌کند، در واقع یک فرآیند برای آن برنامه را راه‌اندازی می‌کند اگر قبلا در حال اجرا نباشد و تمامی کلاسهای لازم برای آن جزء را بارگذاری می‌کند. مثلا اگر برنامه شما activity برنامه‌ی دوربین را که وظیفه‌ی گرفتن تصویر را دارد، راه‌اندازی می‌کند، آن activity داخل فرآیندی که متعلق به برنامه‌ی دوربین است اجرا می‌شود، نه فرآیند برنامه‌ی شما ! بنابراین برخلاف برنامه‌های اکثر سیستم‌عامل‌های دیگر، برنامه‌های اندروید مدخل شروع واحدی ندارد یعنی تابع ()main وجود ندارد.

از آنجایی که سیستم‌عامل هر برنامه را در فرآیندی جداگانه با دسترسی‌های فایل مربوطه که برای بقیه برنامه‌ها محدود است اجرا می‌کند، برنامه‌ی شما نمی‌تواند به مستقیما جزئی از برنامه‌ی دیگر را راه‌اندازی کند. به هر حال سیستم‌عامل می‌تواند.

لذا برای فعال سازی جزئی از برنامه‌ی دیگر، بایستی پیامی در قالب Intent به سیستم‌عامل ارسال گردد. سپس سیستم‌عامل آن جزء را برای شما فعال می‌کند.

۳-۱- راه‌اندازی اجزا

سه مورد از چهار جزء اصلی اندروید (activity، service و broadcast receiver) توسط یک پیام غیر همزمان به‌نام intent فعال سازی می‌شوند. Intentها اجزا را در زمان اجرا به یکدیگر متصل می‌کنند. میتوانید آنها را به دید یک پیام رسان نگاه کنید که عملی را از اجزای دیگر درخواست میدهد، حال آن جزء چه متعلق به برنامه شما باشد چه از برنامه‌ی دیگر.

intent توسط یک شی از نوع Intent ساخته می‌شود که پیامی را برای فعالسازی یک جزء معین (explicit intent) و یا نوع معینی از اجزاء (implicit intent) تعریف می‌کند. (جزئیات بیشتر را در بخش Intent مطالعه نمایید)

برای activity و serviceها یک پیام intent عملی برای اجرا (مثلا نمایش یا ارسال چیزی) تعیین می‌کند و ممکن‌است یک URI از داده‌ای که آن عمل روی آن انجام می‌شود، در بین بقیه چیزهایی که آن جزء در حین فعال شدن نیاز به دانستن داشته باشد معین کند. به‌عنوان نمونه یک intent ممکن‌است حامل درخواستی به یک activity برای نمایش یک تصویر و یا بازکردن صفحه‌ای از وب باشد. در برخی موارد میتوانید یک activity را برای دریافت یک نتیجه آغاز کنید که در اینصورت این activity نیز در حین برگشت پیام intent با خود برمیگرداند. به‌عنوان مثال میتوانید یک intent تدارک ببینید که کاربر مخاطبی را انتخاب کند و به آن مخاطب دسترسی داشته باشید. Intent برگشتی حاوی یک URI می‌باشد که به مخاطب انتخابی کاربر اشاره می‌کند.

برای broadcast receiverها، intent به سادگی پیامی که نیاز به انتشار همگانی است را تعیین می‌کند. مثلا یک broadcast برای اعلان پایین بودن سطح باتری، تنها یک رشته متنی مشخص که نشانگر «باتری ضعیف است» باشد را شامل می‌شود. برخلاف activity، service و broadcast receiver، content provider توسط intent فعالسازی نمی‌شود. بلکه آنها زمانی فعال می‌شوند که توسط یک درخواست از ContentResolver مورد نظر باشند. Content resolver تمامی تراکنشهای مستقیم با content provider را کنترل می‌کند، بنابراین جزئی که تراکنشی را روی provider اعمال می‌کند نیازی به درگیری مستقیم نیست بلکه در عوض متدهایی از شی ContentResolver را فراخوانی می‌کند.

متدهای جداگانه‌ای برای فعالسازی هر نوع از اجزاء وجود دارد:

۱. با استفاده از پاس دادن یک Intent به متدهای startActivity() یا startActivityForResult() (اگر از activity نتیجه‌ای را میخواهید)، میتوانید یک activity را آغاز کرده یا چیز جدیدی را برای انجام به آن بدهید.

۲. از اندروید ۵,۰ (API 21) به بعد میتوانید از کلاس JobScheduler[۱] برای زمان‌بندی کارها استفاده کنید. برای اندرویدهای نسخه‌های پایینتر از طریق پاس دادن یک intent به startService()،

[۱] سرویسی است که اجرای عملیات‌ها را در پشت‌زمینه با زمان‌بندی سیستم‌عامل انجام می‌دهد

میتوانید یک سرویس را راه‌اندازی کرده (یا دستورالعمل‌های جدیدی به سرویس درحال اجرا بدهید). میتوانید با پاس دادن یک intent به bindService() به سرویسی متصل شوید.

۳. با پاس دادن یک شی intent به متدهایی از قبیل sendBroadcast()، sendOrderedBroadcast() یا sendStickyBroadcat() آغازگر یک پیام broadcast باشید.

۴. با صدا زدن متد query() روی ContentResolver میتوانید یک عمل پرس و جو از content provider انجام دهید.

۱-۳-۱- فایل manifest

قبل از اینکه سیستم‌عامل بتواند برنامه‌ای را شروع کند، سیستم بایستی با خواندن فایل شناسنامه AndroidManifest.xml بداند که آن جزء از برنامه وجود دارد. برنامه‌ی شما باید تمامی اجزاء خود را در این فایل اعلان نماید، که بایستی در شاخه‌ی ریشه از ساختار پوشه‌ی پروژه قرار گیرد.

فایل manifest علاوه بر اعلان‌کردن اجزاء، وظایف دیگری دارد که از این قرارند:

۱. مشخصه‌ی هر نوع دسترسی که برنامه نیاز دارد، از جمله دسترسی به اینترنت یا خواندن مخاطبین و...

۲. تعیین حداقل سطح API که برنامه برای اجرا نیاز دارد، بر اساس APIهایی که برنامه به‌کار گرفته‌است.

۳. تعیین کتابخانه‌هایی که برنامه بایستی به آنها لینک شود (غیر از Android framework)، مثل کتابخانه ی Google maps، Picasso، Gson و...

اعلان اجزا

وظیفه‌ی اصلی فایل manifest این است که به سیستم درباره‌ی اجزای برنامه اطلاع دهد. به‌عنوان مثال در فایل manifest میتوان یک activity را به‌صورت زیر تعریف کرد:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

شکل (۱-۱) اعلان Activity در فایل manifest

در المنت<application> ویژگی android:icon به مسیر قرارگیری منابع برنامه اشاره می‌کند، جایی که فایل آیکون برنامه در آن قرار گرفته‌است.

در المنت<activity> ویژگی android:name مشخصه‌ی کامل کلاسی را که از Activity ارث‌بری می‌کند، معین می‌کند و ویژگی android:label یک رشته متنی از عنوان این activity که در بالا صفحه قابل مشاهده می‌گردد، تعیین می‌کند.

شما بایستی تمامی اجزای برنامه را با استفاده از المنت‌های زیر در فایل manifest تعریف کنید:

۱. المنت <activity> برای activityها
۲. المنت <service> برای serviceها
۳. المنت <receiver> برای broadcast receiverها
۴. المنت <provider> برای content providerها

activity، service و content providerهایی که شما در سورس کد خود ایجاد میکنید ولی در فایل manifest اعلان نمیکنید، توسط سیستم قابل دسترسی نیستند و متعاقبا هیچوقت اجرا نمی‌شوند. به هر حال broadcast receiverها می‌توانند هم در فایل manifest اعلان شوند و هم به‌صورت پویا داخل کد به‌عنوان یک شی از BroadcastReceiver ایجاد شده و از طریق فراخوانی متد registerReceiver() داخل سیستم ثبت شوند.

اعلان قابلیت‌های اجزا

همانطور که در بالا بحث شد، در فعالسازی اجزا بایستی از پیام Intent برای آغاز activity، service و broadcast receiverها استفاده کنید. میتوانید از یک Intent با نامگذاری آشکار جزء هدف (استفاده از نام کلاس آن جزء) داخل شی intent استفاده کنید. همچنین میتوانید از یک intent ضمنی که نوع آن عمل را مشخص می‌کند و به‌صورت اختیاری داده‌ای که مایل به انجام آن عمل نیاز است استفاده کنید. Intent ضمنی به سیستم این امکان را میدهد که جزئی که عمل مورد نظر را انجام میدهد در دستگاه پیدا کرده و آنرا اجرا کند. اگر اجزای گوناگونی که بتوانند عمل مد نظر intent ارسال‌شده پیدا شد، جزئی که کاربر نیاز داشته باشد انتخاب می‌کند.

سیستم‌عامل اجزایی که بتوانند به یک پیام intent پاسخ دهند را از طریق مقایسه‌ی intent دریافتی با المنت intent-filter که در فایل manifest برنامه‌های دستگاه وجود دارد شناسایی می‌کند.

زمانی که یک activity را داخل manifest اعلان میکنید، میتوانید به‌صورت اختیاری یکسری intent filter که قابلیت‌های آن activity را اعلان می‌کنند استفاده نمایید، بنابراین activity شما می‌تواند به intentهای دیگر برنامه‌ها پاسخ دهد. یک intent filter برای جزء مورد نظرتان را میتوانید با المنت <intent-filter> به‌عنوان فرزند المنت آن جزء اضافه کنید.

به‌عنوان نمونه اگر در حال نوشتن برنامه‌ی ایمیل با یک activity برای نوشتن ایمیل جدید هستید، میتوانید یک intent filter برای پاسخ دادن intent «ارسال ایمیل جدید» همانند تکه کد زیر تعریف کنید:

```
<manifest ... >
    ...
    <application ... >
        <activity android:name="com.example.project.ComposeEmailActivity">
            <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <data android:type="*/*" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

شکل (۲-۱) اعلان intent-filter در فایل manifest

اگر برنامه‌ی دیگر پیام intent با عمل ACTION_SEND ایجاد کند و آن را به متد startActivity() پاس دهد، سیستم‌عامل ممکن است activity شما را شروع کند تا کاربر بتواند ایمیل جدیدی ارسال کند. (جزئیات بیشتر در بخش Intent)

اعلان پیش نیازهای برنامه

انواع متنوعی از دستگاه‌های مجهز به سیستم‌عامل اندروید در بازار وجود دارد و همه‌ی آنها قابلیت‌ها و ویژگی‌های یکسانی ارائه نمی‌کنند. برای جلوگیری از نصب برنامه روی دستگاه‌هایی که فاقد ویژگی‌های مورد نیاز برنامه‌ی شما هستند، مهم است که به‌وضوح یک پروفایل برای انواع دستگاه‌هایی که برنامه‌ی شما را پشتیبانی می‌کنند با اعلان پیش نیازهای نرم افزاری و سخت افزاری در فایل manifest تعریف کنید. بیشتر این اعلان‌ها صرفا جهت اطلاع هستند و سیستم آنها را نمیخواند ولی سرویس‌ها خارجی مثل گوگل پلی به‌منظور ارائه‌ی سیستم فیلتر در حین جستجوی کاربر برای برنامه‌ها نیاز به خواندن آنها دارد.

به‌عنوان مثال اگر برنامه‌ی شما نیازمند دوربین باشد و از APIهای معرفی‌شده در اندروید ۲٫۱ (API 7) استفاده می‌کند، بایستی این پیش نیازها را در فایل manifest به‌صورت زیر تعریف کنید:

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera.any"
                  android:required="true" />
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
    ...
</manifest>
```

شکل (۳-۱) اعلان حداقل و حداکثر نسخه اندروید در فایل manifest

با اعلان‌هایی که در شکل بالا نشان داده‌شده، دستگاه‌هایی که دوربین نداشته باشند و یا نسخه‌ی اندروید آنها پایینتر از ۲٫۱ باشد نمی‌توانند برنامه‌ی شما را از گوگل پلی نصب کنند. به هر حال میتوانید اعلان کنید که برنامه‌ی شما از دوربین استفاده می‌کند ولی الزامی نیست. در این صورت بایستی ویژگی required را false قرار داده و داخل کد در زمان اجرا بررسی کنید که دستگاه دوربین داشته باشد و در صورتی که فاقد دوربین بود ویژگی‌های مربوط به آن را غیر فعال نمایید.

۲-۳-۱- منابع برنامه[2]

اپلیکیشن اندروید از اجزایی فراتر از کد تشکیل‌شده، که نیازمند منابعی از قبیل تصاویر، فایل‌های صوتی و هر چیزی که مربوط به نمایش ظاهر برنامه باشد، که از سورس کد جدا شده‌است. مثلا میتوانید انیمیشن‌ها، منوها، استایل‌ها، رنگ‌ها و طرح رابط کاربری activityها را با استفاده از فایل‌های xml تعریف کنید. استفاده از منابع (resource) در برنامه، به‌روزرسانی مشخصه‌های مختلفی از برنامه را بدون تغییر کد میسر میسازد. ارائه‌ی مجموعه‌ای از منابع جایگزین شما را قادر به بهبود برنامه در انواع پیکربندی‌ها از قبیل زبان‌ها و اندازه صفحه نمایش‌های مختلف می‌کند.

به‌ازای هر منبع که در پروژه اضافه میکنید، ابزار SDK build tools یک شناسه‌ی عدد صحیح تعریف می‌کند که از طریق آن میتوانید آن منبع را داخل سورس کد برنامه یا دیگر منابعی که در قالب xml تعریف شده‌اند، قید نموده و مورد استفاده قرار دهید. به‌عنوان مثال اگر برنامه‌ی شما محتوی یک فایل تصویری به‌نام logo.png (ذخیره‌شده در مسیر /res/drawable) باشد، ابزار SDK یک شناسه‌ی با نام R.drawable.logo تولید می‌کند. این شناسه به یک عدد صحیح مختص برنامه‌ی شما انتساب یافته که میتوانید از آن برای قیدکردن تصویر و قرار دادن آن در رابط کاربری استفاده

کنید.

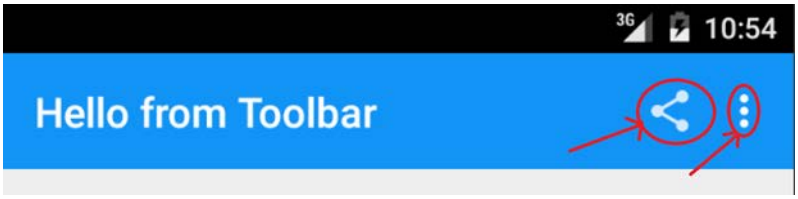
یکی از مهم‌ترین جنبه‌های ارائه‌ی منابعی که از سورس کد برنامه جدا باشد، قابلیت ارائه‌ی منابع جایگزین برای دستگاه‌های با پیکربندی‌های متنوع است. به‌عنوان مثال با انتقال رشته‌های متنی استفاده‌شده در رابط کاربری به داخل فایل xml، میتوانید این رشته‌ها را به‌زبان‌های دیگر ترجمه کرده و آنها را داخل فایل‌های جداگانه ذخیره کنید. سپس سیستم‌عامل رشته‌های متناسب با زبان دستگاه را بر اساس مشخصه‌ی زبان که در نام منابع قید کرده‌اید)مثلا res/values-fa/ برای رشته‌های زبان فارسی) در رابط کاربری نمایش میدهد.

سیستم‌عامل اندروید مشخصه‌های گوناگونی را برای منابع جایگزین پشتیبانی می‌کند. این مشخصه عنوان کوتاهی است که در نام مسیر منابع خود ذکر میکنید که به‌منظور تعریف پیکر بندی دستگاه برای تصمیم گیری اینکه کدام منابع باید مورد استفاده قرار گیرند، می‌باشند. به‌عنوان مثال بایستی طرح‌های مختلفی را برای activityها ایجاد کنید که بر اساس اندازه و جهت صفحه نمایش مورد استفاده قرار گیرند. زمانی که صفحه نمایش دستگاه در حال portrait باشد ممکن است طرحی با دکمه‌های عمودی بخواهید، ولی زمانی که صفحه نمایش در حالت landscape باشد بخواهید دکمه‌ها به‌حالت افقی به‌صف شوند. برای تغییر طرح صفحه برا اساس جهت صفحه نمایش، میتوانید دو طرح مختلف تعریف کرده و نام متناسب با مشخصه‌ی خود را به مسیر هر کدام از طرح‌ها اعمال کنید. سپس سیستم‌عامل به‌طور خودکار طرح متناسب را بر اساس جهت صفحه نمایش انتخاب می‌کند

انواع منابع xml در اندروید

۱. Layout: این فایل به‌منظور طراحی ظاهر صفحه‌ای که در Activity نمایش داده می‌شود به‌کار می‌رود.

۲. Menu: اطلاعات مربوط به منوی یک صفحه که معمولا در نوار بالای صفحه به‌صورت دکمه و یا داخل قسمت سه نقطه قابل مشاهده است.



شکل (۴-۱) منوی toolbar

۳. Animator: نمایش یک انیمیشن برای المان‌های یک صفحه با این فایل انجام می‌شود، مثلا یک دکمه بعد از لمس شدن حرکت دورانی انجام دهد.

۴. Color: برای متمرکزکردن رنگهای استفاده‌شده در برنامه و همچنین استفاده مجدد از آنها در کدها

۵. Strings: برای متمرکزکردن رشته‌های استفاده‌شده در برنامه برای استفاده مجدد و همچنین امکان چندزبانه‌کردن برنامه

۶. Styles: تم‌ها و رنگ بندی المان‌های صفحات در این فایل نگه داری می‌شود

محل قرار گیری این فایل‌های xml در پروژه

۷. Layout: ~\app\src\main\rec\layout\

۸. Menu: ~\app\src\main\rec\menu\

۹. Animator: ~\app\src\main\rec\anim\

۱۰. Color: ~\app\src\main\rec\values\colors.xml

۱۱. Strings: ~\app\src\main\rec\values\strings.xml

۱۲. Styles: ~\app\src\main\rec\values\styles.xml

۴-۱- Activity: فعالیتی در قالب یک صفحه[3]

Activity یکی از اجزای اصلی در اندروید به‌حساب می‌آید که نقطه‌ی شروع برنامه[1]از این جنس است و وظیفه‌ی نمایش یک صفحه‌ی گرافیکی برای کاربر را دارد که فعالیت‌های کاربر روی آن صفحه از جمله لمس‌کردن یک دکمه، اسکرول‌کردن صفحه به بالا، پایین، چپ و راست، واردکردن اطلاعات در صفحه و... همگی با فرآخوانی یکسری متد در Activity مربوطه انجام می‌شود.

اولین صفحه‌ای که با اجرای برنامه نمایش داده می‌شود launcher activity می نامند. بقیه‌ی

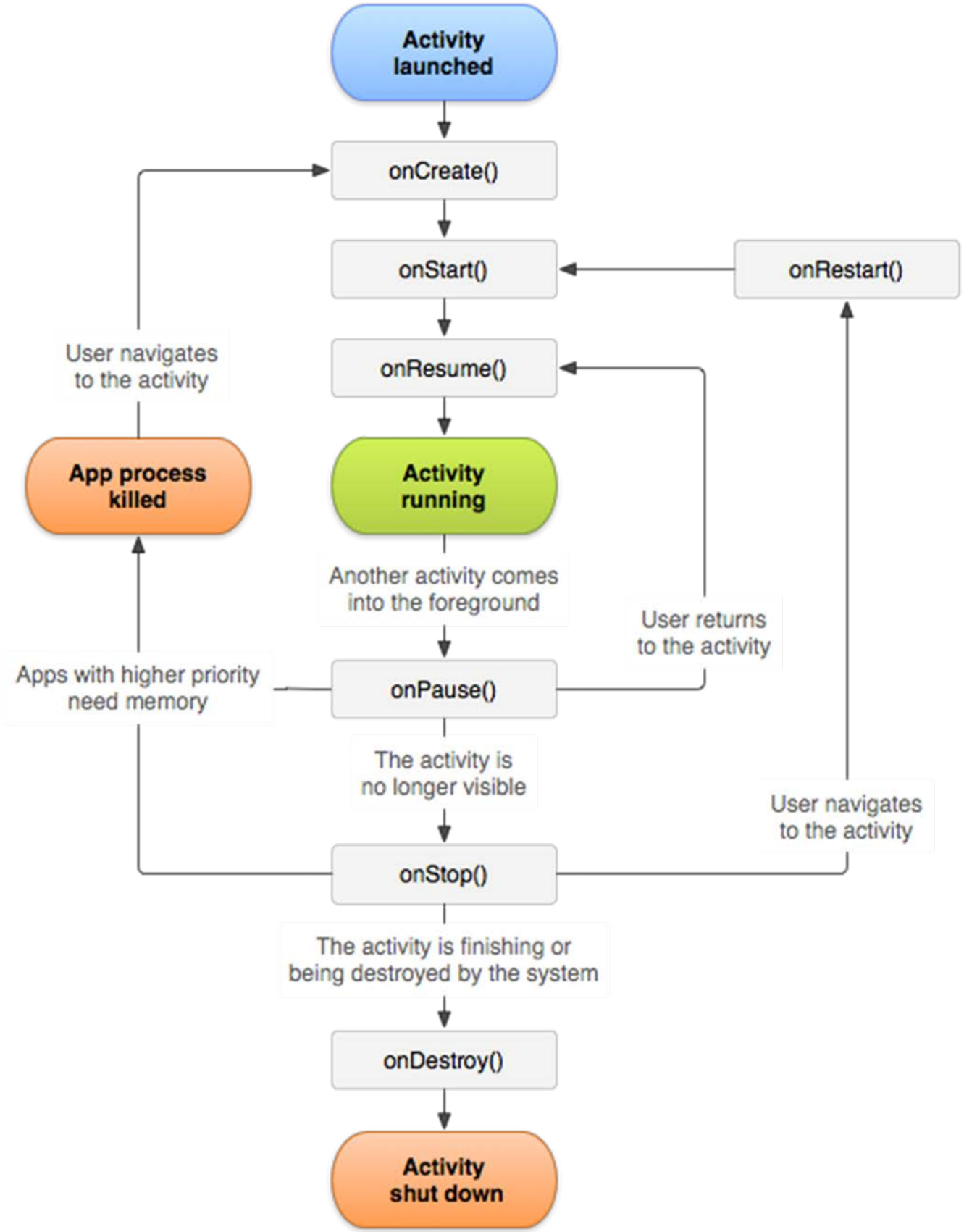
[1] Entry Point

صفحات برنامه بر اثر رویدادهایی که در این صفحه اتفاق می‌افتد، (مثلا کاربر دکمه‌ای را لمس می‌کند) راه‌اندازی می‌شوند. بین صفحات برنامه حداقل وابستگی ممکن وجود دارد و یا گاهی هیچ ارتباطی با یکدیگر ندارند، به‌عنوان مثال با زدن دکمه‌ی نوشتن ایمیل جدید کاربر از صفحه‌ی فعلی به صفحه‌ی مربوط به نوشتن ایمیل جدید هدایت می‌شود و بین این دو صفحه هیچ ارتباطی وجود ندارد. در مثالی دیگر کاربر متن ایمیل خود را نوشته و دکمه‌ی ارسال را لمس می‌کند، در اینجا کاربر از صفحه‌ی نوشتن ایمیل به صفحه‌ی انتظار برای ارسال ایمیل هدایت می‌شود، اطلاعاتی که بین این دو صفحه جابجا می‌شود می‌تواند محتوای ایمیلی که کاربر در صفحه‌ی نوشتن ایمیل جدید وارد کرده باشد و در صفحه‌ی انتظار برای ارسال، مورد استفاده قرار می گیرند.

هر Activity یک کلاس جاوا به‌همراه یک سری فایل xml برای طراحی ظاهر صفحه‌ی مدنظر می‌باشد.(تمام منطق برنامه داخل کلاس‌های جاوا انجام می‌شود و از فایل‌های xml در این کلاس جاوا، رفرنس گرفته و به‌صورت یک شی رفتار می‌گردد)

فرآیند طی‌شده از ایجاد تا پایان یک Activity (Activity lifecycle)

با شروع یک صفحه در اندروید ابتدا کلاس جاوای Activity مربوط به آن بارگیری‌شده و داخل حافظه Ram قرار میگیرد و هر کدام از متدهای زیر تحت شرایط معین، به‌منظور نمایش یا عدم نمایش صفحه‌ی مربوطه فراخوانی می‌شوند:



شکل (۵-۱) چرخه‌ی طول‌عمر Activity

متدهای فرخوانی‌شده در Activity

۱. onCreate(): به‌محض راه‌اندازی صفحه، این متد، اولین متدی است که فراخوانی می‌شود. در این متد کارهای مربوط به آماده سازی ظاهر صفحه، مقدار دهی اولیه‌ی فیلدها و configurationهای کلاس انجام می‌شود.

(با اتمام این متد هنوز صفحه قابل مشاهده نیست و متد بعدی یعنی onStart() فراخوانی می‌شود)

۲. onStart(): این متد که بعد از onCreate() فراخوانی می‌شود به‌منظور مقدار دهی به المان‌های مربوط به ظاهر برنامه و نهایی سازی آن مورد استفاده قرار میگیرد.

(به‌محض خروج از این متد، activity برای کاربر قابل مشاهده است ولی هنوز قابل تعامل نیست (به‌اصطلاح focus روی این صفحه نمی‌باشد) و برای اینکه قابل تعامل شود متد بعدی یعنی onResume() فراخوانی می‌گردد)

۳. onResume(): این متد که بعد از onStart() فراخوانی می‌شود اغلب به‌منظور استفاده از برخی عملکردهای مربوط به سیستم‌عامل از جمله اتصال صفحه به یک رویداد سیستمی مثل تماسها و... مورد استفاده میگیرد.

(به‌محض خروج از این متد، کاربر می‌تواند با صفحه تعامل کند و برنامه به‌حالت در حال اجرا می‌رود و تا زمانی که یک رویداد اتفاق بیافتد ادامه دارد)

۴. onPause(): زمانی که نیاز است که activity دیگری نمایش داده شود، مثلا وقتی که کاربر دکمه‌ی نوشتن ایمیل جدید را لمس می‌کند یا به یک صفحه‌ی دیگر در برنامه‌ی دیگر می‌رود، اولین متدی که برای خروج از صفحه‌ی فعلی فراخوانی می‌شود onPause() می‌باشد. از این متد اغلب به‌منظور قطع اتصال به یک رویداد سیستمی مثلا تماسها و... استفاده می‌گردد.

اگر کاربر به صفحه‌ی قبلی برگردد متد onResume() از آن صفحه صدا زده می‌شود (چرا که نیاز به برقراری تعامل با آن صفحه می‌باشد)

(به‌محض ورود به این متد، دیگر کاربر نمی‌تواند با صفحه تعامل کند ولی صفحه هنوز قابل مشاهده است، لذا متد بعدی یعنی onStop() فراخوانی می‌شود)

۵. onStop(): این متد که بعد از onPause() فراخوانی می‌شود، اغلب اوقات برای کارهای حسابرسی مثلا ثبت زمان طی‌شده در صفحه یا ارسال یک رویداد برای Google Analytics مورد استفاده قرار میگیرد.

(به‌محض ورود به این متد، دیگر صفحه قابل مشاهده نیست و activity داخل پشته‌ی سیستم‌عامل به‌منظور بازیابی در آینده قرار میگیرد و activity دیگر به مرحله‌ی راه‌اندازی می‌رود)

۶. ()onRestart: اگر کاربر بخواهد به این صفحه برگردد (مثلا با زدن دکمه‌ی history این صفحه را انتخاب کند) این متد فراخوانی می‌گردد.

در این متد اغلب کارهای مربوط به بازیابی حالت (state) صفحه استفاده می‌شود تا با اطلاعات قبل از محو شدن صفحه مقدار دهی شوند.

بعد از این متد بلافاصله متد ()onStart به‌منظور آماده سازی برای نمایش صفحه فراخوانی می‌گردد.

۷. ()onDestroy: این متد تحت دو شرط فراخوانی می‌گردد. اول اینکه کار یک صفحه تمام‌شده باشد و به‌ترتیب متدهای ()onPause و ()onStop صدا زده شوند و دیگر نیازی به این صفحه نباشد. دوم اینکه activity داخل پشته‌ی سیستم‌عامل قرار دارد و activityهای دیگری نیاز به حافظه دارند. در این دو حالت ابتدا متد ()onDestroy فراخوانی می‌گردد تا کارهای مربوط به آزاد سازی منابع انجام گیرند.

بعد از اتمام این متد activity از حافظه Ram خارج شده و منابع آن آزاد می‌گردند.

تذکر: در هر یک از دو حالت ()onPause که صفحه قابل تعامل نیست و ()onStop که صفحه قبال مشاهده نیست و یا در پشته‌ی سیستم‌عامل قرار دارد، اگر سیستم‌عامل نیاز شدید به حافظه داشته باشد و میخواهد صفحه‌ای با اولویت بالا را نمایش دهد ولی حافظه کافی نباشد، برنامه فعلی را کشته و منابع آن را از حافظه خارج می‌کند تا صفحه‌ی اولویت بالا را بتواند نمایش دهد.

۵-۱- Intent: پیامی بین اجزای برنامه[4]

Intent یک شی (object) است که حاوی پیامی برای درخواست یک عمل یا ارسال آن پیام به جزء دیگر می‌باشد. اگرچه Intentها ارتباطات بین کامپوننت‌ها را به طرق مختلفی پایه گذاری می‌کنند، سه نمونه کاربرد پایه‌ای آن از این قرار است:

۱. شروع یک Activity

با پاس دادن یک شی Intent به متد ()startActivity (قابل دسترسی از componentهای اصلی اندروید)، میتوانیم هر گونه اطلاعات مورد نیاز آن صفحه را ارسال کنیم

۲. راه‌اندازی یک Service

سرویس یکی از اجزای (component) اصلی در اندروید می‌باشند که فعالیتی را در پس زمینه انجام میدهد. از سرویس میتوان برای انجام کارهایی مثل دانلود یک فایل، پخش موسیقی، همگام سازی با سرور و... استفاده کرد. با پاس دادن یک شی Intent به متد ()startService (قابل دسترسی از componentهای اصلی اندروید) میتوانیم سرویس جدیدی را راه‌اندازی نماییم.

۳. انتشار پیام همگانی یا broadcast

broadcast پیامی است که هر اپلیکیشنی در دستگاه قادر به دریافت آن می‌باشد. به‌عنوان نمونه خود سیستم اندروید در هر لحظه انواع گوناگونی از پیام‌های broadcast را در قالب رویدادهای سیستمی انتشار میدهد، از قبیل پیام بوت شدن سیستم، روشن شدن وایفای، اتصال شارژر، هنذفری و... که همه‌ی اپلیکیشن‌های نصب‌شده روی دستگاه این پیام‌ها را میبینند. با پاس دادن یک شی Intent به متد ()sendBroadcast یا ()sendOrederedBroadcast (قابل دسترسی از componentهای اصلی اندروید) میتوانید پیام broadcast انتشار دهید.

انواع Intent

۱. Intent آشکار

قیدکردن نام کامل مقصد (حاوی اسم کلاس) یا کامپوننتی که قرار است راه‌اندازی شود.

از این نوع زمانی استفاده می‌شود که مقصد intent داخل برنامه خودمان باشد، زیرا اسم کلاس یک activity یا سرویس را میدانیم. به‌عنوان نمونه وقتی کاربر دکمه‌ای را لمس می‌کند در پاسخ به این عمل میتوانیم activity جدیدی را از برنامه خودمان شروع کنیم و یا سرویسی را برای دانلود فایل راه‌اندازی نماییم.

۲. Intent ضمنی

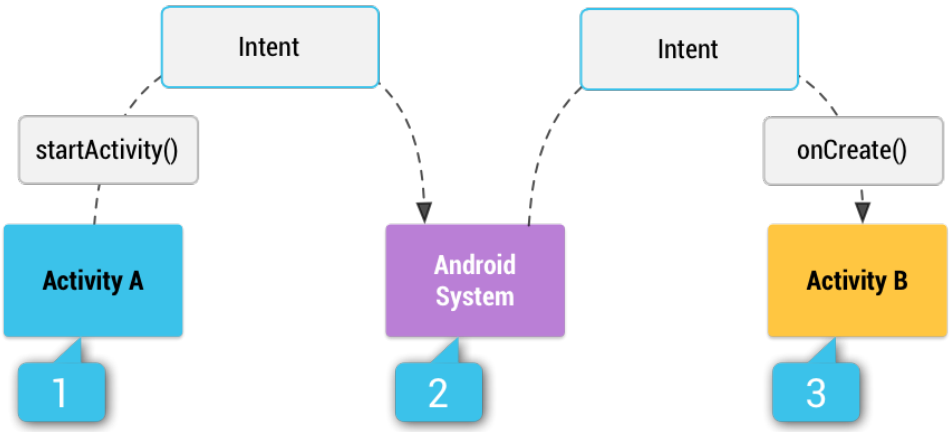
نام کامل مقصد قید نمی‌شود، بلکه در عوض یک action عمومی (عملی که بین برنامه‌ها استاندارد شده، مثل: نمایش یک تصویر) درج می‌شود تا به برنامه‌های دیگر اجازه دهد که این عمل را انجام دهند.

به‌عنوان نمونه اگر بخواهیم مکان یک کاربر روی نقشه را نمایش دهید میتوانید از یک intent ضمنی استفاده کرده و در آن action مربوط به نمایش مکان کاربر را درج کنید، در این صورت هر برنامه‌ای که قادر به نمایش مکان روی نقشه باشد می‌تواند این عمل را انجام دهد. مثلا برنامه‌های google maps و waze کاندیدهای نمایش مکان هستند.

زمانی که از یک Intent آشکار برای شروع activity یا سرویس جدید استفاده میکنیم، سیستم

اندروید بی درنگ کامپوننت درخواستی را اجرا می‌کند.

در اینجا چگونگی انتقال پیام بین دو activity از طریق Intent ضمنی به‌تصویر کشیده شده‌است.



شکل (۶-۱) فرآیند شروع یک activity جدید و انتقال پیام با Intent ضمنی

۱. Activity A قصد شروع Activity B را دارد لذا یک شی از Intent را به متد startActivity() پاس میدهد.

۲. متد startActivity() که در کلاس‌های پدر activity تعریف شده و با سیستم‌عامل و متدهای سطح پایین آن ارتباط دارد، از سیستم‌عامل درخواست جستجو برای تمامی برنامه‌هایی که بتوانند این نوع از Intent را دریافت کنند، انجام میدهد.

۳. زمانی که تطابق پیدا شد (برنامه‌ای که بتواند به این Intent پاسخ دهد)، سیستم‌عامل activity منطبق (Activity B) را با صدا زدن متد onCreate() آن و پاس دادن Intent مذکور به آن، راه‌اندازی می‌کند.

تذکر: برای اینکه برنامه‌ی خود را به لیست برنامه‌هایی که قادر به دریافت Intentهای ضمنی (مثلا نمایش مکان روی نقشه) هستند، ثبت کنیم بایستی یکسری تگ intent-filter به فایل AndroidManifest.xml واقع در مسیر زیر اضافه کنیم: ~\app\src\main\AndroidManifest.xml (جزئیات بیشتر در بخش AndroidManifest)

۶-۱- Service: اجرای پشت زمینه[5]

گاهی اوقات عملیات‌هایی وجود دارند که میخواهید علی رغم اینکه کاربر با برنامه کار می‌کند یا نه، به اجرای خود ادامه دهند. مثلا اگر دانلود فایلی را آغاز کرده‌اید، نمیخواهید زمانی که کاربر برنامه‌ی دیگری را باز می‌کند، متوقف شود. در این مواقع بهترین راه حل که سیستم‌عامل اندروید ارائه داده استفاده از Service است، که هم در کنترل مصرف منابع سیستم و هم بهبود معماری اپلیکیشن در جهت کاهش پیچیدگی راهکار مناسبی می‌باشد.

همانطور که در بخش‌های قبلی اشاره شد Service یکی از اجزای اصلی در اندروید است که وظیفه‌ی اجرای عملیات‌های طولانی در پشت زمینه بدون رابط کاربری دارد. دیگر اجزا می‌توانند این جزء را شروع کنند و با راه‌اندازی آن اجرای برنامه ادامه میابد حتی زمانی که کاربر داخل اپلیکیشن نباشد. علاوه بر این یک جزء می‌تواند به سرویس متصل شده و با آن تعاملاتی را داشته باشد و حتی از طریق ارتباطات بین فرآیندی[1] اطلاعاتی را با آن جابجا کند.

مثلا یک سرویس می‌تواند تراکنش‌های شبکه، پخش موزیک، عملیات‌های ورودی/خروجی یا تعامل با یک content provider را همگی در پشت زمینه کنترل کند.

سه نوع مختلف از Service وجود دارد:

۳. پیش زمینه (Foreground): سرویس پیش زمینه برخی از عملیات‌ها را که برای کاربر قابل مشاهده است انجام میدهد. به‌عنوان مثال یک برنامه‌ی موزیک می‌تواند از یک سرویس پیش زمینه برای پخش موزیک استفاده کند. سرویس‌های پیش زمینه بایستی یک آیکون در نوار وضعیت نشان دهند. این سرویس‌ها به اجرا ادامه میدهند حتی زمانیکه کاربر با برنامه تعامل ندارد.

۴. پس زمینه (Background): سرویس پس زمینه عملیات‌هایی را انجام میدهد که کاربر به‌طور مستقیم از وجود آنها بی اطلاع است. به‌عنوان مثال اگر یک برنامه از سرویسی برای فشرده سازی فضای ذخیره سازی خود استفاده کند معمولا از طریق سرویس پس زمینه این عمل را انجام میدهد.

۵. مقید (Bound): سرویس مقید نامیده می‌شود زمانی که یک جزء از برنامه با فراخوانی متد

[1] Inter process communication

bindService() به آن سرویس متصل می‌شود. یک سرویس مقید که رابط کلاینت سرور ارائه میدهد، بقیه‌ی اجزا را قادر به تعامل با سرویس در جهت ارسال درخواست، دریافت نتیجه و حتی عملیات‌ها بین فرآیندی (IPC) می‌کند. سرویس مقید تا زمانی اجرا می‌شود که دیگر اجزای برنامه به آن متصل باشند. چندین جزء می‌توانند همزمان به این سرویس متصل شوند، ولی زمانی که همه‌ی آنها اتصال را قطع کنند، سرویس خاتمه میابد.

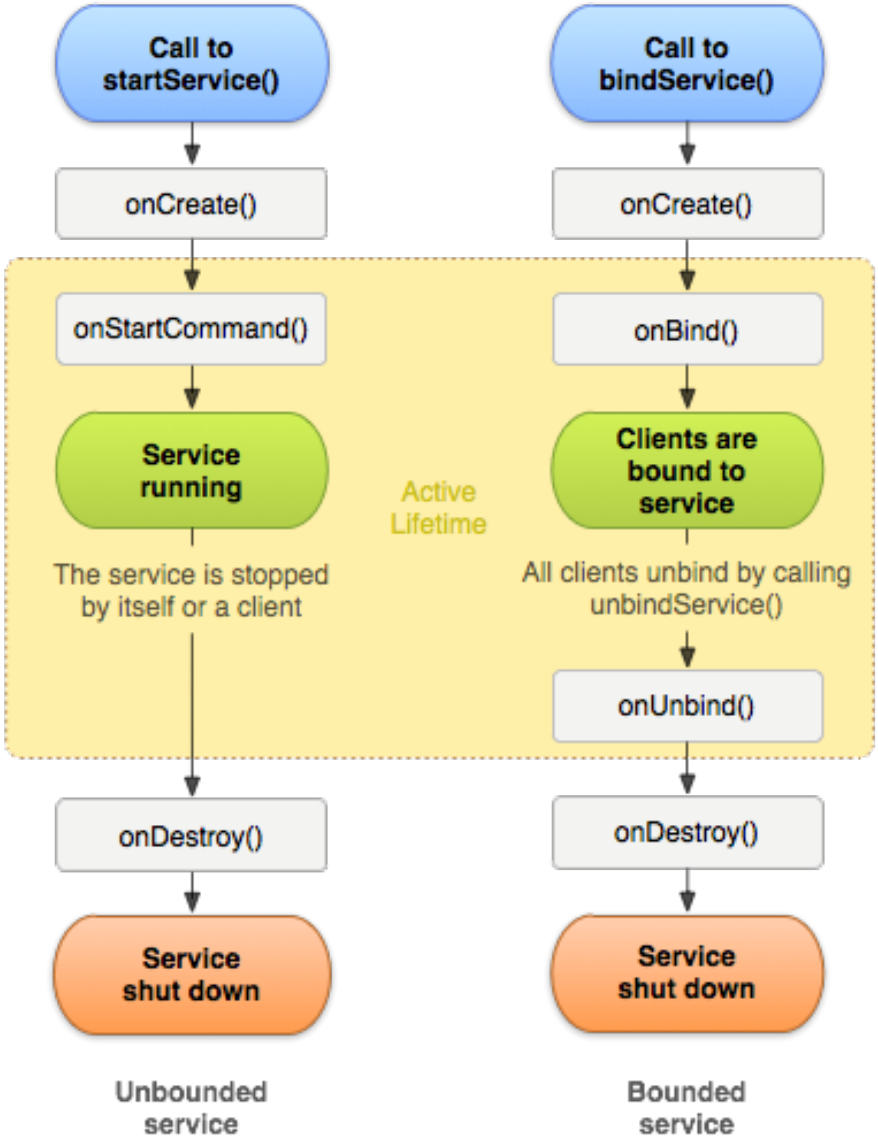
اگرچه در این سند سرویس‌های از نوع Started و Bound را به‌طور جداگانه مورد بحث قرار داده‌ایم، ولی سرویس شما می‌تواند به هر دو روش عمل کند. می‌تواند شروع شود (Started) و تا بینهایت اجرا گردد و همچنین می‌تواند اجازه‌ی اتصال بدهد.

اعمال‌کردن این روش‌ها صرفا به پیاده سازی یک سری متدهای callback بستگی دارد: onStartCommand() برای اجازه دادن شروع سرویس و onBind() برای اجازه‌ی اتصال به سرویس می‌باشند.

علی رغم اینکه اپلیکیشن از نوع Started، Bound و یا هر دو باشد، هر جزئی از برنامه می‌تواند از سرویس استفاده کند (حتی از داخل برنامه‌ای دیگر) به همان طریق که هر جزئی می‌تواند از یک activity با آغاز یک Intent استفاده کند. به هر حال میتوانید سرویس را به‌صورت خصوصی در فایل manifest اعلان کنید و دسترسی از اپلیکیشن‌های دیگر را به روی این سرویس ببندید.

تذکر: سرویس روی thread اصلی فرآیند میزبان اجرا می‌شود. این سرویس thread اختصاصی خودش را ایجاد نمی‌کند و روی فرآیند دیگری اجرا نمی‌شود، مگر اینکه خودتان اعلان کرده باشید. اگر سرویس شما قرار است که فعالیت‌های سنگین پردازشی و یا عملیات‌های بلاک کننده مثل پخش موزیک یا ارتباط با شبکه انجام دهد، بایستی این سرویس را روی thread جدید ایجاد کنید تا فعالیت شما را به‌اتمام برساند. با استفاده از thread جداگانه میتوانید خطر هشدار «اپلیکیشن پاسخ نمیدهد» (زمانی که کدهای مربوط به نمایش و به‌روزرسانی رابط کاربری به خاطر فعالیت‌های بلاک کننده روی پردازنده اجرا نمی‌شوند) را کاهش داده و با اینکار thread اصلی اپلیکیشن را به‌صورت اختصاصی در اختیار فعالیت‌های مربوط به نمایش رابط کاربری قرار دهید.

طول عمر یک سرویس را در شکل زیر مشاهده میکنید:



شکل (۷-۱) چرخه‌ی طول‌عمر Service

۷-۱- Broadcast Receiver: پیام همگانی[6]

برنامه‌های اندرویدی می‌توانند برای سیستم یا برنامه‌های دیگر پیغام‌هایی در جهت اعلان یک رویداد ارسال کنند که به این پیام‌ها broadcast میگویند. ارسال و دریافت این پیام‌ها شباهت زیادی به الگوی طراحی publish-subscriber دارد. به این صورت که زمانی که فرآیندی قصد ارسال پیام broadcast را داشته باشد، لیستی از فرآیندها در انتظار دریافت این پیام هستند و به‌محض انتشار، دستورالعمل‌های خود را متناسب با نوع پیام انجام می‌دهند. به‌عنوان مثال هنگامی که کابل شارژر به گوشی متصل می‌شود سیستم به برنامه‌هایی که در لیست دریافت این اعلان هستند اطلاع میدهد.

نمونه‌های پرکاربرد دیگر از این قبیل هستند: اتصال وایفای، اتصال داده همراه، بوت شدن تلفن، کم بودن سطح باتری، اتمام دانلود فایل، دریافت پیامک، اتصال کارت حافظه و...

گوش دادن به پیام‌های broadcast

سیستم‌عامل اندروید برای این منظور دو روش تعبیه کرده:

۱. اعلان در فایل manifest.xml

در این روش نام پیامی که میخواهیم به آن گوش دهیم را با استفاده از تگ broadcast و فیلترهای متناسب داخل فایل manifest به این صورت اعلان میکنیم:

```
<receiver android:name=".MyBroadcastReceiver"  android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <action android:name="android.intent.action.INPUT_METHOD_CHANGED" />
    </intent-filter>
</receiver>
```

شکل (۸-۱) اعلان broadcast در فایل manifest

در اینجا یک broadcastReceiver با دو فیلتر «BOOT_COMPLETED» برای دریافت پیام کامل شدن بوت سیستم و «INPUT_METHOD_CHANGED» برای زمانی که نوع ورودی تغییر می‌کند مورد استفاده قرار گرفته.

برای اینکه بتوانیم دستورالعمل‌های خودمان را هنگام دریافت این پیام‌ها انجام دهیم بایستی یک کلاس جاوا با نامی که در بالا اعلان کردیم ایجاد کنیم

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    private static final String TAG = "MyBroadcastReceiver";
    @Override
    public void onReceive(Context context, Intent intent) {
        StringBuilder sb = new StringBuilder();
        sb.append("Action: " + intent.getAction() + "\n");
        sb.append("URI: " + intent.toUri(Intent.URI_INTENT_SCHEME).toString() + "\n");
        String log = sb.toString();
        Log.d(TAG, log);
        Toast.makeText(context, log, Toast.LENGTH_LONG).show();
    }
}
```

شکل (۹-۱) کلاس BroadcastReceiver

این کلاس با ارث بری از کلاس BroadcastReceiver و اعلان شدن در فایل manifest.xml واجد

دریافت پیام‌های broadcast می‌شود. این پیام‌ها از طریق متد onReceive به دست ما میرسند که با استفاده از دو پارامتر آن یعنی Context و Intent اطلاعات پیام را استخراج میکنیم.

Context اطلاعات عمومی فرستنده (اعم از وضعیت اجرا، منابع متنسب‌شده، دسترسی‌ها و thread اصلی) و Intent خود پیام را حمل می‌کنند.

۲. اعلان در یکی از اجزای اصلی (activity، service)

در این روش میتوانیم در حین اجرا بر اساس نیاز به پیام‌ها گوش دهیم، بدون اینکه نیاز باشد از قبل در فایل manifest.xml اعلان کرده باشیم. برای این منظور به این صورت عمل میکنیم:

ابتدا یک شی از کلاس BroadcastReceiver ایجاد میکنیم

```
BroadcastReceiver br = new MyBroadcastReceiver();
```

شکل (۱۰-۱) ساخت شی جدید از BroadcastReceiver

سپس با استفاده از متدی که در اجزای اصلی تعبیه شده برای دریافت اعلان ثبت نام میکنیم

```
IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
intentFilter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);
this.registerReceiver(br, filter);
```

شکل (۱۱-۱) ثبت‌نام برای دریافت Broadcast

به این ترتیب میتوانیم با استفاده از شی br پیام‌ها را دریافت کنیم.

از آنجایی که کلاس BroadcastReceiver کلاسی abstract است، بایستی بدنه‌ی این کلاس را پیاده سازی کرده و متد abstract آن یعنی onReceive را override کنیم تا بتوانیم به پیام‌ها گوش دهیم.

پیام‌ها تا زمانی به دست ما میرسند که اجزا در حالت اجرا باشند، لذا اگر اجزا طول عمر (lifecycle) خود را طی کنند و به مرحله‌ی نابودی برسند دیگر قادر به دریافت این پیام‌ها نیستند و بایستی بعد از ورود به چرخه طول عمر دوباره اقدام به ثبت نام کنند.

برای لغو ثبت نام از متد unregisterReceiver(android.content.BroadcastReceiver) استفاده میکنیم

فصل ۲:

عملکرد برنامه سمت اندروید

۱-۲- کلیات

این پروژه به‌زبان جاوا و در محیط توسعه‌ی Android Studio طراحی و پیاده سازی شده. این ابزار توسط شرکت JetBrains طراحی شده و زیر نظر گوگل توسعه یافته و پشتیبانی رسمی می‌گردد. طبق گزارش فروشگاه‌های اپلیکیشن مطرح همچون گوگل پلی و کافه بازار، (تا تاریخ ۱ دی ۱۳۹۶) ۹۹/۸درصد کاربران از اندروید بالای ۴/۱ استفاده می‌کنند، به این ترتیب حداقل نسخه‌ی اندروید قابل پشتیبانی توسط برنامه، نسخه‌ی API 16 یا Android 4.1 (Jelly bean) انتخاب گردید تا استفاده از قابلیت‌ها و ابزارهای به‌روز سهولت یابد.

۱-۱-۲- دسترسی‌های برنامه

به‌منظور استفاده از منابع سیستم‌عامل و دستگاه، نیاز به درخواست یکسری مجوز دسترسی در برنامه وجود دارد:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

شکل (۱-۲) اعلان دسترسی‌های برنامه در فایل manifest

۱. INTERNET: اتصال به اینترنت

۲. ACCESS_NETWORK_STATE: خواندن وضعیت اتصال به شبکه تلفن همراه

۳. ACCESS_WIFI_STATE: خواندن وضعیت اتصال به وایفای

۴. READ_CONTACTS: خواندن مخاطبین دستگاه

۵. WRITE_CONTACTS: نوشتن مخاطبین روی دستگاه

۶. WRITE_EXTERNAL_STORAGE: نوشتن روی حافظه دستگاه

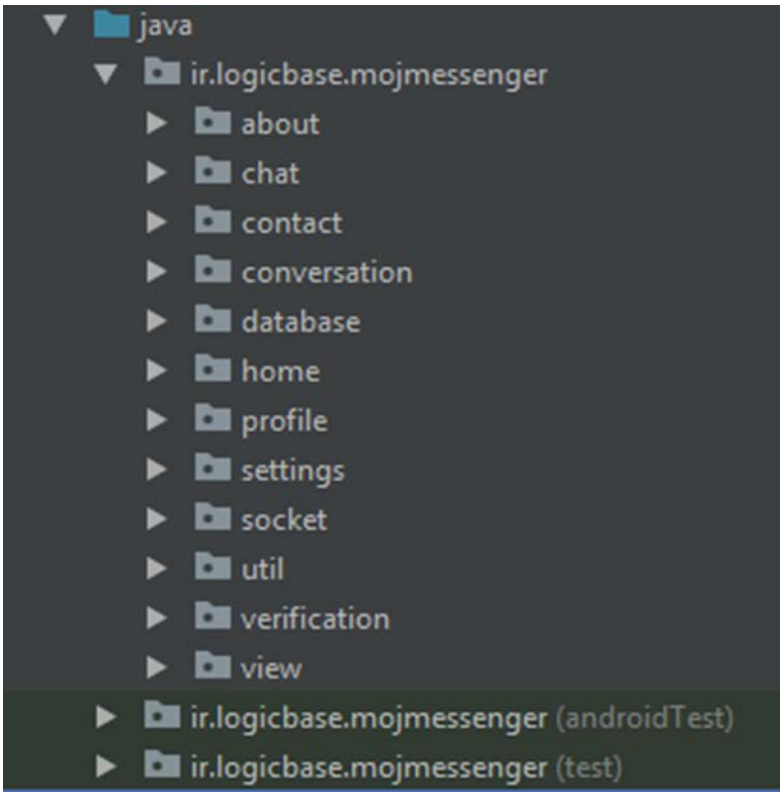
۷. READE_EXTERNAL_STORAGE: خواندن اطلاعات از حافظه دستگاه

با معرفی اندروید ۶، گوگل به‌منظور بالابردن امنیت کاربران، سیاست درخواست دسترسی اپلیکیشن‌ها را تغییر داد. با این تغییر لیستی از دسترسی‌های خطرناک تهیه گردید (این دسترسی‌ها

حریم خصوصی کاربر را در معرض خطر قرار میدهند که به‌عنوان مثال میتوان به دسترسی حافظه، مکان، مخاطبین و دوربین اشاره کرد) که دیگر در حین نصب برنامه کسب نمی‌شوند، و نیاز به کسب آنها در حین اجرای برنامه دارند. با این حال دسترسی‌هایی نیز مجاز شمرده شد (از جمله اتصال به اینترنت) و با نصب برنامه کسب می‌گردند.

مجوز سه دسترسی اول (خواندن وضعیت و اتصال به اینترنت) در حین نصب برنامه کسب می‌شود و چهار دسترسی آخر (خواندن و نوشتن روی حافظه و مخاطبین) در حین اجرای برنامه از کاربر درخواست می‌گردند.

۲-۱-۲- پکیج‌بندی



شکل (۲-۲) پکیج‌های برنامه اندروید

پروژه از یکسری پکیج‌های جاوا تشکیل شده که به‌شرح زیر است:

۱. about: صفحه‌ی درباره‌ی ما

۲. chat: صفحه‌ی لیست مکالمات

۳. contact: صفحه‌ی لیست مخاطبین

۴. conversation: صفحه‌ی مکالمه

۵. database: لایه‌ی پایگاه داده

۶. home: صفحه‌ی خانه

۷. profile: صفحه‌ی پروفایل کاربر

۸. settings: صفحه‌ی تنظیمات برنامه

۹. socket: لایه‌ی شبکه و اتصالات سوکت

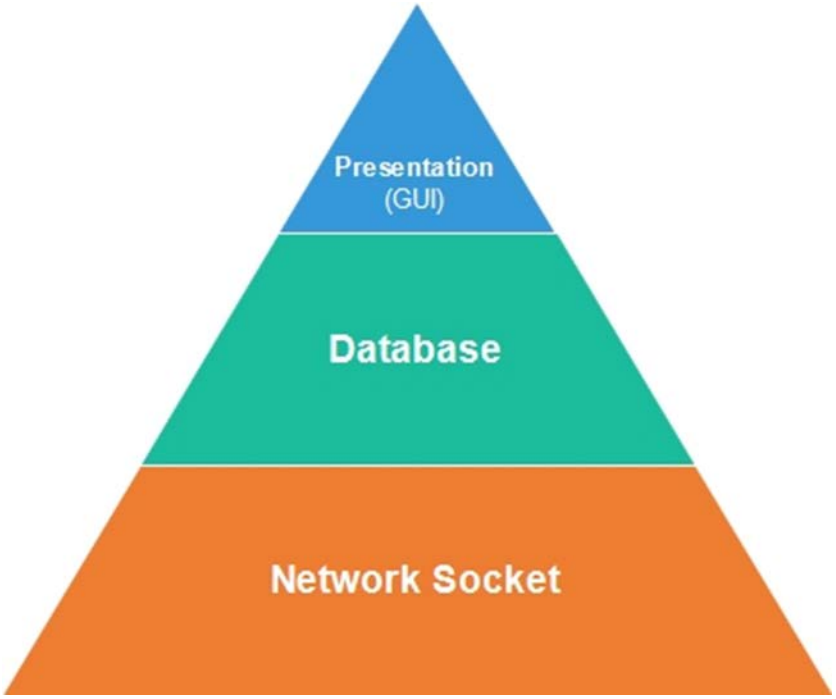
۱۰. util: ابزارهای کاربردی

۱۱. verification: صفحه‌ی احراز هویت کاربر (از طریق پیامک)

۱۲. view: صفحات شخصی سازی‌شده از نسخه‌ی پیشفرض اندروید

۲-۲- لایه‌های برنامه

از آنجایی که دستگاه‌های تلفن همراه منابع حافظه و پردازشی کمتری دارند، یکی از مسائلی که برنامه نویسان اندروید با آن مواجه می‌شوند کنترل پویای منابع است. برای اینکه رابط کاربری مطلوب و روان داشته باشیم نیاز به استفاده‌ی بهینه از منابع داریم، لذا بایستی به thread مربوط به رابط کاربری اجازه دهیم که تنها وظایف به‌روزرسانی رابط کاربری را انجام دهد و عملیات‌های blocking مثل اتصال به شبکه و پایگاه داده را در لایه‌ی مربوط به خود انجام دهیم. برای این منظور سه لایه‌ی مجزا تعریف شد که ارتباط بین لایه‌ها از طریق رابط‌های interface انجام می‌شود.



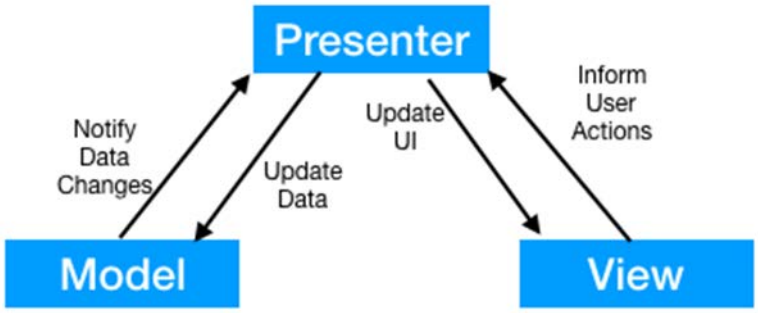
شکل (۳-۲) لایه‌های برنامه

داده از لایه‌ی شبکه وارد برنامه شده، در پایگاه داده ذخیره می‌گردد و سپس از آنجا به رابط کاربری برای به‌روزرسانی صفحات انتقال میابد. وظیفه‌ی هر لایه با لایه‌های زیرین و بالادستی کاملا ایزوله‌شده و ارتباط بین آنها به‌وسیله‌ی رویدادها، listenerها و callbackها انجام می‌شود.

۲-۲-۲- لایه‌ی Presentation

این لایه وظیفه‌ی کنترل و به‌روزرسانی اطلاعات قابل نمایش روی رابط کاربری را دارد. صفحات برنامه شامل Activity، Fragment، Custom view و Adapter برای کنترل لیست‌های برنامه در این لایه قرار میگیرند. هر صفحه‌ی قابل نمایش در برنامه با استفاده از الگوی طراحی (Design Pattern) MVP یا Model-View-Presenter پیاده سازی شد. با استفاده از MVP مسئله‌ی Soc (Separation-of-concerns) [7] به خوبی حل شده و وظایف لایه Presentation از Business logic جدا می‌شود.

در شکل زیر ساختار الگوی طراحی MVP ترسیم شده:



شکل (۴-۲) الگوی طراحی MVP

۱. Model: داده‌هایی که قرار است در رابط کاربری نمایش داده شوند در این کلاس قرار میگیرند. وظیفه‌ی این کلاس ذخیره سازی داده‌ها و اطلاع دادن به Presenter حین تغییر داده‌ها می‌باشد.

۲. Presenter: این کلاس رابط بین Model و View است و دستورالعمل‌های Business Logic در این کلاس قرار میگیرند.

۳. View: کنترل و به‌روزرسانی رابط کاربری توسط این کلاس صورت میپذیرد. به‌ازای هر رویدادی که توسط کاربر انجام می‌شود یک فراخوانی به Presenter انجام شده و در صورتی که عملیات blocking انجام شود نتیجه از طریق callback به‌اطلاع View میرسد.

۳-۲-۲- لایه‌ی Database

سیستم‌عامل اندروید از سیستم مدیریت پایگاه داده رابطه‌ای SQLite پشتیبانی می‌کند. برای این منظور سرویسی مجزا در سطح هسته گنجانده شده تا عملیات‌های مربوط به پایگاه داده را مدیریت کند. ارتباط بین اپلیکیشن با این سرویس از طریق کلاس SQLiteOpenHelper انجام می‌شود. این کلاس حاوی دو متد کلیدی onCreate() وonUpgrade() می‌باشد. متد اول حین اولین فراخوانی، برای ایجاد پایگاه داده مورد استفاده قرار میگیرد، لذا در این متد بایستی Schema و ساختار پایگاه داده از طریق اسکریپت‌های sql ایجاد گردد. متد دوم در صورتی فراخوانی می‌شود که نسخه ی جدیدی از اپلیکیشن روی دستگاه نصب نصب می‌گردد. در این متد نیز در صورت نیاز بایستی اسکریپت‌های مناسب جهت تغییر ساختار پایگاه داده استفاده شود.

تفاوت بین SQLite و دیگر سیستم‌های مدیریت پایگاه داده‌ی رابطه‌ای:

۱. SQLite نسخه‌ی سبک‌تری نسبت به پایگاه داده‌های دیگر است، لذا استفاده از آن در دستگاه‌های قابل حمل و توکار (Embedded) کارآمدتر است.

۲. در SQLite داده‌ها داخل یکسری فایل ذخیره می‌شوند در حالی که MySQL و SQL-Server سرویس محور هستند.

۳. به‌علت سبک بودن به راحتی بین حافظه داخلی و Ram جابجا می‌شود.

۴. SQLite سیستم مدیریت کاربر ندارد. (به‌علت تک کاربره بودن، نیازی به این سیستم نیست)

۵. SQLite از Stored Procedures پشتیبانی نمی‌کند.

۶. SQLite سیستم احراز هویت ندارد و کنترل دسترسی توسط سیستم Sandboxing اندروید انجام می‌شود.

پیاده سازی مستقیم لایه‌ی پایگاه داده با به‌کارگیری کلاس SQLiteOpenHelper به فراخوانی‌های سطح پایین زیادی نیاز دارد که هم خوانایی و نگه داری کد را کاهش میدهد و هم کدهای boilerplate ایجاد می‌کند. لذا از کتابخانه‌ی Room که توسط گوگل در نوامبر ۲۰۱۷ معرفی شد[8] ، استفاده گردید.

اجزای اصلی Room

این کتابخانه از سه موجودیت تشکیل شده:

۱. Entity: کلاسی است که نمایانگر یک جدول از پایگاه داده است.

```java
@Entity
public class User {
    @PrimaryKey
    private int uid;

    @ColumnInfo(name = "first_name")
    private String firstName;

    @ColumnInfo(name = "last_name")
    private String lastName;

    // Getters and setters are ignored for brevity,
    // but they're required for Room to work.
}
```

شکل (۵-۲) کلاس Entity در کتابخانه Room

۲. DAO: واسطه‌ای (interface) است که متدهای مورد نیاز برای کار با جداول در آن قرار میگیرد.

```
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    List<User> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND "
           + "last_name LIKE :last LIMIT 1")
    User findByName(String first, String last);

    @Insert
    void insertAll(User... users);

    @Delete
    void delete(User user);
}
```

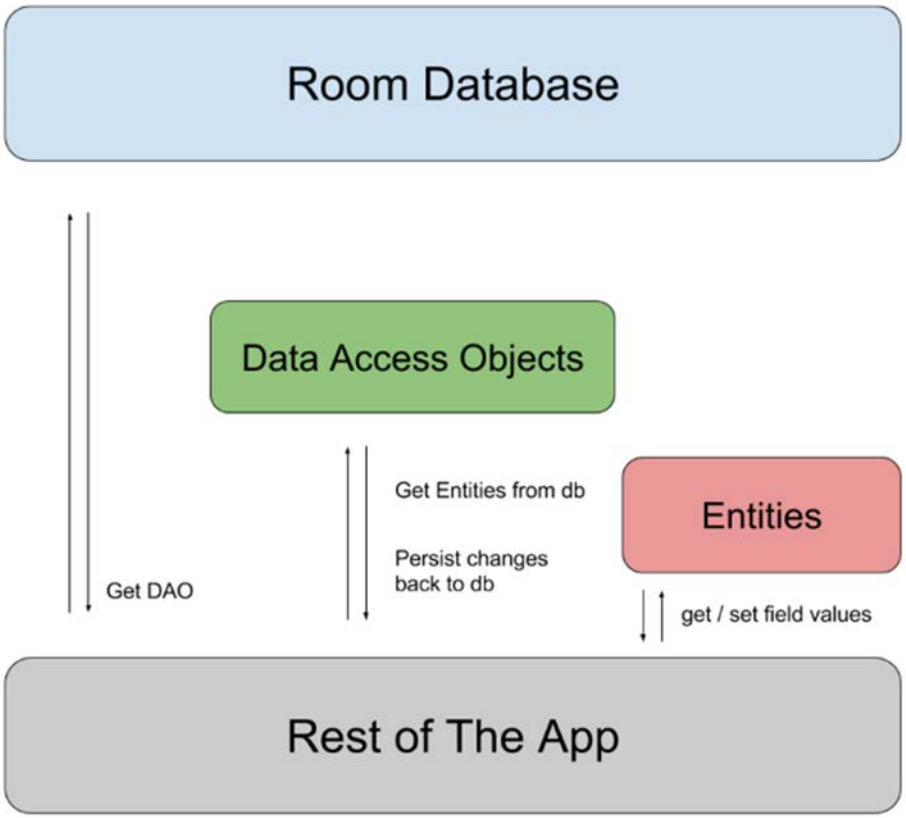
شکل (۶-۲) واسط DAO در کتابخانه Room

۳. Database: کلاسی abstract است که به‌عنوان رابط بین پایگاه داده و سرویس گیرنده عمل کرده و از کلاس RoomDatabase ارث بری می‌کند. این کلاس دربردارنده‌ی متدها و رابط‌های مورد نیاز برای دسترسی و دستکاری پایگاه داده است.

```
@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao();
}
```

شکل (۷-۲) کلاس Database در کتابخانه Room

ارتباط بین این موجودیت‌ها در شکل زیر نمایش داده‌شده:



شکل (۸-۲) ارتباط برنامه با کتابخانه Room

سرویس گیرنده از طریق Room Database واسطه‌ی DAO را دریافت کرده و با استفاده از متدهایی که در این واسطه تعریف شده عملیات‌های پایگاه داده‌ای خود را انجام داده و نتیجه را در قالب یک Entity دریافت می‌کند.

۴-۲-۲- لایه‌ی Network Socket

در این لایه کنترل و همگام سازی اتصالات شبکه انجام می‌شود. ارتباط بین بقیه‌ی لایه‌ها از طریق واسطهای interface صورت گرفته و مسائل همزمانی و همگام‌سازی نیز از طریق کلاس‌های Thread در جاوا و AppExecutor در اندروید انجام می‌شود.

انواع thread در AppExecutor:

۱. Main-Thread: جریان اصلی برنامه (کنترل رابط کاربری) در این نخ انجام می‌شود. این نخ توسط سیستم‌عامل ایجاد و پایان میابد و از طریق متدی استاتیک، اشاره گر آن قابل دسترسی است.

از این نخ برای فراخوانی callbackها و listenerهایی که بایستی نتیجه‌ی آنها روی رابط کاربری اعمال گردند، استفاده میکنیم. عملیات‌های blocking [9] روی این نخ اپلیکیشن را به‌حالت اجرای ANR (Application Not Responding) میبرد و پیغامی به کاربر نمایش میدهد که اپلیکیشن مذکور را ببندد یا منتظر دریافت پاسخ بماند. لذا برای رفع این مشکل از دو نخ زیر استفاده میکنیم

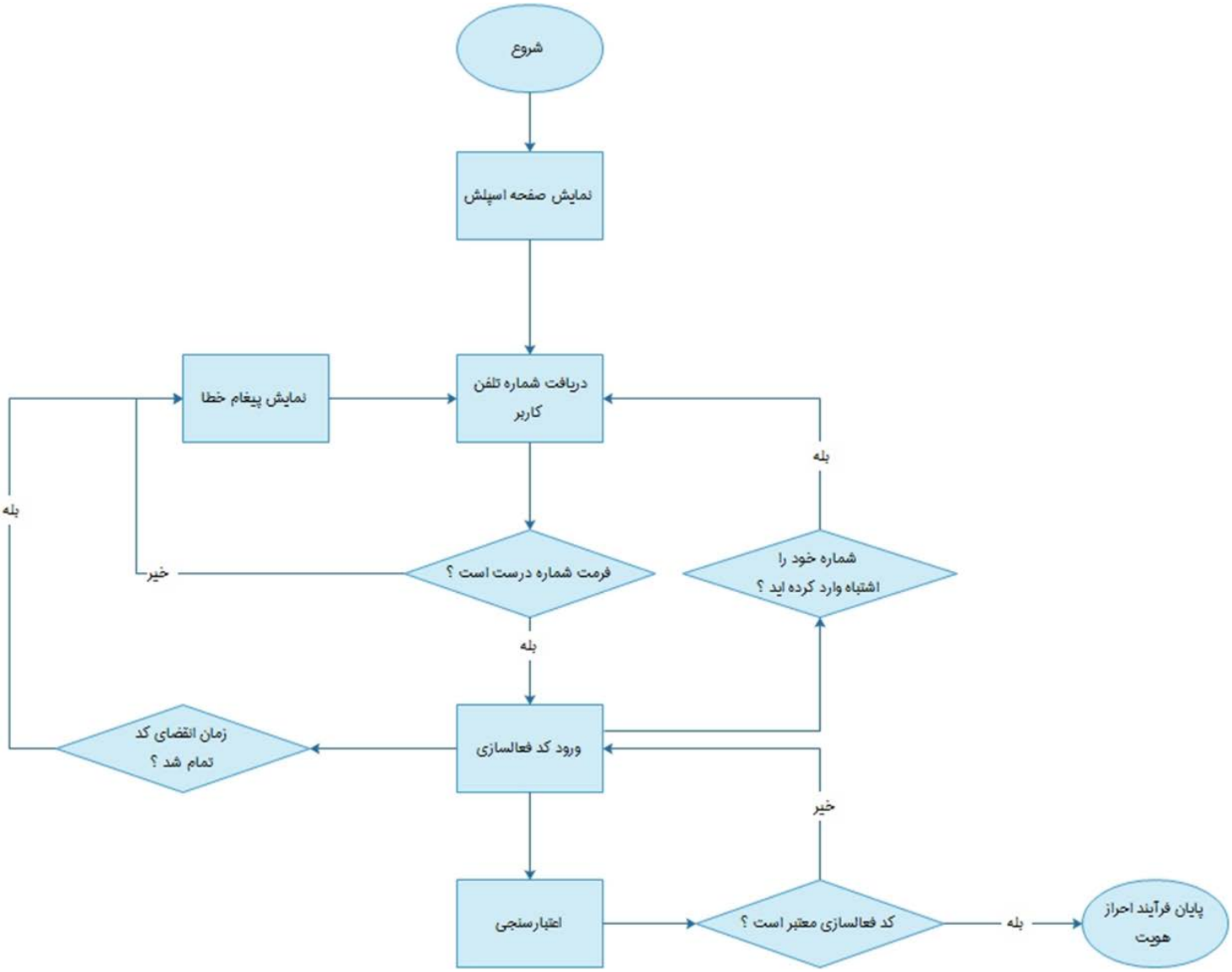
۲. Network-IO: اجرای دستورالعمل‌های مربوط به شبکه در این نخ انجام می‌شود، از آنجایی که اتصالات شبکه قابلیت همزمانی بالایی دارند، به‌منظور بهبود سرعت ارتباطات ۳ عدد نخ تعریف شده.

۳. Disk-IO: اجرای دستورالعمل‌های مربوط به پایگاه داده در این نخ انجام می‌شود. در صورت ارتباط همزمان با پایگاه داده، ریسک تداخل و ناسازگاری داده‌ها بالا رفته و اصل ACID[10] نقض می‌شود، لذا یک نخ برای این ارتباط لحاظ شده که در صورت مشغول بودن این نخ سرویس گیرنده در صف انتظار قرار گیرد.

۳-۲- ثبت‌نام

با اولین ورود به اپلیکیشن کاربر بایستی شماره تلفن خود را وارد کرده و با دریافت کد فعالسازی که از طریق پیامک به دستگاه کاربر ارسال می‌شود، اقدام به ثبت نام کند. شماره تلفن کاربر به‌عنوان شناسه‌ی یکتا در تمام تراکنش‌ها و ارتباطات شبکه مورد استفاده قرار میگیرد.

فرآیند ثبت نام کاربر در فلوچارت زیر نمایش داده‌شده:



شکل (۹-۲) فلوچارت ثبت‌نام در برنامه

فرآیند ثبت نام:

۱. صفحه‌ی ورود شماره

در این صفحه کلاینت به socket متصل می‌شود و بعد از دریافت callback از اتصال آن، شماره‌ای که کاربر وارد کرده را به سرور ارسال می‌کند (سرور نیز یک پیامک حاوی رمز زماندار و یکبار مصرف به شماره‌ی مذکور ارسال می‌کند، اگر شماره صحیح باشد و پیامک ارسال گردد نتیجه «ok» برمی‌گردد و اگر شماره نادرست بوده یا سرور ارسال پیامک پاسخ نمیدهد، نتیجه‌ی «failed» برمی‌گردد)

۲. صفحه‌ی تایید شماره با کد فعالسازی

زمان انقضای کد از سمت سرور ۱۲۰ ثانیه به‌علاوه ۲ ثانیه offset تاخیر بافرهای شبکه لحاظ شده

اگر کاربر پیامک را در این مدت زمان دریافت کرده و موفق به ورود کد به برنامه شود، با زدن دکمه‌ی اعتبارسنجی، کد فعالسازی به سرور ارسال می‌شود.

در سرور نیز این کد با کدهایی که در لیست OTP[1] قرارداده شده‌اند مطابقت شده و در صورت صحیح بودن، آن را از لیست خارج کرده و پیام تایید به کلاینت ارسال می‌کند و در غیر اینصورت پیام خطا برمیگرداند.

در صورت صحیح بودن کد فعالسازی، شماره‌ی ثبت‌شده به سرور ارسال شده و در صورتی که ثبت نام موفقیت آمیز باشد یا کاربر از قبل ثبت نام کرده باشد، نتیجه «ok» برمی‌گردد و برنامه به صفحه‌ی خانه هدایت می‌شود.

۴-۲- مخاطبین

در این بخش مخاطبین دستگاه با سرور همگام شده و اطلاعات پروفایل آنها در صفحه‌ی لیست مخاطبین، پروفایل و مکالمه نمایش داده می‌شود. برای دسترسی به مخاطبین نیاز به کسب مجوز در حین اجرا[2]می‌باشد. لذا در فرآیند ثبت نام این مجوز کسب می‌گردد.

حین ورود به صفحه‌ی مخاطبین در صورتی که کاربر اتصال اینترنت نداشته باشد، مخاطبین از پایگاه داده خوانده شده و در لیست بارگذاری می‌شوند. در صورتی که کاربر به اینترنت متصل باشد ابتدا لیست مخاطبین از پایگاه داده بارگذاری شده و سپس با سرور همگام می‌شوند و در صورت دریافت اطلاعات جدید از سرور لیست به‌روز می‌شود.

اطلاعاتی که از یک مخاطب برای برنامه نیاز است شامل موارد زیر می‌گردد:

۱. نام و نام خانوادگی
۲. شماره تلفن

[1] One-Time-Password

[2] Permission on runtime

۳. آخرین بازدید

۴. تصویر پروفایل

اطلاعاتی از مخاطب که به‌روز می‌شوند:

۱. آخرین بازدید

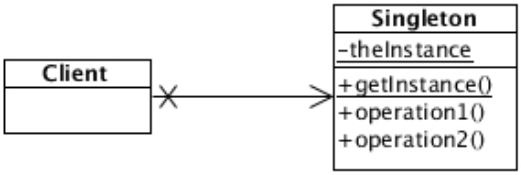
۲. تصویر پروفایل

زمانی که این اطلاعات تغییر می‌کند، سرور یک پیام به‌روزرسانی به تمامی کلاینت‌هایی که این مخاطب را در گوشی خود دارند و هم اکنون به سرور متصل هستند، ارسال می‌کند. اگر کلاینتی به سرور متصل نباشد در اولین اتصال لیست مخاطبین خود را به‌روز می‌کند، لذا اطمینان حاصل می‌شود که تمامی کاربران اطلاعات به‌روزی از وضعیت مخاطبین خود دارند.

۵-۲- سوکت

برای ارتباط با سرور از دو سوکت مجزا، یکی برای ارسال پیام‌ها و دستورات، و دیگری برای آپلود تصویر پروفایل به‌کار گرفته شد. علت استفاده از سوکت‌های مجزا بلاک نشدن سوکت پیام‌ها توسط آپلود بود. یک کلاس مجزا به‌نام ConnectionHandler برای کنترل اتصال به شبکه طراحی گردید. این کلاس نقطه‌ی شروع اتصال به سرور می‌باشد و از آنجایی که بایستی یک نمونه از آن در طول عمر برنامه وجود داشته باشد، با استفاده از الگوی طراحی Singleton پیاده سازی شد. این کلاس در یک حلقه بی نهایت قرار دارد و با استفاده از یک تایمر در فواصل زمانی ۱۰ ثانیه اقدام به بررسی وضعیت اتصال به سوکت می‌کند.

الگوی طراحی Singleton را در شکل زیر مشاهده میکنید:



شکل (۱۰-۲) الگوی طراحی Singleton

این کلاس یک سازنده‌ی private برای جلوگیری از ایجاد شی دارد و یک شی از جنس خودش به‌صورت استاتیک نگه داری می‌کند که از طریق متد ()getInstance قابل دستیابی است. در ابتدا که این شی null است، مقداردهی اولیه انجام شده و شی ساخته می‌شود. دفعات بعدی که متد ()getInstance فراخوانی می‌گردد مقدار قبلی برمی‌گردد.

۲-۵-۲- پروتکل ارتباط سوکت

قالب بسته‌های ارسالی و دریافتی بین سرور و کلاینت به این صورت است:

HeaderKey1: Value1 #
HeaderKey2: Value2 #
HeaderKey3: Value3 #
…
$
data data data …

که در اینجا علامت # به‌عنوان جدا کننده ی headerها، علامت $ جداکننده‌ی قسمت header و body است

به‌عنوان نمونه این بسته برای ثبت نام کاربر به سرور ارسال می‌شود:

Method: register #
Content-Length: 11#
$
{ “phone”: “+989302871049” }

لیست پیام‌های ارسالی و دریافتی از سوکت

۱. heartbeat: برای زنده نگه داشتن اتصال سوکت این پیام هر ۱۰ ثانیه یکبار به سرور ارسال می‌گردد. (پیام هیچ محتوایی ندارد و صرفا header خالی ارسال می‌شود)

۲. otp_request: درخواست کد فعالسازی جهت ثبت نام کاربر

۳. otp_verify: احراز هویت کاربر با کد یکبارمصرفی که از طریق پیامک به کاربر ارسال می‌شود.

۴. register: ثبت نام کاربر

۵. go_online: آنلاین شدن

۶. go_offline: آفلاین شدن

۷. upload_photo: آپلود تصویر پروفایل

۸. change_profile_pic: تغییر تصویر پروفایل کاربر

۹. change_last_seen: تغییر آخرین بازدید کاربر

۱۰. text_message: ارسال پیام متنی

۱۱. text_message_ack: تایید ارسال پیام متنی

۱۲. deliver_message: دریافت پیام متنی

۱۳. deliver_message_ack: تایید دریافت پیام متنی

۱۴. seen_message: مشاهده‌ی پیام

۱۵. seen_message_ack: تایید مشاهده‌ی پیام

۱۶. deliver_seen: دریافت مشاهده‌ی پیام توسط مخاطب

۱۷. :deliver_seen_ack تایید دریافت مشاهده‌ی پیام توسط مخاطب

۱۸. sync_profile: همگام سازی پروفایل (تصویر و آخرین بازدید) در صفحه پروفایل

۱۹. sync_contacts: همگام سازی مخاطبین در پایگاه داده

۲۰. sync_contacts_status: همگام سازی وضعیت مخاطبین (آخرین بازدید) در صفحه مخاطبین

۲۱. sync_deliver_seen_messages: همگام سازی مشاهده‌ی پیام

۲۲. sync_deliver_text_messages: همگام سازی پیام‌های متنی

۳-۵-۲- فرآیند آنلاین و آفلاین شدن کاربر

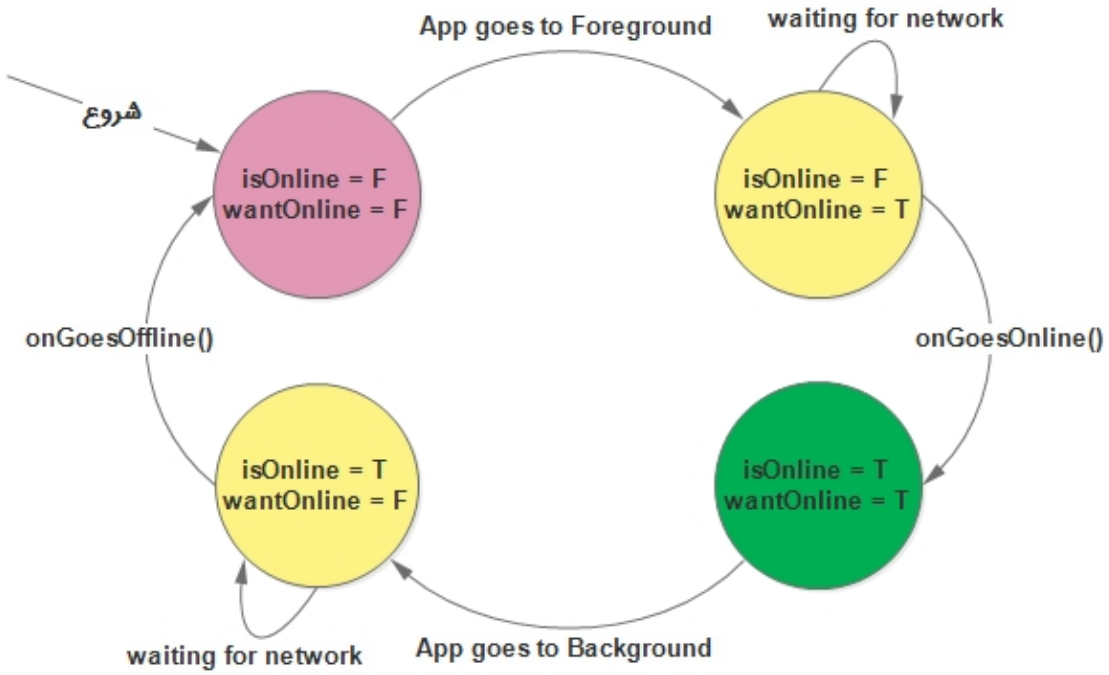
کاربر زمانی آنلاین است که با اپلیکیشن در حال تعامل باشد یعنی به‌اصطلاح Activity فعلی برنامه در حالت Foreground باشد و زمانی آفلاین می‌شود که یکی از متدهای ()onStop، ()onDestroy از Activity فعلی فراخوانی شود (یعنی برنامه به پس زمینه برود به‌طوری که دیگر قابل مشاهده نباشد یا کاربر به برنامه‌ی دیگری جابجا شود).

آنلاین بودن کاربر صرفا برای نمایش وضعیت وی در قسمت آخرین بازدید است و به process پس زمینه که دائما در حال به‌روزرسانی و همگام سازی اطلاعات است ربطی ندارد یعنی کاملا واضح است که مثلا کاربر آفلاین بوده ولی فرآیندهای پس زمینه فعال باشند.

از آنجایی که در سیستم‌عامل اندروید صفحات یا Activityها طول عمر خود را دارند لذا گذر از یک Activity به Activity دیگر با یکسری فراخوانی‌ها همراه است که رویداد مذکور را به اپلکیشن علامت دهد. با گوش دادن به این رویدادها، میتوانیم بدانیم که اپلیکیشن چه زمانی به‌حالت

Background می‌رود و چه زمان به Foreground، برای این منظور واسط ActivityLifecycleCallbacks را در کلاس AppLifecycleTracker پیاده سازی کرده و در کلاس ApplicationLoader (این کلاس اولین نقطه‌ی شروع اپلیکیشن است) این واسط را ثبت نام میکنیم. از طریق متدهای onActivityStarted و onActivityStopped میتوانیم تشخیص دهیم که اپلیکیشن در چه حالتی قرار دارد و سپس اقدام به ارسال پیام آنلاین و آفلاین کنیم.

برای کنترل وضعیت آنلاین یا آفلاین بودن کاربر دو متغیر سراسری Boolean به‌نام‌ها isOnline و wantOnline در کلاس ConnectionHandler ایجاد شد. تمامی حالت‌های ممکن این دو متغیر، وضعیت کاربر و عملیات‌های انجام‌شده در اثر رویدادهای اپلیکیشن در شکل زیر ترسیم شده:

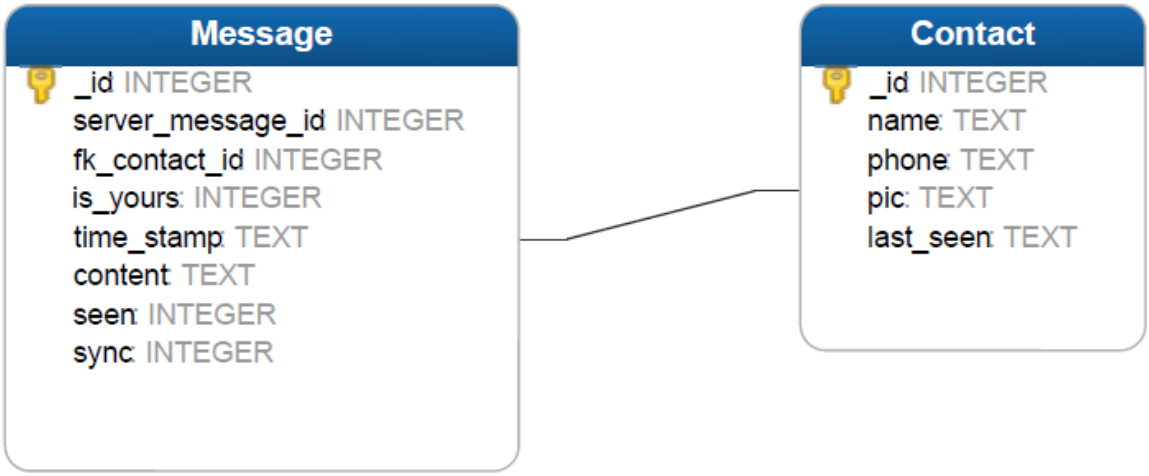


شکل (۱۱-۲) نمودار حالت آنلاین و آفلاین شدن کاربر

۶-۲- پایگاه داده

پایگاه داده از دو جدول «مخاطب» و «پیام» تشکیل شده که برای ذخیره‌ی محلی داده‌های سرور استفاده می‌شوند. برای اینکه دسترسی به پایگاه داده از یک نقطه باشد، کلاسی به‌نام DatabaseBroker ایجاد گردید و دو شی از جداول پایگاه داده به‌صورت متغیر سراسری در این کلاس تعریف شد. همچنین برای کنترل همزمانی و همگام سازی جریان دستورالعمل‌ها یک شی از کلاس AppExecutor نیز در این کلاس تعریف شد.

ساختار جداول پایگاه داده را در شکل زیر مشاهده میکنید:



شکل (۱۲-۲) نمودار ER پایگاه داده‌ی کلاینت

جدول Contact: ذخیره ی مخاطبینی از دستگاه که در اپلیکیشن ثبت نام کرده‌اند.

۱. _id: شناسه‌ی یکتای جدول
۲. name: نام مخاطب در دستگاه
۳. phone: شماره تلفن مخاطب
۴. pic: تصویر پروفایل مخاطب
۵. last_seen: آخرین بازدید مخاطب

جدول Message: ذخیره‌ی پیام‌ها برای دسترسی آفلاین

۱. _id: شناسه‌ی یکتای جدول
۲. server_message_id: ایجاد ارتباط با جدول پیام در سرور
۳. fk_contact_id: کلید خارجی با جدول مخاطب
۴. is_yours: ایجادکننده‌ی پیام (0 برای خود کاربر و 1 برای مخاطب کاربر)
۵. time_stamp: زمان ایجاد پیام
۶. content: محتوای پیام
۷. seen: وضعیت مشاهده‌ی پیام (0 برای مشاهده‌نشده و 1 برای مشاهده‌شده)

۸. sync: وضعیت همگام سازی پیام با سرور (0 برای همگام‌نشده و 1 برای همگام‌شده)

سه فیلد is_yours، seen، sync حالت‌های هشت‌گانه‌ای ایجاد می‌کنند که در هر کدام از این حالت‌ها رویداد خاصی اتفاق می‌افتد. تمامی وضعیت‌ها و رویدادهای متناظر در جدول زیر لیست شده:

جدول (۱-۲) حالت‌های مختلف فیلدهای پرچم، در جدول Message پایگاه داده‌ی کلاینت

نام حالت	Sync	Seen	Is_yours	نیاز به همگام‌سازی	رویداد مورد نیاز
پیام دریافت شده ولی همگام نشده	0	0	0	دارد	deliver_ack
پیام دریافت شده و همگام شده	1	0	0	ندارد	-
پیام مخاطب مشاهده شده ولی همگام نشده	0	1	0	دارد	seen_message
پیام مخاطب مشاهده شده و همگام شده	1	1	0	ندارد	-
پیام ایجاد شده ولی همگام نشده	0	0	1	دارد	text_message
پیام ایجاد شده و همگام شده	1	0	1	ندارد	-
دستور مشاهده پیام دریافت شده ولی همگام نشده	0	1	1	دارد	deliver_seen_ack
دستور مشاهده پیام دریافت شده و همگام شده	1	1	1	ندارد	-

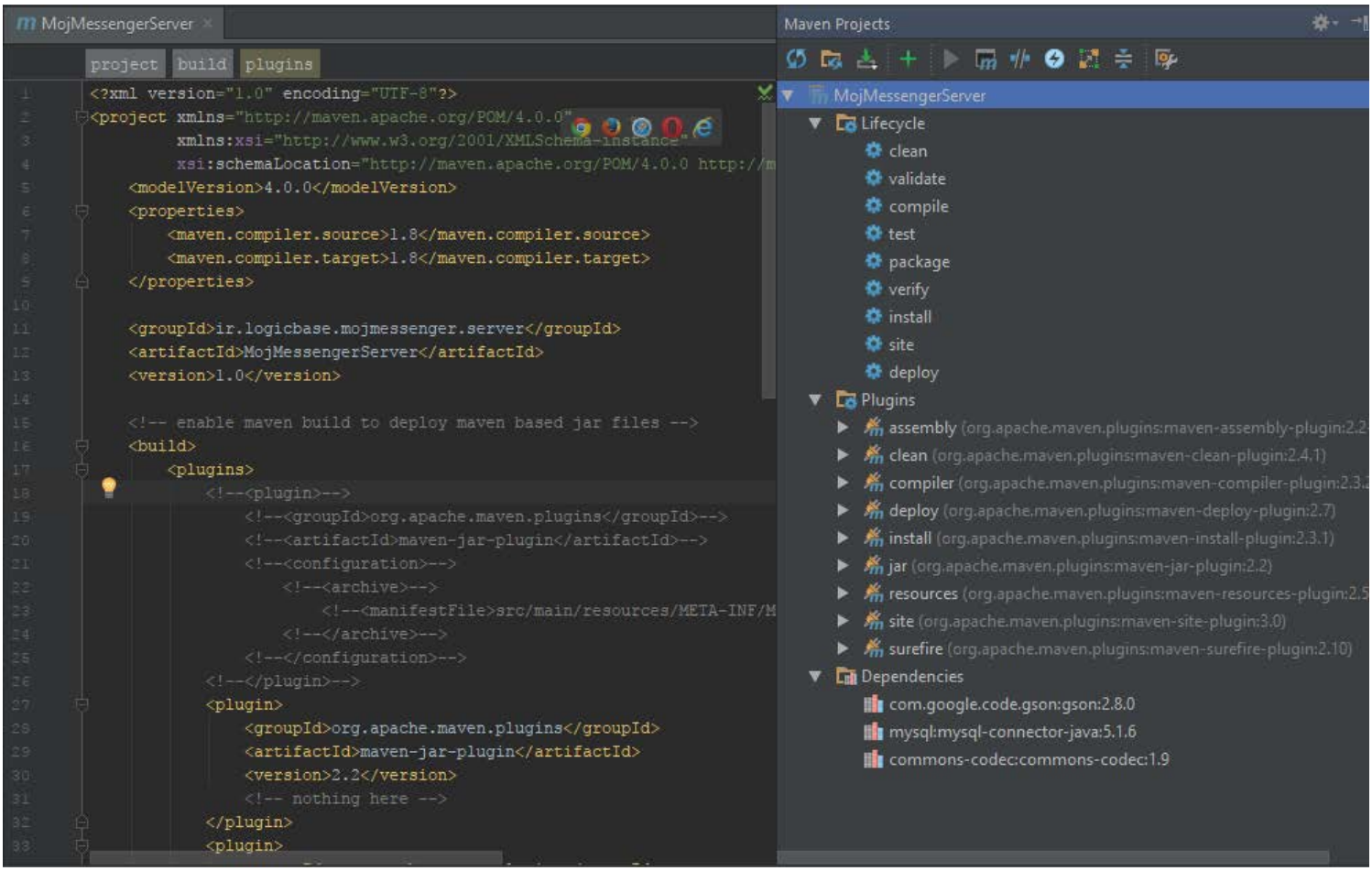
به‌منظور همگام سازی پیام‌ها با سرور، کلاسی به‌نام SyncDatabase ایجاد گردید. این کلاس هنگام اتصال اولیه سوکت و ورود به صفحه‌ی مکالمه اقدام به همگام سازی پیام‌ها و مخاطبین با سرور کرده و رکوردهای متناظر را به‌روزرسانی می‌کند.

فصل ۳:

عملکرد برنامه سمت سرور

۱-۳- کلیات

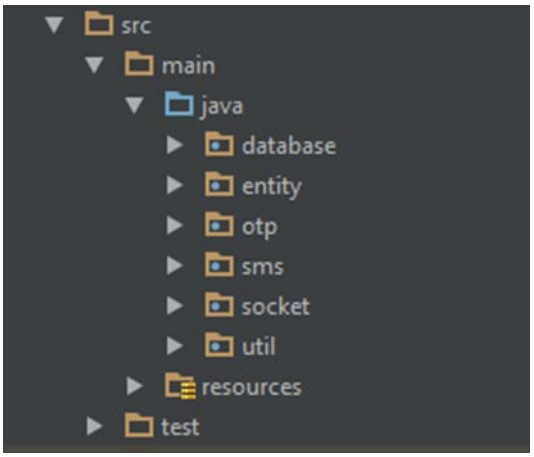
سمت سرور پروژه به‌زبان جاوا نوشته شده[11] و در محیط توسعه‌ی Intellij IDEA طراحی و پیاده‌سازی شده‌است. با استفاده از ابزار Maven وابستگی‌ها و پیکربندی‌های build و deploy پروژه انجام شد. این ابزار برای اتوماتیک‌کردن فرآیند تولید نرم افزار مورد استفاده قرار میگیرد و هدف آن سهولت در توسعه می‌باشد.



شکل (۱-۳) ابزار Maven و چرخه‌ی تولید نرم‌افزار توسط آن

۲-۱-۳- پکیج‌بندی

پروژه از یکسری پکیج‌های جاوا تشکیل شده که وظایف هر پکیج در زیر تشریح داده‌شده:



شکل (۲-۳) پکیج‌های برنامه سمت سرور

۱. database: مدیریت پایگاه داده

۲. entity: مدل داده برای تبادل بین واسط‌ها

۳. otp: تولید و اعتبارسنجی رمز یکبار مصرف ثبت نام

۴. sms: ارتباط با سرور پیامکی

۵. socket: مدخل ارتباط با کلاینت

۶. util: ابزارهای کاربردی

۳-۱-۳- رمز یکبار مصرف OTP

یکی از روش‌های جلوگیری از حدس زدن کلمات عبور ضعیف و نامناسب، استفاده از رمزهای یک بار مصرف (One Time Password) می باشد. برای تولید کلمات عبور یک بار مصرف از ابزاری به‌نام توکن OTP، استفاده می‌گردد.رمز یکبار مصرف برای ایمن سازی دسترسی کاربران به سیستم‌های الکترونیکی است،که در آن از قابلیت‌های رمز نگاری برای تولید رمز تصادفی یک بار مصرف استفاده می‌شود. آنچه که از کلمه رمز در ذهن اغلب افراد تداعی می‌شود، کلمه‌ی رمز ایستا می باشد که مقداری است ثابت و می بایست به خاطره سپرده شود. در مقابل رمز ثابت، رمز یک بار مصرف یا OTP قراردارد که به‌معنای کلمه رمزی است که فقط و فقط یکبار می‌تواند مورد استفاده قرار گیرد. این رمز یکبار مصرف با استفاده از الگوریتم‌های رمزنگاری متقارن و وابسته به زمان (Time-Based) ساخته شده و از پیامک به کاربر ارسال می‌شود. کاربر بعد از دریافت پیامک رمز عبور را در اپلیکیشن

وارد کرده و به سرور ارسال می‌کند. در سرور نیز رمز دریافتی با الگوریتم متناظر اعتبارسنجی شده و نتیجه به کاربر اعلام می‌گردد.

۴-۱-۳- سامانه‌ی پیامکی

ارتباط با سامانه ی پیامکی از طریق پروتکل HTTP انجام شد. به این صورت که بعد از دریافت درخواست رمز یکبارمصرف از سمت کاربر، یک HTTP Request به سامانه ارسال می‌شود و در آنجا پیامک به کاربر ارسال می‌گردد. نتیجه‌ی ارسال پیامک از سامانه دریافت‌شده و از طریق ارتباط سوکت به‌اطلاع کاربر می‌رسد.

ارتباط با سامانه‌ی پیامکی در قالب زیر انجام می‌شود:

ساختار URL:

https://rest.payamak-panel.com/api/SendSMS/SendSMS

متد Post,Get ؛ این متد برای ارسال پیامک به حداکثر ۱۰۰ گیرنده در هر بار فراخوانی استفاده می شود.

پارامتر های ورودی			
نام پارامتر	نوع پارامتر		توضیحات
UserName	String	اجباری	نام کاربری مربوط به حساب شما در سامانه
PassWord	String	اجباری	کلمه عبور مربوط به حساب شما در سامانه
To	String	اجباری	شماره گیرنده،جهت ارسال بیش از یک شماره می توانید با کاراکتر «,» جدا نمایید.
From	String	اجباری	شماره اختصاصی فرستنده
Text	String	اجباری	متن پیامک
IsFlash	Bool	اختیاری	تعیین می کند آیا پیامک بصورت فلش ارسال گردد یا خیر

شکل (۳-۳) ساختار وب سرویس سامانه پیامکی

نتیجه‌ی ارسال پیامک نیز به این صورت می‌باشد:

مقدار بازگشتی			
نام پارامتر	نوع پارامتر	توضیحات	نمونه بازگشتی
Value	String	اگر ارسال موفق باشد در این پارامتر RecID ها قرار داده می شود و برای ارسال به بیش از یک گیرنده با کاراکتر «،» جدا می شود و اگر ارسال ناموفق باشد مقدار ۱۱ برگشت داده می شود. یک عدد: recId ارسال ۰: نام کاربری یا رمز عبور اشتباه می باشد. ۱: درخواست با موفقیت انجام شد. ۲: اعتبار کافی نمی باشد. ۳: محدودیت در ارسال روزانه ۴: محدودیت در حجم ارسال ۵: شماره فرستنده معتبر نمی باشد. ۶: سامانه در حال بروزرسانی می باشد. ۷: متن حاوی کلمه فیلتر شده می باشد.	{ “Value”: “recID,...”, “RetStatus”: 1, “StrRetStatus”: “Ok” }
RetStatus	Int	اگر ارسال موفق باشد مقدار ۱ دریافت می کنید.	
StrRetStatus	String	اگر ارسال موفق باشد مقدار OK دریافت می کنید.	

شکل (۴-۳) ساختار مقدار بازگشتی وب سرویس سامانه پیامکی

۲-۳- سوکت

نقاط اتصال کلاینت با سرور از دو پورت شبکه انجام شد. یکی پورت ۶۷۰۰ برای انتقال پیام‌ها و دستورات، دیگری پورت ۶۷۰۱ برای آپلود تصاویر پروفایل مورد استفاده قرار گرفت. این جداسازی از انتظار بی مورد کاربر برای ارسال پیام‌ها جلوگیری به‌عمل می‌آورد و کارایی اپلیکیشن را بهبود میبخشد.

کلاس ServerSocketBroker: درگاه اتصال کاربر به سرور و نقطه‌ی شروع برنامه می‌باشد. با شروع برنامه، سوکت سرور راه‌اندازی شده و در یک حلقه‌ی بی‌نهایت در انتظار اتصال کلاینت می‌ماند. به‌محض اتصال کلاینت، شی جدیدی (ClientTask) با Thread جداگانه ایجاد می‌کند تا ادامه‌ی عملیات‌های کاربر در آن نخ انجام گیرد. زمان TimeOut برای اتصال کلاینت نیز ۳۰ ثانیه در نظر گرفته شد تا از قطع اتصال باخبر شویم. هر ۱۰ ثانیه یکبار کلاینت پیامی به‌نام HeartBeat برای اطلاع به سرور از ادامه دار بودن اتصال، ارسال می‌کند.

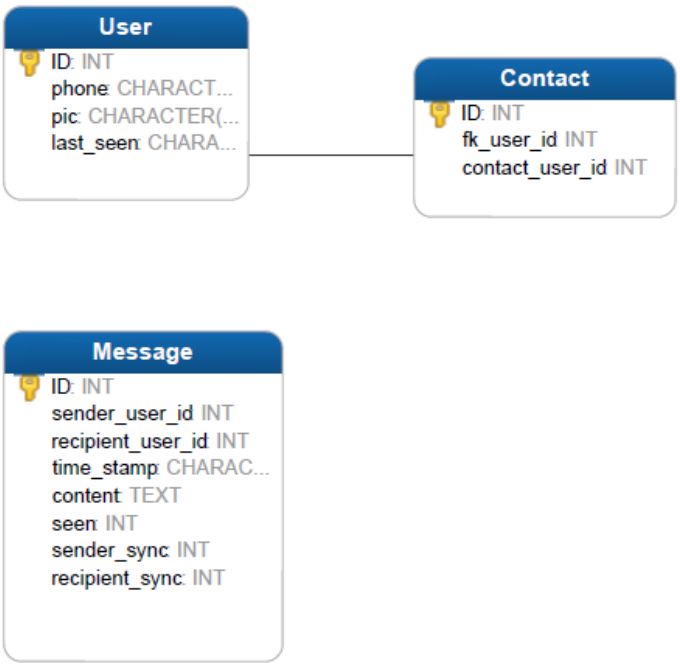
کلاس ClientTask: با دریافت سوکت کلاینت، این کلاس در انتظار دریافت پیام‌ها از درگاه ورودی خود میماند. با دریافت پیام، محتویات آن با استفاده از کلاس MessageHelper تفسیر شده و کنترل

جریان برنامه به کلاس ActionCenter انتقال میابد. در این کلاس بر اساس محتوای پیام، عملیات مناسب انجام شده و نتیجه به کلاس ClientTask برمی‌گردد. در صورت نیاز به پاسخ کلاینت، کلاسی به‌نام MessageSender در نظر گرفته شده که دارای صف انتظار پیام‌ها بوده و Thread جداگانه‌ای برای ارسال پیام دارد.

کلاس UploadServerBroker: درگاه آپلود تصاویر پروفایل می‌باشد. این کلاس حلقه‌ای بی‌نهایت دارد که در انتظار اتصال کلاینت قرار گرفته، به‌محض دریافت اتصال، کلاینت را به کلاس UploadServerTask هدایت می‌کند و در آنجا تصویر آپلودشده را دریافت در سیستم فایلی ذخیره می‌کند. پاسخ آپلود انجام‌شده از طریق قرار گیری پیام متناسب در صف انتظار شی ClientTask مربوط به کاربر آپلود کننده قرار میگیرد.

۳-۳- پایگاه‌داده

برای ذخیره‌ی اطلاعات سیستم مدیریت پایگاه داده MySQL انتخاب گردید. ارتباط با این سیستم از طریق یک راه‌انداز (Driver) انجام می‌شود که روی پورت ۳۳۰۶ داده‌ها را ارسال می‌کند. جداول مورد نیاز برای ذخیره سازی اطلاعات به‌شرح زیر طراحی گردید:



شکل (۵-۳) نمودار ER پایگاه داده سرور

جدول User: ذخیره کاربرانی که در اپلیکیشن ثبت نام کرده اند.

۱. ID: شناسه‌ی یکتای کاربر
۲. phone: شماره تلفن همراه کاربر
۳. pic: تصویر پروفایل کاربر
۴. last_seen: آخرین بازدید کاربر

جدول Contact: ذخیره‌ی مخاطبین کاربران

۱. ID: شناسه‌ی یکتای مخاطب
۲. fk_user_id: کلید خارجی با جدول کاربر
۳. contact_user_id: شناسه‌ی مخاطب کاربر

جدول Message: ذخیره‌ی پیام‌های کاربران

۱. ID: شناسه‌ی یکتای پیام
۲. sender_user_id: شناسه‌ی فرستنده
۳. recipient_user_id: شناسه‌ی گیرنده
۴. time_stamp: زمان ارسال پیام
۵. content: محتوای پیام
۶. seen: وضعیت مشاهده‌ی پیام (0 برای مشاهده‌نشده و 1 برای مشاهده‌شده)
۷. sender_sync: وضعیت همگام سازی پیام با فرستنده (0 برای همگام‌شده و 1 برای همگام‌نشده)
۸. recipient_sync: وضعیت همگام سازی پیام با گیرنده (0 برای همگام‌شده و 1 برای همگام‌نشده)

سه فیلد seen، sender_sync و recipient_sync حالت‌های هشت‌گانه‌ای ایجاد می‌کنند که در هر کدام از این حالت‌ها رویداد خاصی اتفاق می‌افتد.

جدول (۱-۳) حالت‌های مختلف فیلدهای پرچم، در جدول Message پایگاه داده‌ی سرور

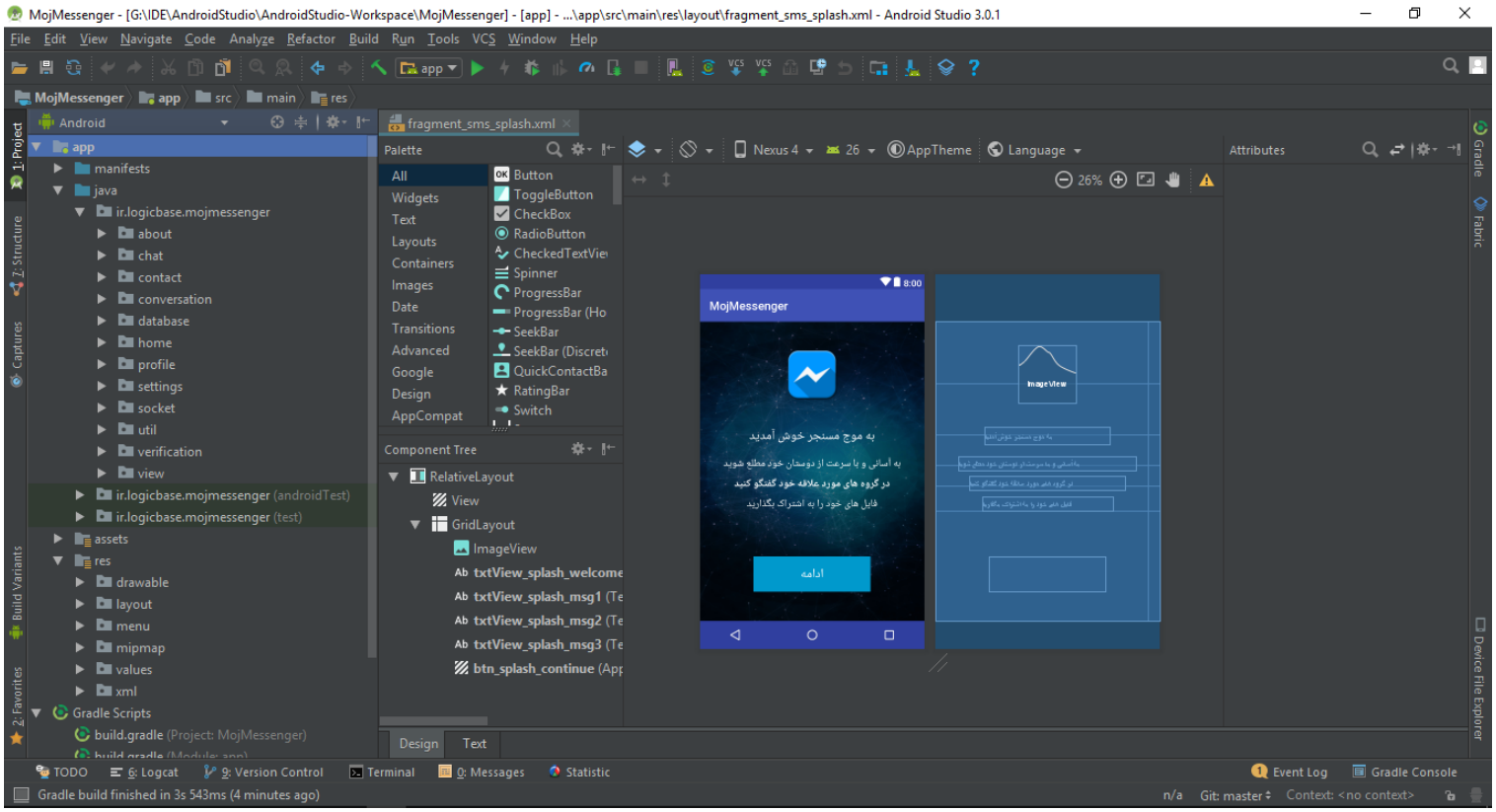
نام حالت	seen	recipient_sync	sender_sync	نیاز به همگام‌سازی	رویداد مورد نیاز
پیام فرستنده تایید نشده و به دست گیرنده نرسیده	0	0	0	دارد	text_message_ack deliver_message
ممکن نیست	1	0	0	-	-
پیام فرستنده تایید نشده	0	1	0	دارد	text_message_ack
پیام مشاهده شده ولی فرستنده اطلاع ندارد	1	1	0	دارد	deliver_seen
پیام به دست گیرنده نرسیده	0	0	1	دارد	deliver_message
ممکن نیست	1	0	1	-	-
تایید مشاهده پیام به گیرنده نرسیده	0	1	1	دارد	seen_message_ack
پیام به دست گیرنده رسیده و مشاهده شده و همگام شده	1	1	1	ندارد	-

فصل ۴:

امکانات، ابزارها و اسکرین شات

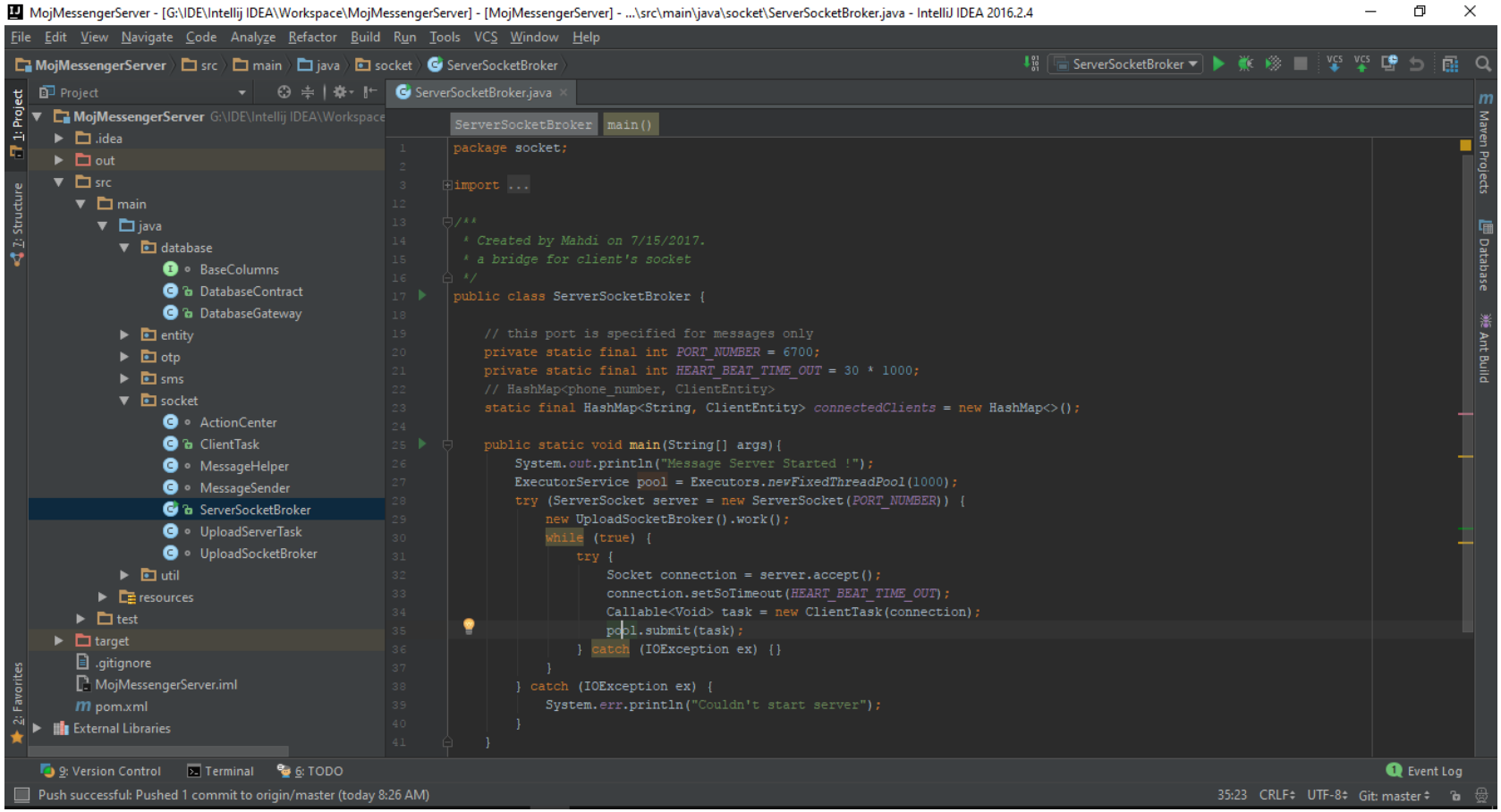
۴-۱- ابزارهای استفاده‌شده در پروژه

۴-۱-۱- Android Studio: توسعه‌ی اپلیکیشن سمت اندروید



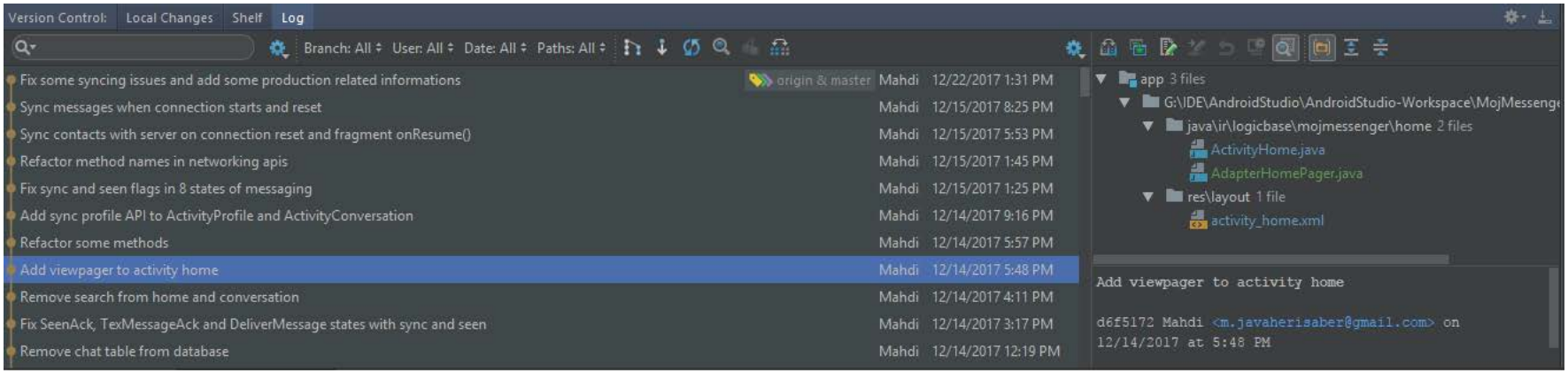
شکل (۴-۱) Android Studio IDE

۲-۱-۴- Intellij IDEA: توسعه‌ی اپلیکیشن سمت سرور

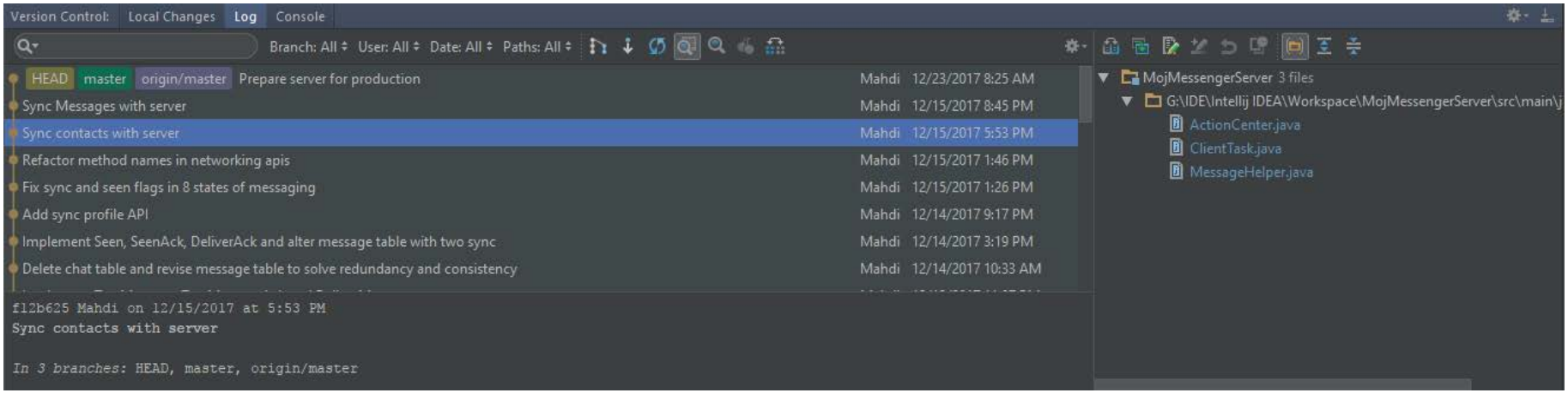


شکل (۲-۴) Intellij IDEA

۳-۱-۴- Git: سیستم مدیریت نسخه برای نگهداری سابقه‌ی تغییرات

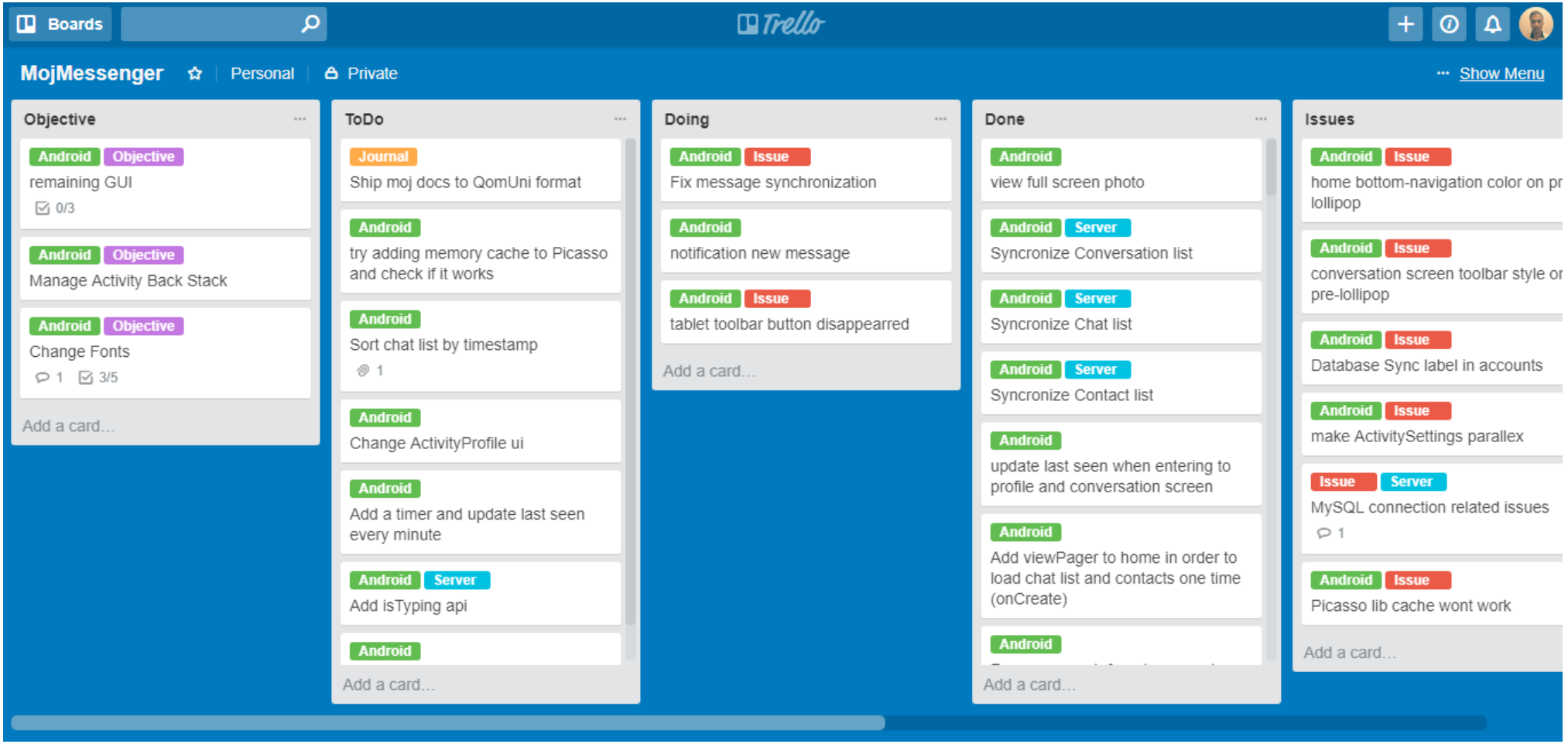


شکل (۴-۳) Android Studio Git panel



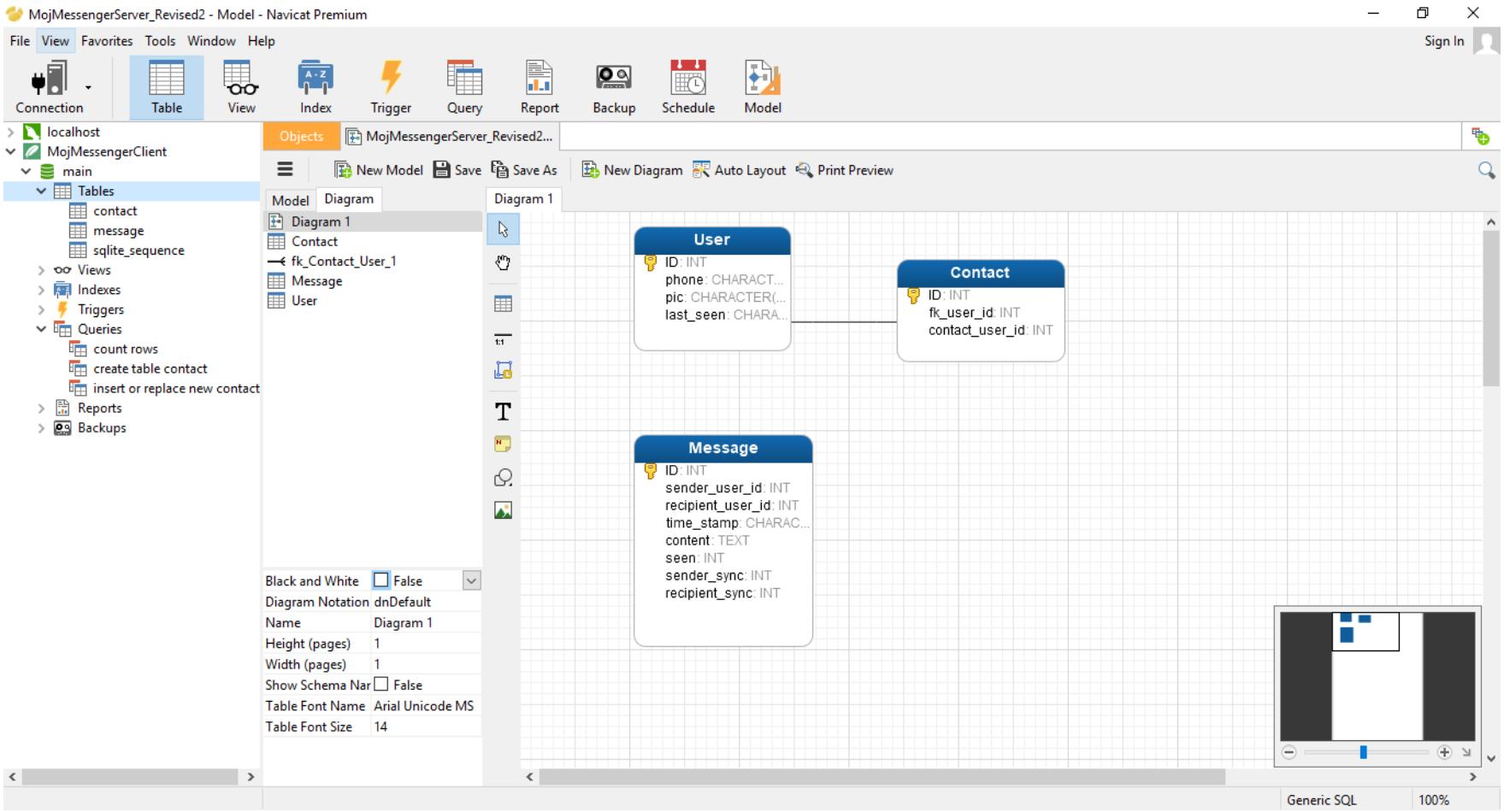
شکل (۴-۴) Intellij Git panel

۴-۱-۴- Trello: سیستم مدیریت پروژه برای تعریف ریز کارها (Trello.com)



شکل (۴-۵) Trello

۴-۱-۵- Navicat: سیستم مدیریت پایگاه داده

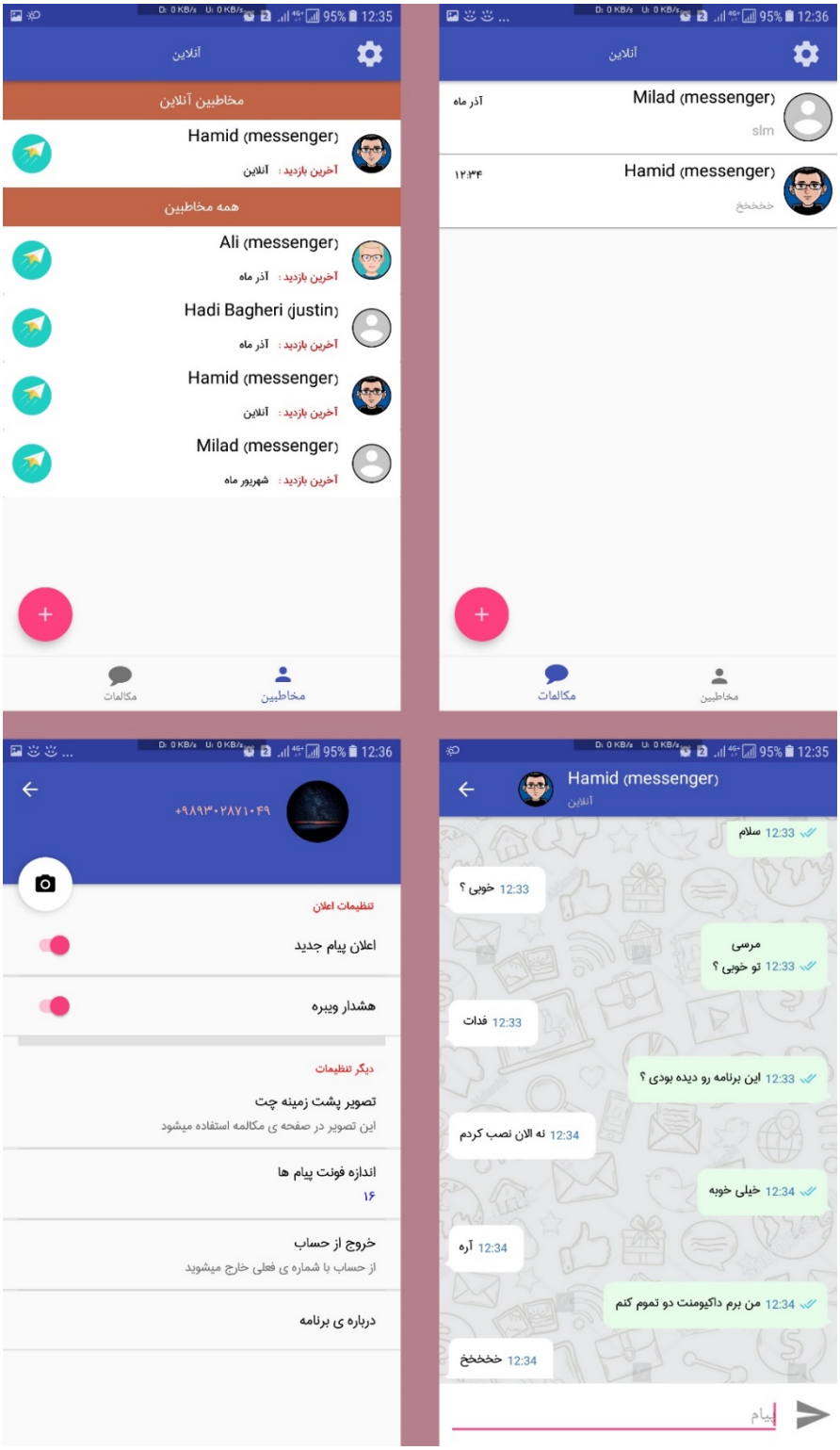


شکل (۴-۶) Navicat

۲-۴- امکانات و قابلیت‌های اپلیکیشن

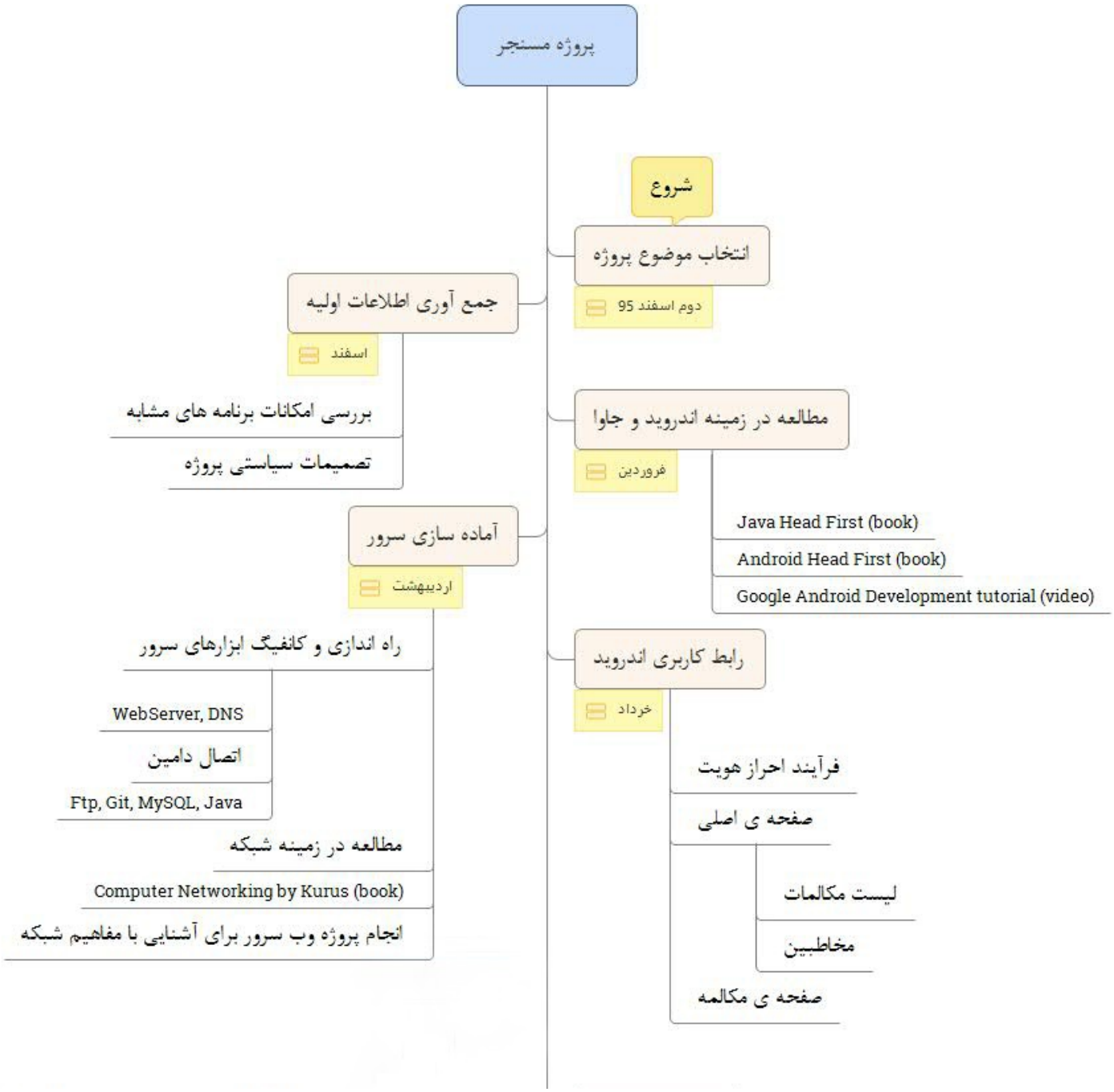
۱. ثبت نام با استفاده از شماره تلفن (احراز هویت پیامکی)
۲. چت خصوصی
۳. افزودن مخاطب
۴. آخرین بازدید کاربر
۵. نمایش جداگانه لیست مخاطبین آنلاین
۶. نوار وضعیت اتصال به سرور در صفحه خانه
۷. اتصال خودکار به سرور هنگام قطع و وصل شدن اینترنت
۸. نمایش آفلاین پیام‌ها
۹. نمایش آفلاین مخاطبین
۱۰. طراحی رابط کاربری با استاندارد متریال دیزاین
۱۱. اعلان پیام جدید
۱۲. امکان تغییر تصویر پشت زمینه‌ی چت
۱۳. امکان تغییر فونت پیام‌ها در چت
۱۴. تصویر پروفایل
۱۵. همگام سازی خودکار اطلاعات در پشت زمینه
۱۶. نمایش وضعیت مشاهده شدن پیام توسط مخاطب

۳-۴- اسکرین‌شات از محیط برنامه

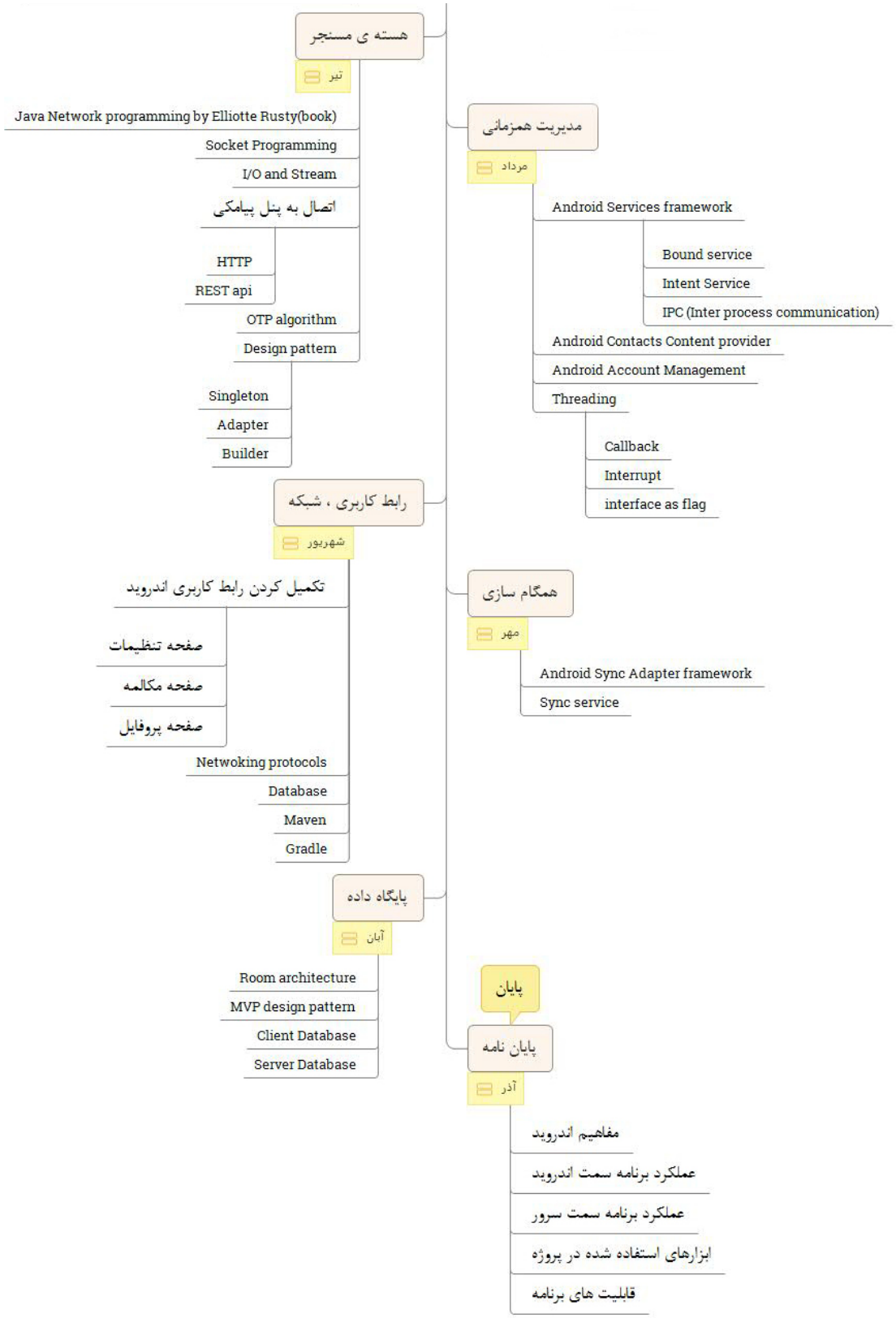


شکل (۷-۴) اسکرین‌شات از محیط برنامه

۴-۴- درخت زمان‌بندی کار روی پروژه



شکل (۸-۴) درخت زمان‌بندی پروژه (۱)



شکل (۹-۴) درخت زمان‌بندی پروژه (۲)

منابع

منابع

[1] G. Developers, "Application Fundamentals," 24 December 2017. [Online]. Available: https://developer.android.com/guide/components/fundamentals.html.

[2] M. L. Murphy, The Busy Coder's Guide to Android Development, COMMONSWARE, 2017.

[3] D. G. Dawn Griffiths, Head First Android Development, Oreily©, 2017.

[4] "Intents and Intent Filters," Google, 2017. [Online]. Available: https://developer.android.com/guide/components/intents-filters.html.

[5] "Services," Google, 2017. [Online]. Available: https://developer.android.com/guide/components/services.html.

[6] "Broadcasts," Google, 2017. [Online]. Available: https://developer.android.com/guide/components/broadcasts.html.

[7] A. Hunt and D. Thomas, Pragmatic Programmer, The: From Journeyman to Master, 1999.

[8] "Saving Data Using the Room Persistence Library," Google, 2017. [Online]. Available: https://developer.android.com/training/data-storage/room/index.html.

[9] E. R. Harold, Java Network Programming 4th ed, Oreilly, October 2013.

[10] "ACID principle in database design," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/ACID.

[11] K. Sierra and B. Bates, Head First Java Second edition, Oreilly, 2005.