



دانشکده مهندسی کامپیوتر

گزارش Polaris

شبکه های تلفن همراه

عرفان همتی	حامد سادات	مهران رزاقی
۴۰۰۵۲۲۳۱۹	۴۰۰۵۲۱۴۰۵	۴۰۰۵۲۱۳۶۹

نیم سال دوم  
سال تحصیلی ۱۴۰۴-۱۴۰۳

# فهرست مطالب

۶	۱	مقدمه
۶	۱.۱	بیان مسئله
۷	۲.۱	تعریف پروژه
۷	۳.۱	هدف کلی
۷	۴.۱	اجزاء پروژه
۹	۲	اندروید
۹	۱.۲	معماری Polaris Client
۱۰	۱.۱.۲	اجزای اصلی
۱۱	۲.۱.۲	تزریق وابستگی با Hilt
۱۱	۳.۱.۲	وظایف پس‌زمینه و WorkManager
۱۱	۴.۱.۲	مدیریت دسترسی‌ها
۱۲	۵.۱.۲	سرویس foreground برای تست شبکه
۱۲	۶.۱.۲	ارتباط شبکه و مدل‌های داده
۱۲	۷.۱.۲	چرخه حیات برنامه و تجربه کاربری
۱۳	۸.۱.۲	مقداردهی Manifest و تنظیمات امنیتی
۱۳	۹.۱.۲	خلاصه فناوری‌ها
۱۳	۲.۲	احراز هویت
۱۳	۱.۲.۲	لایه دامنه (Domain Layer)
۱۴	۲.۲.۲	لایه داده (Data Layer)
۱۵	۳.۲.۲	تزریق وابستگی (Dependency Injection)

۱۵	لایه ارائه (Presentation Layer)	۴.۲.۲
۱۷	صفحه اصلی و اندازه‌گیری شبکه	۳.۲
۱۷	لایه دامنه (Domain Layer)	۱.۳.۲
۱۷	لایه داده (Data Layer)	۲.۳.۲
۱۸	تزریق وابستگی (Dependency Injection)	۳.۳.۲
۱۸	لایه ارائه (Presentation Layer)	۴.۳.۲
۱۹	ذخیره‌سازی در پایگاه داده	۴.۲
۱۹	لایه دامنه (Domain Layer)	۱.۴.۲
۲۰	لایه داده (Data Layer)	۲.۴.۲
۲۰	تزریق وابستگی (Dependency Injection)	۳.۴.۲
۲۰	لایه ارائه (Presentation Layer)	۴.۴.۲
۲۰	همگام‌سازی با سرور	۵.۲
۲۱	لایه دامنه (Domain Layer)	۱.۵.۲
۲۱	لایه داده (Data Layer)	۲.۵.۲
۲۲	تزریق وابستگی (Dependency Injection)	۳.۵.۲
۲۲	لایه ارائه (Presentation Layer)	۴.۵.۲
۲۲	تنظیمات و ترجیحات	۶.۲
۲۲	لایه دامنه (Domain Layer)	۱.۶.۲
۲۳	لایه داده (Data Layer)	۲.۶.۲
۲۳	تزریق وابستگی (Dependency Injection)	۳.۶.۲
۲۴	لایه ارائه (Presentation Layer)	۴.۶.۲
۲۴	دریافت دسترسی و اجرا در پس‌زمینه	۷.۲
۲۴	لایه دامنه (Domain Layer)	۱.۷.۲
۲۵	لایه داده (Data Layer)	۲.۷.۲
۲۵	لایه ارائه (Presentation Layer)	۳.۷.۲
۲۶	سرویس‌ها و اجرای پس‌زمینه	۴.۷.۲
۲۶	مدیریت در سطح فعالیت اصلی	۵.۷.۲
۲۶	چالش‌ها و محدودیت‌ها	۸.۲

## ۳ بک‌اند

۲۸	ساختار و جزئیات Polaris Server	۱.۳
----	--------------------------------	-----

۲۸	.....	فهرست اجزاء	۲.۳
۲۹	.....	فایل پیکربندی و وابستگی ها	۳.۳
۲۹	.....	requirements.txt	۱.۳.۳
۲۹	.....	تنظیمات اصلی (polaris/settings.py)	۲.۳.۳
۲۹	.....	مدل های داده	۴.۳
۲۹	.....	User	۱.۴.۳
۳۰	.....	Device	۲.۴.۳
۳۰	.....	Measurement	۳.۴.۳
۳۱	.....	TestResult	۴.۴.۳
۳۱	.....	سریالایزرها	۵.۳
۳۱	.....	اپلیکیشن users	۶.۳
۳۱	.....	معرفی و نقش	۱.۶.۳
۳۲	.....	مدل ها	۲.۶.۳
۳۲	.....	لیست کامل Endpoint ها	۳.۶.۳
۳۴	.....	نحوه پیاده سازی و نکات فنی	۴.۶.۳
۳۴	.....	ویوها و API ها	۷.۳
۳۴	.....	مسیرها URLs	۱.۷.۳
۳۴	.....	ViewSet Measurement	۲.۷.۳
۳۵	.....	ViewSet TestResult	۳.۷.۳
۳۵	.....	ViewSet BulkUpload	۴.۷.۳
۳۵	.....	ViewSet BulkDelete	۵.۷.۳
۳۵	.....	ViewSet HTTPTest	۶.۷.۳
۳۵	.....	احراز هویت و مجوزها	۸.۳
۳۶	.....	زیرساخت و دیپلوی	۹.۳
۳۶	.....	Dockerfile	۱.۹.۳
۳۶	.....	entrypoint.sh	۲.۹.۳
۳۶	.....	پایگاه داده	۳.۹.۳
۳۶	.....	compose.yml docker	۴.۹.۳
۳۶	.....	فایل .env	۵.۹.۳
۳۷	.....	استفاده از nginx	۶.۹.۳
۳۷	.....	بکاپ گیری	۷.۹.۳

۳۷	۱۰.۳ ابزارها و توابع کمکی
۳۷	۱.۱۰.۳ reports mobile در utils.py
۳۷	۲.۱۰.۳ utils.py users
۳۷	۱۱.۳ ملاحظات امنیتی و عملکردی
۳۸	۱۲.۳ دستورالعمل اجرای محلی
۳۹	۴ فرانت‌اند
۳۹	۱.۴ ساختار و جزئیات Web Application
۳۹	۱.۱.۴ فایل‌ها و پی‌کربندی سطح بالا
۴۰	۲.۱.۴ پوشه public
۴۰	۳.۱.۴ پوشه src
۴۲	۲.۴ احراز هویت
۴۲	۱.۲.۴ مدیریت وضعیت (State Management)
۴۲	۲.۲.۴ مسیریابی محافظت‌شده (Protected Routing)
۴۳	۳.۲.۴ راه‌اندازی سراسری (Global Setup)
۴۳	۴.۲.۴ مدیریت منطق تجاری (Business Logic)
۴۴	۵.۲.۴ لایه رابط کاربری (UI Layer)
۴۶	۳.۴ داشبورد
۴۶	۱.۳.۴ مدیریت منطق تجاری (Business Logic)
۴۶	۲.۳.۴ هوک‌های سفارشی (Custom Hooks)
۴۷	۳.۳.۴ لایه رابط کاربری (UI Layer)
۵۰	۴.۳.۴ ابزارهای کمکی (Utility Functions)
۵۰	۴.۴ نقشه
۵۰	۱.۴.۴ مرور کلی
۵۱	۲.۴.۴ ساختار کلی و نقش هر بخش
۵۱	۳.۴.۴ نحوه واکنش و مدیریت داده‌ها
۵۲	۴.۴.۴ رندر نقشه و کنترل‌ها
۵۲	۵.۴.۴ منطق تعیین رنگ و نمایش نشانگرها
۵۳	۶.۴.۴ توابع کمکی کلیدی
۵۳	۷.۴.۴ جریان داده
۵۴	۵.۴ مدیریت کاربران و سطح دسترسی

۵۴	.....	مدیریت منطق تجاری (Business Logic)	۱.۵.۴
۵۴	.....	هوک‌های سفارشی (Custom Hooks)	۲.۵.۴
۵۴	.....	لایه رابط کاربری (UI Layer)	۳.۵.۴

# فصل ۱

## مقدمه

### ۱.۱ بیان مسئله

با گسترش روزافزون شبکه‌های تلفن همراه و تنوع خدمات ارائه‌شده بر بستر آن‌ها، ارزیابی دقیق کیفیت شبکه و تجربه‌ی کاربری به یک ضرورت کلیدی برای اپراتورها، نهادهای نظارتی و حتی کاربران عادی تبدیل شده است. چالش اصلی در این حوزه، نبود یک سامانه‌ی یکپارچه و دقیق برای جمع‌آوری، پردازش و تحلیل داده‌های میدانی از دیدگاه کاربر نهایی است.

روش‌های سنتی ارزیابی شبکه معمولاً متکی بر ابزارهای تخصصی و آزمون‌های آزمایشگاهی هستند که هزینه‌بر بوده و لزوماً شرایط واقعی استفاده‌ی کاربران را منعکس نمی‌کنند. این موضوع باعث ایجاد شکاف میان شاخص‌های فنی ثبت‌شده توسط اپراتورها و تجربه‌ی واقعی کاربران می‌شود.

بنابراین، نیاز به سیستمی که بتواند به‌صورت خودکار، مداوم و در شرایط واقعی داده‌های عملکرد شبکه را جمع‌آوری کرده، پردازش و تحلیل کند و نتایج را به‌صورت قابل فهم و عملیاتی ارائه دهد، بیش از پیش احساس می‌شود.

## ۲.۱ تعریف پروژه

پروژه‌ی Polaris یک سامانه‌ی جامع برای پایش و تحلیل کیفیت شبکه‌های تلفن همراه است که با هدف ارائه‌ی داده‌های دقیق و قابل اتکا از تجربه‌ی واقعی کاربران طراحی و پیاده‌سازی شده است. این سامانه با ترکیب ابزارهای جمع‌آوری داده، پردازش متمرکز، و ارائه‌ی گزارش‌های تحلیلی، امکان ارزیابی مستمر وضعیت شبکه و شناسایی نقاط ضعف و قوت آن را فراهم می‌کند.

در این پروژه، داده‌های میدانی به‌صورت خودکار از محیط واقعی جمع‌آوری شده، پردازش و تحلیل می‌شوند و نتایج به شکل قابل فهم و کاربردی در اختیار کاربران ذی‌نفع قرار می‌گیرد. این رویکرد شکاف میان شاخص‌های فنی اپراتور و تجربه‌ی واقعی کاربر را کاهش می‌دهد و بستری برای تصمیم‌گیری مبتنی بر داده فراهم می‌سازد.

## ۳.۱ هدف کلی

هدف کلی پروژه‌ی Polaris طراحی و پیاده‌سازی سامانه‌ای یکپارچه برای پایش و تحلیل کیفیت شبکه‌های تلفن همراه از دیدگاه کاربر نهایی است. این سامانه با جمع‌آوری داده‌های میدانی در شرایط واقعی، پردازش هوشمندانه و ارائه نتایج به‌صورت گزارش‌ها و نمودارهای تحلیلی، امکان ارزیابی دقیق‌تر کیفیت خدمات شبکه را فراهم می‌سازد.

این پروژه می‌کوشد با ارائه ابزارهایی برای اندازه‌گیری پارامترهای مختلف شبکه مانند کیفیت سیگنال، سرعت انتقال داده، و زمان پاسخ‌گویی سرویس‌ها، شکاف موجود بین شاخص‌های فنی اپراتورها و تجربه واقعی کاربران را کاهش دهد و به بهبود کیفیت خدمات کمک نماید.

## ۴.۱ اجزاء پروژه

پروژه‌ی Polaris از سه بخش اصلی تشکیل شده است که هر یک نقش مکملی در جمع‌آوری، پردازش و نمایش داده‌های کیفیت شبکه ایفا می‌کنند:

۱. **کلاینت اندرویدی:** یک برنامه موبایل توسعه‌یافته با زبان Kotlin که وظیفه‌ی انجام تست‌های مختلف شبکه از جمله اندازه‌گیری سرعت دانلود و آپلود، زمان پاسخ Ping، زمان پاسخ DNS، ارسال و دریافت پیامک، و اندازه‌گیری پارامترهای کیفیت سیگنال را بر عهده دارد. داده‌های حاصل به‌صورت ساختاریافته به سرور ارسال می‌شوند.

۲. **بک‌اند:** سروری مبتنی بر فریم‌ورک Django که وظیفه‌ی دریافت، اعتبارسنجی، پردازش و ذخیره‌سازی



داده‌ها در پایگاه داده MySQL را دارد. این بخش از طریق API‌هایی امن، ارتباط میان کلاینت‌ها و رابط کاربری وب را مدیریت می‌کند.

۳. رابط کاربری وب: یک پنل مدیریتی و تحلیلی مبتنی بر فناوری‌های وب که داده‌های ذخیره‌شده را در قالب نقشه‌های تعاملی، نمودارهای تحلیلی و جداول قابل جست‌وجو نمایش می‌دهد و امکان تحلیل و استخراج گزارش‌های مختلف را فراهم می‌کند.

## فصل ۲

# اندروید

### ۱.۲ معماری Polaris Client

کلاینت Polaris بر اساس اصول معماری پاک (Clean Architecture) و MVVM (Model-View-) ساخته شده است و کدی ماژولار، قابل تست و قابل نگهداری دارد:

- لایه ارائه: رابط کاربری با Jetpack Compose، ViewModel هایی که توسط Hilt مدیریت می شوند، برای کنترل وضعیت UI و تعاملات کاربر.
- لایه دامنه: استفاده از Use Case ها برای کپسوله کردن منطق کسب و کار و تعامل با رابط های Repository.
- لایه داده: پیاده سازی Repository ها برای تعامل با دیتابیس محلی Room و API های راه دور Retrofit.
- لایه سرویس: سرویس های اندروید برای اجرای وظایف طولانی در پس زمینه، مانند تست شبکه.
- تزریق وابستگی: استفاده از Hilt برای مدیریت وابستگی ها و جداسازی مسئولیت ها.
- WorkManager: مدیریت همگام سازی دوره ای داده ها با سرور به صورت آگاه از چرخه حیات و بهینه شده برای باتری.

## ۱.۱.۲ اجزای اصلی

### لایه ارائه

- صفحات Jetpack Compose: اجزای UI اعلانی شامل صفحات دسترسی‌ها، تنظیمات و داشبورد اصلی.
- PermissionsViewModel: مدیریت وضعیت دسترسی‌ها، نمایش دیالوگ‌های توضیح دسترسی و به‌روزرسانی UI هنگام تغییر دسترسی‌ها.
- ناوبری: استفاده از Jetpack Navigation Compose برای جابجایی بین صفحات، شامل صفحه دسترسی‌ها و تنظیمات.
- مدیریت وضعیت: استفاده از Kotlin StateFlow و Compose mutableStateOf برای به‌روزرسانی واکنشی UI.

### لایه دامنه

- Use Case ها: کپسوله کردن منطق مانند بررسی دسترسی‌ها (PermissionUseCase)، همگام‌سازی داده‌ها و اجرای تست‌های شبکه.
- مدل‌ها: مدل‌های دامنه نماینده داده‌های شبکه و درخواست‌های اندازه‌گیری.

### لایه داده

- پیاده‌سازی Repository: مانند NetworkRepositoryImpl که داده‌ها را از دیتابیس Room و API شبکه مدیریت می‌کند.
- دیتابیس Room: ذخیره محلی داده‌های اندازه‌گیری شبکه (NetworkDataDao).
- API Retrofit: ارتباط با API بک‌اند برای بارگذاری دسته‌ای داده‌های شبکه.

### لایه سرویس

- TestExecutionService: یک سرویس اندروید در حالت Foreground که تست‌های شبکه را اجرا می‌کند و مجوزها و وضعیت دستگاه را رعایت می‌کند.
- DataSyncWorker: CoroutineWorker در WorkManager که داده‌های همگام‌نشده شبکه را به صورت دوره‌ای به سرور ارسال می‌کند. وابستگی‌ها توسط Hilt تزریق می‌شوند.

### ۲.۱.۲ تزریق وابستگی با Hilt

- برنامه با افزودن @HiltAndroidApp به کلاس PolarisApp قابلیت‌های Hilt را فعال می‌کند.
- Hilt وابستگی‌ها را به ViewModel ها، Worker ها، Service ها و Repository ها تزریق می‌کند.
- DataSyncWorker از @HiltWorker و تزریق ترکیبی (Assisted Injection) برای دریافت Context، WorkerParameters و سایر وابستگی‌ها بهره می‌برد.
- کلاس PolarisApp رابط Configuration.Provider را پیاده‌سازی می‌کند تا WorkManager با WorkerFactory ساخته شده توسط Hilt کار کند.

### ۳.۱.۲ وظایف پس‌زمینه و WorkManager

- DataSyncWorker وظیفه همگام‌سازی دوره‌ای داده‌ها را با محدودیت‌هایی مانند اتصال شبکه انجام می‌دهد.
- ابزار DataSyncScheduler زمان‌بندی و مدیریت وظایف همگام‌سازی را بر اساس فواصل تنظیم شده توسط کاربر (ذخیره شده در SharedPreferences) بر عهده دارد.
- استفاده از PeriodicWorkRequestBuilder برای درج منحصر به فرد وظایف در صف اجرا و جلوگیری از تکراری شدن.
- ادغام با WorkManager تضمین می‌کند که وظایف همگام‌سازی با سلامت سیستم (باتری، شبکه) همخوانی داشته باشند و پس از ری‌استارت برنامه نیز ادامه یابند.

### ۴.۱.۲ مدیریت دسترسی‌ها

- دسترسی‌ها با کمک PermissionManager و PermissionUtils دسته‌بندی و بررسی می‌شوند.
- برنامه دسترسی‌ها را به صورت پویا درخواست می‌کند و در صورت نیاز، کاربر را به تنظیمات سیستم برای تایید دستی هدایت می‌کند.
- PermissionsViewModel وضعیت UI مربوط به لیست دسترسی‌ها و دیالوگ‌های توضیح را فراهم می‌کند.
- UI با استفاده از Context فعال، Intent‌های تنظیمات را به شکل ایمن اجرا می‌کند تا با محدودیت‌های اندروید سازگار باشد.

- دسترسی‌ها شامل لوکیشن (پس‌زمینه و جلویی)، پیامک‌ها، وضعیت تلفن، اعلان‌ها، آلارم‌های دقیق و بهینه‌سازی باتری است.

## ۵.۱.۲ سرویس foreground برای تست شبکه

- TestExecutionService تست‌های شبکه را در حالت Foreground اجرا می‌کند تا اولویت بالایی داشته باشد و توسط سیستم بسته نشود.
- قبل از شروع، مجوزهای لازم و وضعیت لوکیشن دستگاه را بررسی می‌کند.
- به صورت asynchronous با استفاده از Kotlin Coroutine اجرا می‌شود.
- اعلان دائمی با استفاده از کانال اعلان (Notification Channel) طبق بهترین شیوه‌های اندروید نمایش می‌دهد.
- از Repository تزریق شده برای دریافت توکن و اجرای تست‌های شبکه استفاده می‌کند.

## ۶.۱.۲ ارتباط شبکه و مدل‌های داده

- داده‌های شبکه با استفاده از کلاس‌های داده‌ای Kotlin مانند NetworkData و MeasurementRequest مدل‌سازی شده‌اند.
- بارگذاری JSON با استفاده از کتابخانه Gson انجام می‌شود.
- توکن‌های احراز هویت به صورت ایمن از طریق CookieManager مدیریت شده و با Kotlin Flows به‌روزرسانی می‌شوند.
- تماس‌های شبکه با Retrofit و همراه با header ها و مدیریت خطاهای مناسب انجام می‌گیرند.

## ۷.۱.۲ چرخه حیات برنامه و تجربه کاربری

- ناوبری توسط Compose Navigation مدیریت می‌شود.
- وضعیت دسترسی‌ها در رویدادهای چرخه حیات مانند ON\_RESUME به روز می‌شود تا تغییرات انجام شده خارج از برنامه نیز منعکس شود.
- دیالوگ‌های توضیح دسترسی، راهنمایی شفاف و ناوبری مستقیم به تنظیمات سیستم ارائه می‌دهند.
- سرویس Foreground و همگام‌سازی پس‌زمینه به گونه‌ای طراحی شده‌اند که حداقل تاثیر بر تجربه کاربر داشته باشند و در عین حال داده‌ها دقیق باقی بمانند.

## ۸.۱.۲ مقداردهی Manifest و تنظیمات امنیتی

- فایل AndroidManifest.xml شامل تمامی مجوزهای مورد نیاز با دستورهای uses-permission و uses-feature است.
- نوع سرویس‌های Foreground و ویژگی exported با دقت برای امنیت و تطابق تنظیم شده‌اند.
- تنظیمات امنیت شبکه از استانداردهای مدرن اندروید برای ارتباط HTTPS پشتیبانی می‌کند.

## ۹.۱.۲ خلاصه فناوری‌ها

فناوری	کاربرد
Jetpack Compose & Kotlin	زبان مدرن و ابزار UI برای اندروید
Hilt	فریمورک تزریق وابستگی
WorkManager	زمان‌بندی و اجرای وظایف پس‌زمینه
Room Database	ذخیره‌سازی محلی داده‌ها
Gson & Retrofit	API شبکه و سریال‌سازی JSON
Coroutine Workers	وظایف پس‌زمینه asynchronous
Android Services	وظایف طولانی مدت در foreground
Android Permissions API	مدیریت پویا دسترسی‌ها

## ۲.۲ احراز هویت

این بخش از برنامه مسئول مدیریت ورود، ثبت‌نام، تأیید حساب و بازیابی رمز عبور است. ساختار این بخش بر اساس معماری پاک (Clean Architecture) و الگوی MVVM پیاده‌سازی شده و شامل چهار لایه اصلی است: دامنه، داده، تزریق وابستگی، و ارائه.

### ۱.۲.۲ لایه دامنه (Domain Layer)

- `/model/LoginRequest.kt`: مدل داده‌ای که برای ارسال درخواست ورود به سرور استفاده می‌شود. شامل شماره تلفن یا ایمیل کاربر و گذرواژه او است. از `@SerializedName` برای نگاشت کلیدهای JSON به فیلدهای کاتلین استفاده می‌کند.
- `/model/LoginResponse.kt`: مدل داده‌ای که پاسخ موفقیت‌آمیز ورود را از سرور دریافت می‌کند. حاوی توکن دسترسی و ایمیل کاربر است و برای ذخیره یا استفاده در سایر بخش‌ها به کار می‌رود.

- `/model/LoginResult.kt`: کلاس مهر و موم شده (sealed class) که نتیجه فرآیند ورود را مدل می‌کند. سه حالت دارد: موفقیت، نیاز به تأیید حساب، و خطا با پیام مشخص.
- `/model/ResetPasswordRequest.kt`: مدل درخواست تغییر رمز عبور که شامل شماره یا ایمیل کاربر، کد تأیید و گذرواژه جدید است. برای مرحله نهایی بازیابی رمز استفاده می‌شود.
- `/model/SendResetCodeRequest.kt`: مدلی برای ارسال درخواست دریافت کد بازیابی رمز عبور. تنها شامل شماره یا ایمیل کاربر بوده و به سرور فرستاده می‌شود.
- `/model/SignUpRequest.kt`: مدل داده‌ای برای ثبت نام کاربر جدید. شامل ایمیل، شماره تلفن و گذرواژه است. شماره تلفن با کلید JSON "phone\_number" نگاشت می‌شود.
- `/model/VerificationRequest.kt`: مدل درخواست تأیید حساب که شامل شماره یا ایمیل کاربر، گذرواژه و کد تأیید است. این مدل معمولاً بعد از ثبت نام یا ورود مشروط استفاده می‌شود.
- `/model/VerificationRetryRequest.kt`: مدلی برای ارسال دوباره کد تأیید حساب. تنها شماره یا ایمیل کاربر را به سرور می‌فرستد تا کد جدید صادر شود.
- `/model/VerifyResetCodeRequest.kt`: مدلی برای بررسی اعتبار کد بازیابی رمز عبور که شامل شماره یا ایمیل کاربر و کد ارسال شده است.
- `/repository/AuthRepository.kt`: یک اینترفیس که قرارداد عملیات احراز هویت را تعریف می‌کند. شامل متدهایی برای ثبت نام، ورود، تأیید حساب، ارسال دوباره کد، ارسال کد بازیابی رمز، تأیید کد بازیابی و تغییر رمز عبور است. این لایه مستقل از جزئیات پیاده‌سازی عمل می‌کند.
- `/usecase/auth/Auth.kt`: شامل مجموعه‌ای از کلاس‌های UseCase مربوط به عملیات احراز هویت است. هر UseCase یک عمل خاص (مثل ثبت نام یا ورود) را انجام داده و درخواست را به مخزن داده (Repository) می‌فرستد. این لایه منطق کاربردی را از جزئیات داده جدا می‌کند.
- `/usecase/auth/AuthUseCases.kt`: یک کلاس داده که تمام UseCase‌های مرتبط با احراز هویت را در یک ساختار واحد گردآوری می‌کند. این کار تزریق وابستگی (DI) و استفاده از چندین UseCase را ساده‌تر می‌سازد.

## ۲.۲.۲ لایه داده (Data Layer)

- `/local/CookieManager.kt`: کلاسی برای مدیریت ذخیره‌سازی و بازیابی داده‌های احراز هویت مانند توکن و ایمیل کاربر با استفاده از DataStore. شامل متدهایی برای ذخیره، خواندن و پاک کردن داده‌ها است. این کلاس به صورت Singleton @Singleton تعریف شده تا در کل برنامه نمونه یکتا

داشته باشد.

- **remote/AuthApi.kt**: اینترفیس Retrofit که متدهای ارتباط با سرور برای عملیات احراز هویت را تعریف می‌کند. شامل ثبت‌نام، ورود، تأیید حساب، ارسال و تأیید کد بازیابی رمز و تغییر گذرواژه است. هر متد یک درخواست HTTP POST به آدرس مشخصی از API ارسال می‌کند.
- **repository/AuthRepositoryImpl.kt**: پیاده‌سازی رابط AuthRepository که با استفاده از AuthApi درخواست‌های شبکه را ارسال کرده و با CookieManager داده‌های احراز هویت را ذخیره می‌کند. شامل متدهایی برای ثبت‌نام، ورود، تأیید حساب، ارسال و تأیید کد بازیابی و تغییر رمز عبور است. هر عملیات با مدیریت خطا و پاسخ‌های سرور انجام می‌شود.

### ۳.۲.۲ تزریق وابستگی (Dependency Injection)

- **NetworkModule.kt**: ماژول Hilt برای پیکربندی و ارائه وابستگی‌های شبکه شامل Retrofit، OkHttpClient و AuthApi. این ماژول تنظیمات لاگ‌برداری درخواست‌ها و پاسخ‌ها را فعال کرده و پایه ارتباط HTTP را با آدرس سرور فراهم می‌کند.
- **RepositoryModule.kt**: ماژول Hilt برای ساخت و ارائه وابستگی‌های مخزن داده (Repository). در اینجا نمونه AuthRepository با استفاده از AuthRepositoryImpl ساخته شده و به‌صورت @Singleton برای استفاده در کل برنامه در دسترس قرار می‌گیرد.
- **UseCaseModule.kt**: ماژول Hilt برای ارائه‌ی مجموعه‌ی کامل AuthUseCases شامل سناریوهای ثبت‌نام، ورود، تأیید، ارسال و تأیید کد بازیابی و تغییر رمز عبور. این ماژول وابستگی‌ها را از طریق AuthRepository به هر UseCase تزریق می‌کند.

### ۴.۲.۲ لایه ارائه (Presentation Layer)

- **auth/components/ResetIdentifierStep.kt**: کامپوننت Jetpack Compose برای دریافت ایمیل یا شماره تلفن کاربر به‌منظور شروع فرآیند بازیابی رمز عبور. شامل اعتبارسنجی ورودی و نمایش پیام خطا قبل از ارسال کد به کاربر است.
- **auth/components/ResetPasswordStep.kt**: کامپوننت Jetpack Compose برای وارد کردن و تأیید رمز عبور جدید در فرآیند بازیابی. دارای ورودی‌های متنی با قابلیت نمایش/مخفی کردن رمز، اعتبارسنجی همخوانی رمزها، و نمایش خطاهای مرتبط.
- **auth/components/ResetVerificationStep.kt**: کامپوننت Jetpack Compose



برای وارد کردن کد تأیید پنج رقمی در فرآیند بازیابی رمز عبور. شامل مدیریت فوکوس بین فیلدها، اعتبارسنجی کد، و امکان درخواست ارسال مجدد کد است.

- `/auth/AuthUiState.kt`: کلاس `sealed` برای تعریف حالات مختلف رابط کاربری احراز هویت مانند آماده، در حال بارگذاری، موفقیت، نیاز به تأیید، کد ارسال شده، کد تأیید شده و خطا با پیام مشخص.

- `/auth/AuthViewModel.kt`: کلاس `ViewModel` مسئول مدیریت منطق احراز هویت و ارتباط بین لایه `UI` و `Domain`. شامل متدهای اعتبارسنجی ورودی‌ها، ثبت نام، ورود، تأیید هویت، ارسال و تأیید کد بازیابی، تغییر رمز عبور و بررسی وضعیت ورود کاربر است. از `StateFlow` برای مدیریت وضعیت رابط کاربری و `Hilt` برای تزریق وابستگی‌ها استفاده می‌کند.

- `/auth/LoginScreen.kt`: کامپوزیل صفحه ورود که شامل فرم ورود با فیلدهای شماره یا ایمیل و گذرواژه است. اعتبارسنجی ورودی‌ها انجام شده و خطاها نمایش داده می‌شوند. از `Snackbar` برای نمایش پیام‌ها استفاده می‌کند و در هنگام بارگذاری، `DotsLoader` نمایش می‌دهد. همچنین مسیرهای ناوبری به ثبت نام، فراموشی رمز و تأیید کد فعال است.

- `/auth/ResetPasswordScreen.kt`: صفحه بازیابی رمز عبور چندمرحله‌ای که شامل مراحل وارد کردن شناسه (ایمیل یا شماره)، کد تأیید و تنظیم رمز جدید است. وضعیت `UI` بر اساس پاسخ‌ها و مراحل تغییر می‌کند و پیام‌های مربوطه با `Snackbar` نمایش داده می‌شوند. ناوبری برگشت به مراحل قبلی و به صفحه ورود مدیریت شده است. از `DotsLoader` برای نمایش حالت بارگذاری استفاده می‌شود.

- `/auth/SignUpScreen.kt`: صفحه ثبت نام کاربر با فیلدهای ایمیل، شماره تلفن و رمز عبور همراه با اعتبارسنجی ورودی‌ها و نمایش خطاها. شامل کنترل نمایش یا مخفی سازی رمز عبور و نمایش وضعیت بارگذاری است. پس از ثبت نام موفق، کاربر به مرحله تأیید هدایت می‌شود. همچنین امکان انتقال به صفحه ورود فراهم شده است.

- `/auth/SplashScreen.kt`: صفحه آغازین برنامه که هنگام بارگذاری وضعیت ورود کاربر را بررسی می‌کند. در این صفحه نشانگر بارگذاری (`DotsLoader`) نمایش داده شده و پس از اعتبارسنجی توکن، به صفحه مناسب هدایت می‌شود.

- `/auth/VerificationScreen.kt`: صفحه تأیید کد ارسالی به ایمیل یا شماره تلفن کاربر با پنج فیلد ورود کد و کنترل فوکوس خودکار بین آنها. این صفحه شامل اعتبارسنجی کد، ارسال مجدد کد و نمایش پیام‌های موفقیت یا خطا از طریق `Snackbar` است. همچنین قابلیت بازگشت به صفحه قبل را دارد.

• **MainActivity.kt**: کلاس اصلی برنامه که نقطه ورود اپلیکیشن است و ناوبری صفحات مختلف را مدیریت می‌کند. با استفاده از Compose Jetpack و Compose Navigation مسیرهای مختلف مانند صفحه شروع (Splash)، ورود، ثبت‌نام، تأیید کد، بازنشانی رمز عبور و صفحه اصلی را تعریف می‌کند. همچنین با مدیریت هوشمند مسیرها و پارامترهای ناوبری، جریان کاربری را روان و یکپارچه نگه می‌دارد.

## ۳.۲ صفحه اصلی و اندازه‌گیری شبکه

این بخش مسئول نمایش وضعیت شبکه، اجرای تست‌ها، جمع‌آوری داده‌های سلولی و ذخیره‌سازی نتایج است. ساختار آن مشابه سایر بخش‌ها بر اساس معماری پاک (Clean Architecture) و الگوی (MVVM) پیاده‌سازی شده و شامل چهار لایه اصلی است: دامنه، داده، تزریق وابستگی و ارائه.

### ۱.۳.۲ لایه دامنه (Domain Layer)

• **/model/NetworkData.kt**: مدل داده‌ای اصلی برای نمایش اطلاعات شبکه. شامل مختصات کاربر (طول و عرض جغرافیایی)، نسل شبکه، شناسه سلول، فرکانس، باند فرکانسی، PLMN ID، پارامترهای سیگنال (RSRP، RSRQ، RxLev) و غیره و نتایج تست‌های عملکردی (آپلود، دانلود، PING، DNS، Web Response).

• **/model/TestSelection.kt**: مدل داده‌ای که انتخاب تست‌ها توسط کاربر را مدل می‌کند (آپلود، دانلود، PING، DNS، Web Response، SMS). این انتخاب‌ها تعیین می‌کنند کدام تست‌ها توسط TestRunner اجرا شوند.

### ۲.۳.۲ لایه داده (Data Layer)

• **/utils/ParametersUtility.kt**: این کد اطلاعات سلولی شبکه‌های موبایل (GSM، WCDMA، LTE) را پردازش می‌کند. با استفاده از کلاس‌های CellInfo، جزئیات هر نوع شبکه را استخراج می‌کند. فرکانس و باند هر سلول را بر اساس مقادیر EARFCN، UARFCN و EARFCN محاسبه می‌کند. نتیجه را در قالب یک شیء NetworkData که شامل اطلاعاتی مانند شناسه سلول، قدرت سیگنال و نوع شبکه است، برمی‌گرداند. این داده‌ها برای ذخیره در پایگاه داده و نمایش در رابط کاربری آماده می‌شوند.

• **/utils/tests/\***: مجموعه کلاس‌های اجرای تست‌ها شامل HttpDownloadTest.kt و

HttpUploadTest.kt برای دانلود و آپلود، SmsTest.kt برای پیامک، PingTest.kt برای پینگ، DnsTest.kt برای زمان پاسخ DNS و WebTest.kt برای زمان بارگذاری صفحات وب. نتایج این تست‌ها مستقیماً در NetworkData درج می‌شوند.

### ۳.۳.۲ تزریق وابستگی (Dependency Injection)

- **RepositoryModule.kt**: مخازن داده مربوط به شبکه (Network Repository) و ذخیره‌سازی (Room) از این طریق به برنامه تزریق می‌شوند تا HomeViewModel بتواند به سادگی داده‌های شبکه و تست را مدیریت کند.
- **UseCaseModule.kt**: مجموعه‌ای از UseCase‌های مرتبط با دریافت داده شبکه، اجرای تست‌ها و ذخیره‌سازی آن‌ها در پایگاه داده از طریق این ماژول فراهم شده و به ViewModel تزریق می‌شوند.

### ۴.۳.۲ لایه ارائه (Presentation Layer)

- **/home/HomeUiState.kt**: یک sealed class برای مدیریت حالت‌های مختلف صفحه اصلی شامل Loading، Empty، Error، Success (دریافت موفق داده‌های شبکه) و LocationSuccess.
- **/home/HomeViewModel.kt**: ViewModel اصلی که وظیفه مدیریت داده‌های شبکه و اجرای تست‌ها را بر عهده دارد. انتخاب تست‌ها از طریق TestSelection دریافت شده، با استفاده از کلاس‌های تست اجرا می‌شود و نتایج در NetworkData ذخیره می‌گردد. همچنین این کلاس وضعیت صفحه (UI State) را برای نمایش مناسب به رابط کاربری ارسال می‌کند.
- **/home/HomeScreen.kt**: صفحه اصلی رابط کاربری که با استفاده از Compose Jetpack طراحی شده است. شامل نوار بالا (HomeTopBar)، محتوای اصلی (HomeContent) و مدیریت وضعیت از طریق HomeStateContent. کاربر می‌تواند تست‌ها را انتخاب و اجرا کند و نتایج به صورت کارت‌های اطلاعاتی نمایش داده می‌شوند.
- **/home/components/HomeTopBar.kt**: نوار ابزار بالای صفحه اصلی با دکمه‌های تازه‌سازی داده، تنظیمات، دسترسی‌ها و خروج از حساب.
- **/home/components/HomeContent.kt**: ساختار اصلی صفحه که شامل دکمه اجرای تست (RunTestButton)، بخش انتخاب تست‌ها (TestSelectionSection) و نمایش نتایج شبکه (networkResults) است.

- `/home/components/HomeStateContent.kt`: مدیریت وضعیت‌های مختلف صفحه شامل بارگذاری (نمایش DotsLoader)، خطا، نبود داده و نمایش موفق داده‌ها.
- `/home/components/RunTestButton.kt`: دکمه اصلی برای اجرای تست‌ها با افکت انیمیشن پالس. کاربر با فشردن این دکمه تست‌های انتخاب‌شده را اجرا می‌کند.
- `/home/components/TestSelectionSection.kt & TestToggleButton.kt`: رابط کاربری برای انتخاب تست‌های مختلف توسط کاربر (آپلود، دانلود، Web، DNS، PING، Response و SMS). نتایج انتخاب به `HomeViewModel` ارسال می‌شود.
- `/home/components/networkResults.kt`: نمایش نتایج تست‌ها و اطلاعات شبکه در قالب کارت‌های مجزا (`NetworkInfoCard`) شامل اطلاعات کاربر، مشخصات سلول و کیفیت سیگنال.
- `/home/components/KeyMetricCard.kt`: کارت شاخص‌های کلیدی عملکرد برای نمایش نتایج اصلی تست‌ها مانند سرعت آپلود و دانلود، زمان پینگ، پاسخ DNS و زمان بارگذاری وب.
- `/home/components/NetworkInfoCard.kt & KeyValueHandling.kt`: نمایش جزئیات هر بخش از داده‌های شبکه به صورت ردیف‌های کلید-مقدار.

## ۴.۲ ذخیره‌سازی در پایگاه داده

این بخش وظیفه ذخیره‌سازی محلی داده‌های شبکه و نتایج تست‌ها را برعهده دارد. هدف آن فراهم کردن امکان دسترسی آفلاین، نگهداری تاریخچه تست‌ها و آماده‌سازی داده برای همگام‌سازی با سرور است. پیاده‌سازی این بخش مبتنی بر `Room Database` انجام شده است.

### ۱.۴.۲ لایه دامنه (Domain Layer)

- `/model/NetworkData.kt`: مدل داده‌ای که هم در تست‌های شبکه و هم در ذخیره‌سازی پایگاه داده مورد استفاده قرار می‌گیرد. این مدل شامل شناسه یکتا، اطلاعات سلول، مکان کاربر، نتایج تست‌ها و زمان اجرای تست است.
- `/model/NetworkDataDao.kt`: رابط دسترسی به داده‌ها در `Room`. شامل متدهایی برای درج داده جدید، واکشی آخرین تست‌ها و مدیریت تاریخچه تست‌ها است.
- `/model/NetworkDataDatabase.kt`: کلاس پیکربندی پایگاه داده `Room` که انتیتی‌ها

و Dao ها را تعریف می‌کند. این کلاس نقطه‌ی اصلی ارتباط با Room بوده و دیتابیس را مقداردهی اولیه می‌کند.

## ۲.۴.۲ لایه داده (Data Layer)

- `/data/local/AppDatabaseHelper.kt`: این کد یک دیتابیس محلی (Database Room) برای ذخیره‌سازی اطلاعات شبکه ایجاد می‌کند. از الگوی Singleton استفاده می‌کند تا فقط یک نمونه از دیتابیس در طول عمر برنامه وجود داشته باشد. دیتابیس با نام `network_data.db` ساخته می‌شود و در صورت نیاز به تغییر اسکیمای دیتابیس، به صورت خودکار ریست می‌شود (`fallbackToDestructiveMigration`). دسترسی به دیتابیس از طریق متد `getDatabase` امکان‌پذیر است که در صورت عدم وجود، آن را ایجاد می‌کند.

## ۳.۴.۲ تزریق وابستگی (Dependency Injection)

- `DatabaseModule.kt`: ماژول Hilt برای ساخت و ارائه نمونه پایگاه داده و Dao. این ماژول تضمین می‌کند که یک نسخه یکتا از پایگاه داده در سراسر اپلیکیشن مورد استفاده قرار گیرد.

## ۴.۴.۲ لایه ارائه (Presentation Layer)

- `/home/HomeViewModel.kt`: پس از اجرای هر تست، داده‌های به‌دست‌آمده در پایگاه داده ذخیره می‌شوند. ViewModel داده‌های اخیر را بارگذاری کرده و از طریق `UiState` به صفحه اصلی ارسال می‌کند.
- `/home/HomeScreen.kt`: صفحه اصلی علاوه‌بر نمایش داده‌های زنده، تاریخچه تست‌ها را نیز از پایگاه داده دریافت کرده و در رابط کاربری نمایش می‌دهد.

## ۵.۲ همگام‌سازی با سرور

این بخش مسئول انتقال نتایج تست‌های ذخیره‌شده در پایگاه داده محلی به سرور مرکزی است. فرآیند همگام‌سازی به صورت خودکار و دوره‌ای توسط `WorkManager` اجرا شده و تضمین می‌کند که هیچ داده‌ای از دست نرود و رکوردها پس از ارسال موفق به‌روزرسانی شوند.

## ۱.۵.۲ لایه دامنه (Domain Layer)

- `/model/Measurement.kt` و `/model/MeasurementRequest.kt`: ساختار داده‌ای که نتایج تست (موقعیت جغرافیایی، نوع شبکه، شناسه سلول، کیفیت سیگنال و نتایج عملکردی مثل پینگ و DNS) را تعریف می‌کند. این داده‌ها برای ارسال به سرور در قالب `MeasurementRequest` گردآوری می‌شوند.
- `/model/NetworkData.kt`: مدل اصلی ذخیره‌سازی داده در پایگاه داده محلی که بعداً به فرمت `Measurement` تبدیل می‌شود.
- `/model/NetworkDataDao.kt`: رابط DAO برای دریافت داده‌های ذخیره‌نشده (unsynced) و علامت‌گذاری رکوردهای همگام‌شده.

## ۲.۵.۲ لایه داده (Data Layer)

- `/data/remote/NetworkDataApi.kt`: رابط Retrofit برای ارسال داده‌های شبکه به سرور شامل متد `uploadNetworkData` برای ارسال دسته‌ای است.
- `/data/repository/NetworkRepositoryImpl.kt`: پیاده‌سازی `NetworkRepository` که علاوه بر اجرای تست‌ها، متد `uploadNetworkData` را برای همگام‌سازی داده‌های محلی با سرور پیاده‌سازی می‌کند. از `CookieManager` برای دریافت توکن احراز هویت و از `NetworkDataApi` برای ارسال داده استفاده می‌کند.
- `/service/DataSyncWorker.kt`: کلاس `Worker` مبتنی بر `CoroutineWorker` که وظیفه اجرای همگام‌سازی دوره‌ای را بر عهده دارد. داده‌های `unsynced` را از پایگاه داده واکشی می‌کند. با استفاده از `measurementConverter` به مدل `Measurement` تبدیل می‌کند. داده‌ها را با `uploadNetworkDataBatch` به سرور می‌فرستد. در صورت موفقیت رکوردها را با متد `markAsSynced` علامت‌گذاری می‌کند.
- `/utils/DataSyncScheduler.kt`: ابزار زمان‌بندی همگام‌سازی که با استفاده از `WorkManager` کارگر دوره‌ای تعریف می‌کند. کاربر می‌تواند بازه همگام‌سازی (۱۵ دقیقه تا ۲۴ ساعت) را تنظیم کند و این زمان در `SharedPreferences` ذخیره می‌شود.

### ۳.۵.۲ تزریق وابستگی (Dependency Injection)

- **NetworkModule.kt**: نمونه NetworkDataApi را از طریق Retrofit ساخته و در اختیار سایر بخش‌ها قرار می‌دهد.
- **RepositoryModule.kt**: نمونه NetworkRepository را با پیاده‌سازی NetworkRepositoryImpl ایجاد کرده و به‌صورت Singleton در سطح برنامه قابل استفاده می‌کند.

### ۴.۵.۲ لایه ارائه (Presentation Layer)

- **/home/components/HomeContent.kt** و **/home/components/HomeStateContent.kt**: داده‌های ذخیره‌شده پس از هر تست در این کامپوننت‌ها مدیریت شده و به‌صورت پس‌زمینه برای همگام‌سازی آماده می‌شوند.
- **/service/DataSyncWorker.kt**: وضعیت اجرای همگام‌سازی را در لاگ ثبت کرده و نتیجه را به WorkManager برمی‌گرداند تا UI بتواند در صورت نیاز پیام موفقیت یا خطا را نمایش دهد.
- **/utils/DataSyncScheduler.kt**: از طریق تنظیمات برنامه (Settings) کاربر می‌تواند بازه همگام‌سازی را تغییر دهد که مستقیماً در لایه ارائه قابل مدیریت است.

## ۶.۲ تنظیمات و ترجیحات

این بخش امکان پیکربندی ترجیحات کاربر، انتخاب سیم‌کارت، تغییر بازه همگام‌سازی و تعریف پارامترهای تست‌ها را فراهم می‌کند. تمام این مقادیر در حافظه محلی ذخیره شده و در اجرای تست‌ها و همگام‌سازی داده‌ها مورد استفاده قرار می‌گیرند.

### ۱.۶.۲ لایه دامنه (Domain Layer)

- **/domain/usecase/settings/SettingsUseCases.kt**: مجموعه‌ای از UseCase‌ها را برای مدیریت تنظیمات در اختیار ViewModel قرار می‌دهد. این موارد شامل مدیریت پیکربندی تست‌ها، بارگذاری سیم‌کارت‌ها و به‌روزرسانی بازه همگام‌سازی است.
- **/domain/usecase/settings/LoadSimCards.kt**: لیست سیم‌کارت‌های فعال دستگاه را با استفاده از SubscriptionManager واکشی کرده و به مدل SimInfo تبدیل می‌کند.

• **/domain/usecase/settings/TestConfig.kt**: شامل مجموعه‌ای از UseCase های مرتبط با پیکربندی تست‌ها:

- **SetSmsTestNumberUseCase**: تعیین شماره مقصد برای تست پیامک.
- **SetPingAddressUseCase**: تعیین آدرس مقصد برای تست پینگ.
- **SetDnsAddressUseCase**: تعیین دامنه برای تست DNS.
- **SetWebAddressUseCase**: تعیین URL برای تست وب.
- **SetSelectedSimUseCase**: انتخاب سیم‌کارت پیش‌فرض برای اجرای تست‌ها.
- **GetSelectedSimSubsIdUseCase** و **GetSelectedSimSlotIdUseCase**: واکنشی سیم‌کارت انتخاب‌شده.

• **/domain/usecase/settings/UpdateSyncInterval.kt**: دریافت یا تغییر بازه همگام‌سازی پس‌زمینه با استفاده از **DataSyncScheduler**.

• **/domain/model/SimInfo.kt**: برای مدیریت و نگهداری اطلاعات مربوط به سیم‌کارت‌های دستگاه نام اپراتور، شماره اسلات سیم‌کارت و آی‌دی اشتراک ذخیره می‌شوند.

## ۲.۶.۲ لایه داده (Data Layer)

• **/utils/TestConfigManager.kt**: مدیریت تمام تنظیمات کاربر شامل شماره تست پیامک، آدرس پینگ، دامنه DNS، URL تست وب و سیم‌کارت انتخاب‌شده. این داده‌ها در **SharedPreferences** ذخیره و بازیابی می‌شوند.

• **/utils/DataSyncScheduler.kt**: وظیفه ذخیره‌سازی و به‌روزرسانی بازه همگام‌سازی دوره‌ای و زمان‌بندی کارگر همگام‌سازی را بر عهده دارد.

## ۳.۶.۲ تزریق وابستگی (Dependency Injection)

• **SettingsUseCases**: از طریق **Hilt** به **SettingsViewModel** تزریق می‌شود و تمام UseCase های مرتبط با تنظیمات را در اختیار لایه ارائه قرار می‌دهد.



## ۴.۶.۲ لایه ارائه (Presentation Layer)

- `/presentation/settings/SettingsViewModel.kt`: مدیریت وضعیت تنظیمات شامل: لیست سیم‌کارت‌ها، سیم انتخاب‌شده، بازه همگام‌سازی و مقادیر پارامترهای تست. این View-Model با استفاده از SettingsUseCases داده‌ها را در SharedPreferences ذخیره یا بارگذاری می‌کند.
- `/presentation/settings/SettingsScreen.kt`: رابط کاربری صفحه تنظیمات که شامل بخش‌های انتخاب سیم‌کارت، تعیین بازه همگام‌سازی و پیکربندی تست‌ها است. این صفحه از کامپوننت‌های مختلف برای نمایش و ویرایش مقادیر استفاده می‌کند.
- `/presentation/settings/components/SimSelectionSection.kt`: رابط کاربری برای نمایش لیست سیم‌کارت‌ها و انتخاب سیم پیش‌فرض با استفاده از RadioButton.
- `/presentation/settings/components/SyncIntervalSection.kt`: رابط کاربری برای انتخاب بازه همگام‌سازی داده‌ها از میان مقادیر از پیش تعریف‌شده (۱۵ دقیقه تا ۲۴ ساعت).
- `/presentation/settings/components/TestConfigurationSection.kt`: رابط کاربری برای تعیین پارامترهای تست شامل شماره پیامک، آدرس پینگ، دامنه DNS و URL تست وب.

## ۷.۲ دریافت دسترسی و اجرا در پس‌زمینه

این بخش مسئول مدیریت دسترسی‌های حیاتی (مکان، پیامک، وضعیت تلفن، اعلان‌ها و غیره) و همچنین تضمین اجرای تست‌های شبکه در پس‌زمینه است. پیاده‌سازی این قابلیت در چند لایه مجزا انجام شده است.

## ۱.۷.۲ لایه دامنه (Domain Layer)

- `/domain/usecase/permission/PermissionUseCase.kt`: بررسی وضعیت تمام دسترسی‌های مورد نیاز با استفاده از توابعی نظیر `requiredPermissions()` و `permissionStatus()` و بازگرداندن آن‌ها به صورت `PermissionItemState`.

## ۲.۷.۲ لایه داده (Data Layer)

- `/utils/permission/PermissionManager.kt`: مدیریت لیست دسترسی‌های پایه و ویژه (بر اساس نسخه اندروید)، شامل دسترسی مکان، وضعیت تلفن، پیامک، اعلان‌ها و دسترسی‌های خاص مانند `Schedule Exact Alarm` و `Battery Optimization`.
- `/utils/permission/PermissionUtility.kt`: تعریف مدل `AppPermission` به همراه توضیحات و مسیر هدایت به تنظیمات سیستم برای هر دسترسی. این کلاس وظیفه ساختن `Intent`‌های لازم برای تغییر تنظیمات مانند «غیرفعال‌سازی بهینه‌سازی باتری» یا «فعال‌سازی آلارم دقیق» را برعهده دارد.
- `/utils/LocationUtility.kt`: بررسی فعال بودن سرویس مکان‌یابی (`GPS` یا `Network Provider`) پیش از اجرای تست‌ها.

## ۳.۷.۲ لایه ارائه (Presentation Layer)

- `/presentation/permission/PermissionsViewModel.kt`: مدیریت وضعیت دسترسی‌ها و ذخیره آن‌ها به صورت `StateFlow`. همچنین مسئول نمایش یا پنهان‌سازی دیالوگ راهنما برای هر دسترسی است.
- `/presentation/permission/PermissionScreen.kt`: رابط کاربری برای نمایش لیست دسترسی‌ها به کاربر. در این صفحه وضعیت هر دسترسی نمایش داده می‌شود و کاربر می‌تواند برای فعال‌سازی به تنظیمات سیستم هدایت گردد.
- `PermissionItemCard.kt`: کارت نمایشی برای هر دسترسی به همراه وضعیت (فعال یا غیرفعال) و دکمه درخواست دسترسی.
- `PermissionRationaleDialog.kt`: دیالوگ راهنما برای توضیح اهمیت هر دسترسی و ارائه لینک به تنظیمات دستگاه.
- `PermissionsContent.kt`: لیست کلی دسترسی‌ها همراه با وضعیت آن‌ها و تعداد دسترسی‌های فعال شده.
- `PermissionTopBar.kt`: نوار بالای صفحه با امکان بازگشت به صفحه قبل.

## ۴.۷.۲ سرویس‌ها و اجرای پس‌زمینه

• `/service/TestExecutionService.kt`: یک `Foreground Service` که مسئول اجرای تست‌های شبکه در پس‌زمینه است. این سرویس:

- پیش از اجرا وضعیت تمام دسترسی‌ها و فعال بودن مکان‌یابی را بررسی می‌کند.
- تست‌ها را بر اساس تنظیمات کاربر (`TestConfigManager`) اجرا و نتایج را ذخیره می‌کند.
- در هنگام اجرا یک اعلان (`Notification`) نمایش می‌دهد تا سیستم از متوقف کردن آن جلوگیری کند.
- پس از پایان تست یا بروز خطا سرویس متوقف می‌شود.

## ۵.۷.۲ مدیریت در سطح فعالیت اصلی

• `/MainActivity.kt`: نقطه شروع بررسی دسترسی‌ها. در این فایل:

- دسترسی‌های اولیه در زمان شروع بررسی و در صورت نیاز از کاربر درخواست می‌شوند.
- پس از تأیید، مراحل تکمیلی شامل درخواست دسترسی مکان پس‌زمینه و اعلان‌ها انجام می‌شود.
- وضعیت بهینه‌سازی باتری و مجوز آلارم دقیق بررسی می‌شود.
- وظایف دوره‌ای از طریق `DataSyncScheduler` و `TestAlarmScheduler` زمان‌بندی می‌شوند.
- در نهایت فعال بودن سرویس مکان‌یابی کنترل و محتوای اصلی برنامه بارگذاری می‌گردد.

## ۸.۲ چالش‌ها و محدودیت‌ها

برخی از پارامترهای قدیمی مانند `RAC` و `Ec/No` به عنوان اطلاعات غیرضروری یا مربوط به نسل‌های منسوخ در نظر گرفته می‌شوند و در نتیجه در سطح کاربر عمومی (`TelephonyManager`) ارائه نمی‌شوند. تنها در سطح رابط‌های داخلی (مانند `ITelephony`) یا ابزارهای مهندسی و با دسترسی روت ممکن است این داده‌ها قابل مشاهده باشند، فلذا مشاهده می‌شود که `RAC` برابر `null` و مقدار `Ec/No` برابر 0 بازگردانده می‌شود.

• `RAC (Routing Area Code)`: این شناسه تنها در شبکه‌های نسل دوم کاربرد دارد. اما در

نسخه‌های جدید اندروید، مقدار RAC به صورت عمومی در API ارائه نمی‌شود و تنها مقدار LAC قابل دسترسی هستند. بنابراین حتی در حالت اتصال به 2G مقدار RAC همیشه null خواهد بود.

- **Ec/No**: این پارامتر کیفیت سیگنال مختص شبکه‌های نسل سوم است. با این حال در بسیاری از گوشی‌ها، مودم‌های پایه (Qualcomm MediaTek) این مقدار را به چارچوب اندروید گزارش نمی‌دهند و متد `getEcNo()` صرفاً مقدار پیش‌فرض 0 را بازمی‌گرداند. بنابراین حتی در حالت اتصال به 3G نیز مقدار واقعی Ec/No قابل دسترسی نیست.

## فصل ۳

# بک‌اند

### ۱.۳ ساختار و جزئیات Polaris Server

این بخش شامل پیاده سازی API های سمت سرور، مدل های داده، مکانیزم احراز هویت و زیرساخت لازم برای اجرای برنامه است. Backend با استفاده از چارچوب Django و Framework REST Django نوشته شده و وظیفه دریافت گزارش های اندازه گیری از اپلیکیشن های موبایل، ذخیره سازی آن ها در دیتابیس، ارائه API های مدیریتی و تأمین امنیت با JWT را بر عهده دارد.

### ۲.۳ فهرست اجزاء

- اپلیکیشن ها: users و reports mobile.
- تنظیمات پروژه: پکیج polaris شامل settings.py، urls.py و swagger.py.
- زیرساخت کانتینری: Dockerfile در پوشه backend و فایل docker compose.yml در ریشه پروژه.
- وابستگی ها: requirements.txt.

## ۳.۳ فایل پیکربندی و وابستگی ها

### ۱.۳.۳ requirements.txt

فهرست بسته های پایتون مورد نیاز شامل: Django، djangorestframework، django-drf-yasg، restframework، simplejwt، mysqlclient، environ، re-pillow، django-quests، cors headers و غیره.

### ۲.۳.۳ تنظیمات اصلی (polaris/settings.py)

تنظیمات پروژه موارد زیر را مشخص می کند:

- استفاده از مدل کاربر سفارشی: `MODEL USER AUTH = 'users.User'`.
- پایگاه داده: استفاده از MySQL با `django.db.backends.mysql` ENGINE و مقادیر اتصال از طریق متغیرهای محیطی خوانده می شوند شامل `DB NAME`، `DB USER`، `PASSWORD`، `DB HOST`، `DB PORT`.
- تنظیمات JWT با استفاده از `django-rest-framework-simplejwt` و دوره عمر توکن ها.
- فعال بودن CORS در محیط توسعه و مسیرهای `static` و `media` تعریف شده اند.

## ۴.۳ مدل های داده

در بخش بک اند چهار مدل اصلی تعریف شده اند: `Device`، `Measurement`، `TestResult` و `Users`.

### ۱.۴.۳ User

برای دسترسی به هریک از قابلیت ها و API ها نیاز به داشتن یک اکانت است. اطلاعات هر یوزر در این مدل ذخیره می شود.

ویژگی های مهم:

- `number phone` شماره تلفن کاربر که معمولاً یکتا است
- `username` در هنگام ثبت نام به صورت رندوم ثبت می شود اما قابل تغییر توسط کاربر است
- `email` ایمیل کاربر که اجباری و معمولاً یکتا است

- image عکس پروفایل کاربر
- is، verification verified، at expire code، مقادیری برای کنترل اعتبار اطلاعات داده شده
- banned is یک مقدار Boolean که نشان می دهد کاربر بن شده یا خیر
- access admin allow مشخص می کند آیا یک ادمین اجازه دارد به داده های measurement این کاربر دسترسی داشته باشد یا نه

### ۲.۴.۳ Device

ویژگی های مهم:

- id device کلید اصلی به صورت CharField
- version os model، manufacturer، مشخصات سخت افزاری و نرم افزاری دستگاه
- active is یک Boolean برای فعال بودن دستگاه
- timestamp seen last at، created، های خودکار
- ارتباط با کاربر: (user = ForeignKey, User, related name='devices')

این مدل برای ثبت دستگاه های کاربر و نگهداری آخرین وضعیت آن ها طراحی شده است  
نکته: این مدل به دلیل تغییر ساختار کنونی API ها استفاده نمیشود

### ۳.۴.۳ Measurement

این مدل گزارش های اندازه گیری شبکه را ذخیره می کند و فیلدهای زیر را دارد:

- موقعیت جغرافیایی: longitude latitude
- زمان اندازه گیری: timestamp
- نوع شبکه: type network مانند GSM، ۵G LTE و غیره
- شناسه های سلولی: lac، cell tac، id plmn
- پارامترهای رادیویی: frequency frequency، arfcn، band، rscp، rsrp، rxLev ecIo
- پارامترهای عملکردی شبکه: http upload، http download، ping، dns time، web response، time delivery sms response،

- at created خودکار و ایندکس ها: ایندکس روی (user, timestamp) و (latitude, longitude) برای بهینه سازی کوئری های زمانی و مکانی

در صورت نبود هر یک از موارد مقدار آن را null قرار می دهیم.

### ۴.۴.۳ TestResult

این مدل برای ذخیره نتایج تست های جداگانه طراحی شده است و فیلدهای اصلی عبارتند از:

- timestamp test type با مجموعه مقادیر مشخص، value مقدار عددی نتیجه، Boolean success
- info additional از نوع JSONField برای نگهداری متادیتای دلخواه
- ایندکس و together unique روی (user, timestamp) در مایگریشن ها تعریف شده است نکته: این مدل ممکن است بسته به نسخه API در همه جریان ها استفاده نشود

## ۵.۳ سریالایزرها

برای هر مدل سریالایزرها متناظر تعریف شده اند تا تبدیل داده ها بین مدل و JSON انجام شود:

- Register, Login Serializer, Register و سایر سریالایزرها برای ورود و ثبت نام و اعتبارسنجی داده ها
- Device Serializer شامل فیلدهای readonly مانند at, created, seen last
- Measurement Serializer و TestResult Serializer شامل فیلدهای مدل با readonly برای at created و user
- Bulk سریالایزرها برای دریافت و پردازش دسته ای measurement ها و تست ها
- Bulk Delete Serializer برای حذف گروهی با یک لیست از شناسه ها

## ۶.۳ اپلیکیشن users

### ۱.۶.۳ معرفی و نقش

اپلیکیشن users مسئول مدیریت حساب ها، احراز هویت و تمامی عملیات مرتبط با کاربران شامل ثبت نام ورود و بازیابی رمز است. این اپلیکیشن از مدل کاربر سفارشی استفاده می کند و با JWT یکپارچه شده است.



### ۲.۶.۳ مدل ها

مدل کاربر سفارشی دارای خصوصیات زیر است:

- فیلدهای شناسایی و تماس: image, username, email, number, phone
- فیلدهای وضعیت و کنترل: is, is active, is staff, is superuser, is verified, allow banned, access admin
- فیلدهای زمانی و امنیتی: date joined, last login, last seen, password
- مدل های کمکی اختیاری: PhoneOTP یا VerificationCode برای ذخیره کدها با فیلد code, expires

ارتباط با سایر مدل ها از طریق ForeignKey یا name related برقرار می شود تا داده های وابسته به هر کاربر قابل واکشی باشند.

### ۳.۶.۳ لیست کامل Endpoint ها

در ادامه فهرست تمام endpoint های مهم در اپ users آمده است.

#### ثبت نام Register

- مسیر: /api/users/register/
- متد: POST
- دسترسی: AllowAny
- رفتار: اگر ورودی معتبر باشد، کاربر ایجاد می شود و is verified برابر false خواهد بود و کد تأیید ارسال می شود

#### تأیید Verify OTP

- مسیر: /api/users/verify/
- متد: POST
- دسترسی: AllowAny
- رفتار: چک انقضا و در صورت تطابق فیلد is verified برای کاربر برابر true قرار می گیرد

**درخواست مجدد Resend OTP**

- مسیر: `/api/users/resend/`
- متد: POST
- دسترسی: AllowAny

**ورود Login**

- مسیر: `/api/users/login/`
- متد: POST
- دسترسی: AllowAny

**تجدید توکن Refresh Token**

- مسیر: `/api/users/token/refresh/`
- متد: POST
- دسترسی: AllowAny

**پروفایل کاربر ProfileView**

- مسیر: `/api/users/profile/`
- متدها: GET, PATCH, PUT
- دسترسی: IsNotBanned IsAuthenticated

**تغییر رمز Password Change**

- مسیر: `/api/users/change_password/`
- متد: POST
- دسترسی: IsNotBanned IsAuthenticated

## مدیریت کاربران Admin

مسیرها:

- GET /api/admin/all\_users/ برای لیست کاربران
- GET /api/admin/user\_info/id/ گرفتن اطلاعات یک کاربر
- POST /api/admin/ban\_user/ برای بن کردن
- POST /api/admin/allow\_user/ برای رفع بن
- POST /api/admin/make\_admin/ برای ایجاد ادمین با دادن رمز
- GET /api/admin/all\_data/ برای گرفتن تمام measurement های کاربرانی که اجازه گرفتن داده های آنها هست

## ۴.۶.۳ نحوه پیاده سازی و نکات فنی

- تمام endpoint ها با ViewSet یا های APIView DRF پیاده سازی شده اند و از Routerها برای ثبت مسیرها در urls.py استفاده می شود.
- سریالایزرها ورودی ها را اعتبارسنجی می کنند و فیلدهای readonly در صورت نیاز تعریف می شوند.
- ارسال OTP در حالت پایه با فراخوانی سرویس پیامک یا ایمیل انجام می شود؛ در محیط تولید پیشنهاد می شود ارسال به صورت غیرهمزمان و صفی با ابزارهایی مانند Celery انجام شود.

## ۷.۳ ویوها و API ها

API ها با استفاده از ViewSet ها و Router های DRF ساخته شده اند.

## ۱.۷.۳ مسیرها URLs

روتر اصلی اپلیکیشن mobile مسیرهای delete، bulk upload، bulk result، test measurement را ثبت می کند و از DefaultRouter برای ثبت روترها استفاده شده است.

## ۲.۷.۳ ViewSet Measurement

ویژگی ها:

- متدهای پیش فرض: Destroy List، Retrieve، Create، permission classes = [ IsNotBanned IsAuthenticated ] نیاز به JWT و بررسی بن بودن کاربر
- هنگام ایجاد مقدار user از request.user ست می شود
- اکشن های سفارشی: latest، all برای admin و get network types

### ۳.۷.۳ ViewSet TestResult

ساختار مشابه Measurement است و اکشن latest برای گرفتن آخرین نتیجه دارد.

### ۴.۷.۳ ViewSet BulkUpload

دارای اکشن های POST برای measurement و report test است که داده ورودی را با سریالایزهای bulk اعتبارسنجی کرده و رکوردها را ایجاد می کند و تعداد ایجاد شده را برمی گرداند.

### ۵.۷.۳ ViewSet BulkDelete

دریافت لیستی از شناسه ها و حذف رکوردهای مرتبط با کاربر جاری.

### ۶.۷.۳ ViewSet HTTPTest

یک ViewSet سبک برای تست عملکرد دانلود و آپلود:

- download متد GET که بایت تصادفی به اندازه (۲\*۱۹) randbytes را برمی گرداند
- upload متد POST که با کد ۲۰۴ پاسخ می دهد

## ۸.۳ احراز هویت و مجوزها

- JWT با simplejwt.djangorestframework پیکربندی شده و هدر Bearer Authorization پشتیبانی می شود
- Permission سفارشی IsNotBanned باید در ویوهای محافظت شده به کار رود
- برای عملیات مدیریتی از IsAdminUser یا بررسی request.user.is\_staff استفاده می شود

## ۹.۳ زیرساخت و دیپلوی

### ۱.۹.۳ Dockerfile

تصویر پایه از python ۱۳.۳ استفاده می‌کند. در ادامه برنامه بسته‌های سیستمی لازم برای اتصال به MySQL و ابزارهای کمکی نصب می‌شوند، requirements.txt داخل ایمج نصب شده و entrypoint.sh اجرایی می‌شود. پورت ۹۰۰۰ برای Django در نظر گرفته شده است.

### ۲.۹.۳ entrypoint.sh

در هنگام شروع کانتینر:

۱. انتظار برای آماده شدن سرویس MySQL با بررسی اتصال به آدرس database و پورت ۳۳۰۶
۲. اجرای migrations با migrate manage.py python
۳. جمع‌آوری فایل‌های static با static collectstatic --noinput
۴. اجرای سرور Django بر روی پورت ۹۰۰۰

### ۳.۹.۳ پایگاه داده

پیکربندی پیش فرض MySQL است و مقادیر اتصال از فایل env خوانده می‌شوند. در زمان دیپلوی لازم است سرویس MySQL توسط compose docker یا سرویس DB جداگانه مهیا شود.

### ۴.۹.۳ compose.yml docker

این فایل اجرای کانتینرها را مدیریت می‌کند. برای دیپلوی کردن پروژه کافی است به دایرکتوری ریشه پروژه رفته و دستور docker compose up -d --build را اجرا کنید. در زمان اجرا، ابتدا کانتینر دیتابیس بالا می‌آید و پس از اطمینان از سلامت آن، سرویس backend اجرا می‌شود.

### ۵.۹.۳ فایل env

مقادیر حساس و ثابت مانند SECRET KEY، اطلاعات دیتابیس، توکن سرویس پیامک و رمز ادمین در فایل env ذخیره می‌شوند و نباید در کنترل نسخه قرار گیرند.

### ۶.۹.۳ استفاده از nginx

برای ارائه سرویس بدون نیاز به وارد کردن پورت از proxy reverse مانند nginx استفاده شده است. nginx وظیفه مسیر دهی درخواست ها به فرانت و بک اند، مدیریت SSL و دامنه ها را بر عهده دارد. تبدیل http به https توسط certbot و دریافت گواهی SSL انجام می شود.

### ۷.۹.۳ بکاپ گیری

به دلیل اهمیت داده ها باید اسکرپتی جهت بکاپ گیری از دیتابیس MySQL وجود داشته باشد و آن را به صورت دوره ای با crontab یا ابزار مشابه اجرا کرد. در این پروژه از یک اسکرپت ساده استفاده شده بود که sqldump را سر ساعت ۱۲ هر روز در فایل backup.sql قرار میداد در صورت نیاز به لود کردن دیتا بیس فقط لازم است تا sqldump را با داده های بکاپ گیری شده جایگزاری کنیم

## ۱۰.۳ ابزارها و توابع کمکی

### ۱.۱۰.۳ reports mobile در utils.py

یک تابع کمکی مهم وجود دارد که با نام arfcn to frequency ورودی arfcn و نوع شبکه را گرفته و فرکانس متناظر را باز می گرداند. پیاده سازی شامل شاخه بندی برای GSM، LTE UMTS و ۵G است و در صورت نامشخص بودن مقدار None برمی گرداند.

### ۲.۱۰.۳ utils.py users

تابعی برای ارسال OTP با استفاده از سرویس خارجی پیاده سازی شده است. تابع OTP send ساختار درخواست را می سازد و با requests.post به سرویس ارسال می کند. روش تأیید می تواند از طریق SMS یا Email انتخاب شود که از طریق VERIFICATION METHOD در فایل env. مشخص می شود. در محیط توسعه معمولاً ایمیل قابل اطمینان تر است.

## ۱۱.۳ ملاحظات امنیتی و عملکردی

- نگهداری مقادیر حساس در فایل env. مانند SECRET KEY و اطلاعات دیتابیس

- در محیط تولید CORS باید محدود شود و فقط دامنه های مجاز مجوز داشته باشند
- اعمال محدودیت نرخ برای endpoint های حساس مانند ارسال OTP و ورود

## ۱۲.۳ دستورالعمل اجرای محلی

۱. فایل env. را در شاخه والد backend بسازید و مقادیر مورد نیاز شامل DB NAME, DB USER, DB PASSWORD, DB HOST, DB PORT, SECRET KEY, SMS API TOKEN, ADMIN PASSWORD, ALLOWED HOSTS, VERIFICATION METHOD و غیره را قرار دهید.

۲. با دستور docker compose up -d --build پروژه را اجرا کنید تا کانتینرها ساخته و راه اندازی شوند.

## فصل ۴

# فرانت‌اند

### ۱.۴ ساختار و جزئیات Web Application

این بخش به بررسی ساختار و اجزای اصلی برنامه وب می‌پردازد. برنامه با استفاده از کتابخانه React و ابزار Vite توسعه داده شده و از معماری ماژولار برای جداسازی مسئولیت‌ها و بهبود نگهداشت‌پذیری استفاده می‌کند.

#### ۱.۱.۴ فایل‌ها و پیکربندی سطح بالا

در ریشه پروژه، تعدادی فایل پیکربندی و مستندات وجود دارد:

- `package.json` و `package-lock.json`: مدیریت وابستگی‌ها و اسکریپت‌های پروژه.
- `vite.config.js`: تنظیمات مربوط به ابزار ساخت Vite.
- `eslint.config.js`: پیکربندی استانداردهای کدنویسی و بررسی استاتیک کد.
- `index.html`: نقطه ورود HTML برنامه.
- `README.md`: توضیحات و راهنمای پروژه.
- `Dockerfile`: پیکربندی ساخت و اجرای برنامه در محیط Docker.



### ۲.۱.۴ پوشه public

این پوشه شامل منابع استاتیک مانند تصاویر و آیکون‌ها است که بدون پردازش توسط Vite در دسترس خواهند بود:

- logo.svg، logo\_icon.svg: لوگوهای برنامه.
- phone.jpeg و phone\_dark.jpeg: تصاویر پیش‌نمایش برنامه در حالت روشن و تاریک.
- content.svg: عناصر گرافیکی برای بخش‌های مختلف.

### ۳.۱.۴ پوشه src

پوشه src محل اصلی کدهای برنامه است و شامل چند بخش مهم می‌شود:

**router.jsx، main.jsx، App.jsx**

- App.jsx: کامپوننت اصلی که ساختار کلی برنامه را تعریف می‌کند.
- main.jsx: نقطه شروع React که برنامه را در عنصر اصلی HTML بارگذاری می‌کند.
- router.jsx: مسیرها و صفحات مختلف برنامه را با استفاده از کتابخانه ناوبری تعریف می‌کند.

### پوشه context

این بخش شامل فایل Authorization.jsx است که Context مربوط به وضعیت احراز هویت و مجوزهای کاربر را فراهم می‌کند.

### پوشه hooks

هوک‌های سفارشی برای مدیریت داده‌ها و منطق برنامه:

- useDashboardData.js: مدیریت و دریافت داده‌های داشبورد.
- useUserList.js: مدیریت لیست کاربران و داده‌های مربوطه.

### پوشه managers

این بخش وظایف مدیریتی و ارتباط با API را برعهده دارد:

- ApiManager.js: مدیریت درخواست‌های HTTP.
- CookieManager.js: مدیریت کوکی‌ها.
- LoginManager.js, SignUpManager.js, ResetPasswordManager.js, VerifyManager.js: مدیریت عملیات مربوط به احراز هویت.
- DashboardManager.js, MapManager.js, UserListManager.js: مدیریت داده‌های بخش‌های مختلف برنامه.
- Constants.js: تعریف مقادیر ثابت مورد استفاده.

#### پوشه pages

شامل صفحات مختلف رابط کاربری است. هر صفحه ممکن است پوشه components مخصوص به خود را برای اجزای کوچکتر UI داشته باشد:

- Dashboard: نمایش داشبورد، نمودارها، فیلترها و جدول داده‌ها.
- Landing: صفحه اصلی معرفی برنامه شامل بخش‌هایی مانند ویژگی‌ها، دانلودها.
- Login, SignUp, Verify, ResetPassword: صفحات مربوط به فرآیند ورود و ثبت‌نام.
- Map: نمایش نقشه و شاخص‌های سیگنال.
- UserList: نمایش اطلاعات کاربران برای ادمین.
- NotFound: صفحه خطای ۴۰۴.

#### پوشه utils

ابزارهای کمکی که در بخش‌های مختلف استفاده می‌شوند:

- DatetimeUtility.js و FormatDatetime.js: قالب‌بندی و پردازش تاریخ و زمان.
- MapUtils.js: توابع کمکی برای کار با نقشه.
- ThemeManager.js: مدیریت تم و حالت روشن و تاریک برنامه.

## ۲.۴ احراز هویت

در این بخش، به تشریح معماری و پیاده‌سازی سیستم احراز هویت در پروژه می‌پردازیم. این سیستم بر اساس ساختار ماژولار و تفکیک وظایف (separation of concerns) طراحی شده است.

### ۱.۲.۴ مدیریت وضعیت (State Management)

- `src/context/Authorization.jsx`: این فایل هسته‌ی اصلی مدیریت وضعیت احراز هویت در کل برنامه است و با استفاده از `React Context API` پیاده‌سازی شده است. وظایف اصلی آن شامل موارد زیر است:

- نگهداری وضعیت ورود کاربر (`isAuthenticated`)، وضعیت مدیر بودن (`isAdmin`) و وضعیت بارگذاری (`isLoading`).
- مقداردهی اولیه‌ی وضعیت با استفاده از داده‌های ذخیره‌شده در کوکی‌ها توسط `CookieManager`.
- ارائه‌ی متد `setAuthentication` برای فعال‌کردن وضعیت احراز هویت پس از ورود موفق کاربر.
- ارائه‌ی متد `resetAuthentication` برای بازنشانی وضعیت احراز هویت در هنگام خروج یا منقضی‌شدن توکن.
- استفاده از `useCallback` برای جلوگیری از ایجاد توابع جدید در هر بار رندر و بهبود کارایی.

### ۲.۲.۴ مسیریابی محافظت‌شده (Protected Routing)

- در فایل `src/context/Authorization.jsx` همچنین یک کامپوننت به نام `ProtectedRoute` پیاده‌سازی شده است که وظیفه‌ی کنترل دسترسی به مسیرهای حساس را بر عهده دارد. این کامپوننت:

- در صورت فعال بودن حالت `isLoading`، یک پیام بارگذاری به کاربر نمایش می‌دهد.
- اگر مسیر `adminOnly` باشد، تنها زمانی اجازه دسترسی می‌دهد که کاربر هم مدیر باشد و هم احراز هویت شده باشد.
- در سایر مسیرهای محافظت‌شده، تنها کافی است کاربر احراز هویت شده باشد.
- در صورت عدم احراز هویت، کاربر را به مسیر ورود (`/login`) هدایت می‌کند.

### ۳.۲.۴ راه‌اندازی سراسری (Global Setup)

• `src/App.jsx`: این فایل نقطه‌ی ورود اصلی رابط کاربری است که در آن ارائه‌دهنده‌های سراسری (Global Providers) مقداردهی می‌شوند:

- `ThemeProvider` برای مدیریت تم روشن و تاریک با استفاده از ترجیحات سیستم.
- `ToastContainer` برای نمایش اعلان‌ها به کاربر.
- `QueryClientProvider` برای مدیریت درخواست‌های داده با `@tanstack/react-query`.
- `AuthProvider` که از `Authorization.jsx` وارد شده و مسئولیت ارائه‌ی وضعیت احراز هویت به تمام بخش‌های برنامه را دارد.
- `RouterProvider` برای بارگذاری مسیرها طبق پیکربندی `router.jsx`.

### ۴.۲.۴ مدیریت منطق تجاری (Business Logic)

• `src/managers/`: این دایرکتوری شامل ماژول‌هایی است که منطق تجاری احراز هویت و ارتباط با API را از رابط کاربری جدا می‌کنند تا کد ساختارمند و قابل نگهداری باشد.

- `ApiManager.js`: یک لایه انتزاعی برای برقراری ارتباط با API که با استفاده از کتابخانه `axios` پیاده‌سازی شده است. این ماژول توکن احراز هویت را از `CookieManager` بارگذاری کرده و به هدر درخواست‌ها اضافه می‌کند. همچنین در صورت دریافت خطای ۴۰۱، کاربر را به صفحه ورود هدایت می‌کند.
- `Constants.js`: شامل ثابت‌های سراسری مانند آدرس پایه API، دامنه اصلی و کلید ذخیره‌سازی کوکی است تا وابستگی به مسیرها و مقادیر ثابت متمرکز و قابل تغییر باشد.
- `CookieManager.js`: مسئول ذخیره‌سازی، بازیابی و حذف امن داده‌های احراز هویت (توکن و وضعیت مدیر بودن) در کوکی‌های مرورگر است. این ماژول از کتابخانه `js-cookie` برای مدیریت کوکی‌ها استفاده می‌کند.
- `LoginManager.js`: منطق ورود کاربر را پیاده‌سازی کرده و درخواست را به نقطه پایانی مشخص ارسال می‌کند. در صورت ورود موفق، اطلاعات دریافتی را در کوکی ذخیره کرده و خطاهای احتمالی (مانند عدم تأیید ایمیل) را مدیریت می‌کند.
- `SignUpManager.js`: منطق ثبت‌نام کاربر جدید را از طریق ارسال درخواست به نقطه پایانی

معین مدیریت می‌کند. خطاهای برگشتی از API را پردازش کرده و به لایه رابط کاربری ارسال می‌کند.

– **ResetPasswordManager.js**: مسئول مدیریت فرآیند بازیابی رمز عبور شامل سه مرحله است: ارسال کد تأیید، بررسی صحت کد و تنظیم رمز عبور جدید. تمامی این مراحل با نقاط پایانی مشخص API انجام می‌شود.

– **VerifyManager.js**: فرآیند تأیید هویت کاربر را با ارسال کد تأیید و اطلاعات کاربری به API انجام می‌دهد. در صورت موفقیت، داده‌های کاربر در کوکی ذخیره می‌شوند تا وضعیت احراز هویت در سراسر برنامه فعال شود.

## ۵.۲.۴ لایه رابط کاربری (UI Layer)

• **src/pages/**: این دایرکتوری شامل کامپوننت‌های صفحه‌ای است که با کاربر تعامل مستقیم دارند و داده‌ها را از کاربر دریافت یا به او نمایش می‌دهند.

– **Login/index.jsx**: این کامپوننت صفحه ورود کاربر را پیاده‌سازی می‌کند و شامل یک فرم با اعتبارسنجی سمت کاربر با استفاده از کتابخانه‌های react-hook-form و zod است. فیلد number\_or\_email بررسی می‌کند که ورودی یک ایمیل معتبر یا شماره تلفن با فرمت صحیح باشد و رمز عبور حداقل ۸ کاراکتر باشد. پس از ارسال فرم، داده‌ها به متد login از LoginManager.js ارسال می‌شوند تا فرآیند احراز هویت سمت سرور انجام شود. در صورت موفقیت، پیام موفقیت با استفاده از react-toastify نمایش داده شده و کاربر به داشبورد هدایت می‌شود. در صورت بروز خطای ۴۰۱، کاربر به صفحه تأیید حساب (/verify) منتقل می‌شود و اطلاعات ورودش همراه کد تأیید ارسال می‌شود. این صفحه از Material-UI برای طراحی رابط کاربری واکنش‌گرا استفاده می‌کند و قابلیت نمایش/مخفی‌سازی رمز عبور را فراهم کرده است. همچنین لینک‌های ناوبری برای بازیابی رمز عبور (/reset-password) و ثبت‌نام (/sign-up) در صفحه قرار داده شده‌اند تا تجربه کاربری کامل‌تری ارائه شود.

– **SignUp/index.jsx**: این کامپوننت مسئول نمایش و مدیریت فرم ثبت‌نام کاربر است. از کتابخانه react-hook-form همراه با zod برای اعتبارسنجی داده‌ها استفاده می‌کند. اسکیمای signUpSchema شامل اعتبارسنجی ایمیل، شماره تلفن، رمز عبور و تأیید رمز عبور است، و از ویژگی refine برای اطمینان از تطابق رمز عبور و تکرار آن بهره می‌برد. برای مدیریت وضعیت نمایش/عدم نمایش رمز عبور و تأیید آن، دو state محلی تعریف شده است. در تابع onSubmit، داده‌های معتبر از طریق SignUpManager.signUp به سرور ارسال شده و

در صورت موفقیت، کاربر به صفحه تأیید هدایت می‌شود و پیام موفقیت با `react-toastify` نمایش داده می‌شود. در صورت خطا، پیام مناسب خطا به کاربر نشان داده می‌شود. این فرم با استفاده از کامپوننت‌های `Material-UI` طراحی شده و شامل ورودی‌های شماره تلفن، ایمیل، رمز عبور، تأیید رمز عبور، و لینک تغییر مسیر به صفحه ورود است. نمایش بصری حالت بارگذاری نیز در دکمه ثبت‌نام پیاده‌سازی شده است.

– `ResetPassword/index.jsx`: کامپوننت مربوط به فرآیند بازیابی رمز عبور که در سه مرحله پیاده‌سازی شده است.

۱. مرحله ۱ (دریافت ایمیل یا شماره موبایل): با استفاده از `react-hook-form` و `zod` ورودی کاربر اعتبارسنجی شده و با متد `ResetPasswordManager.sendResetCode` کد تأیید ارسال می‌شود.

۲. مرحله ۲ (تأیید کد): کاربر یک کد ۵ رقمی را وارد کرده و این کد با متد `ResetPasswordManager.verifyResetCode` بررسی می‌شود. امکان ارسال مجدد کد با محدودیت زمانی نیز فراهم شده است.

۳. مرحله ۳ (ایجاد رمز عبور جدید): کاربر رمز عبور جدید را وارد کرده و تکرار آن نیز بررسی می‌شود. اعتبارسنجی با `zod` انجام و داده‌ها با متد `ResetPasswordManager.resetPassword` به سرور ارسال می‌شوند.

۴. مدیریت وضعیت: با استفاده از `useState` و `useForm` وضعیت‌هایی مانند مرحله فعلی، بارگذاری (`loading`)، شمارش معکوس و نمایش/مخفی کردن رمز مدیریت می‌شود.

۵. رابط کاربری: با کتابخانه `Material-UI` طراحی شده و از کامپوننت‌هایی مانند `TextField`، `Button` و `CircularProgress` استفاده شده است.

۶. بازخورد به کاربر: پیام‌های موفقیت یا خطا با کتابخانه `react-toastify` نمایش داده می‌شوند.

۷. ناوبری: پس از موفقیت در تغییر رمز عبور، کاربر به صفحه ورود هدایت می‌شود. برای جابه‌جایی بین مراحل نیز از `useNavigate` استفاده شده است.

– `Verify/index.jsx`: صفحه‌ای برای تأیید هویت کاربر که پس از ثبت‌نام یا ورود نیازمند تأیید ایمیل یا شماره است. این صفحه با استفاده از کتابخانه‌های `React Hook Form` و `Zod` اعتبارسنجی کد پنج‌رقمی را انجام می‌دهد. داده‌های کاربر مانند ایمیل و رمز عبور از `location.state` دریافت می‌شوند. تابع `onSubmit` درخواست تأیید را به `VerifyManager`

ارسال کرده و در صورت موفقیت، کاربر را وارد سیستم می‌کند. امکان ارسال مجدد کد با شمارش معکوس نیز پیاده‌سازی شده است تا از ارسال مکرر جلوگیری شود. طراحی و استایل صفحه با Material UI و تم پروژه هماهنگ شده و از CircularProgress برای نمایش وضعیت بارگذاری استفاده می‌شود. همچنین از Toastify برای نمایش پیام‌های موفقیت یا خطا بهره گرفته شده است. لینک بازگشت به صفحه ورود نیز در انتها قرار داده شده تا کاربر بتواند مسیر خود را تغییر دهد.

## ۳.۴ داشبورد

### ۱.۳.۴ مدیریت منطق تجاری (Business Logic)

- `managers/DashboardManager.js`: این ماژول وظیفه مدیریت منطق مربوط به دریافت داده‌های داشبورد را بر عهده دارد. در این فایل، درخواست‌های لازم برای دریافت اطلاعات اندازه‌گیری‌های موبایل به نقطه پایانی API ارسال می‌شود. در صورت موفقیت، داده‌ها به صورت مستقیم برگردانده می‌شوند و در صورت خطا، پیام خطا به صورت دقیق پردازش و بازگردانده می‌شود. این ماژول لایه‌ای میان رابط کاربری و API است که به جداسازی مسئولیت‌ها کمک کرده و مدیریت خطاها را به صورت متمرکز انجام می‌دهد. به طور خلاصه، این فایل یک نقطه ورود ساده و کارا برای واکنشی داده‌های خام داشبورد فراهم می‌کند.

### ۲.۳.۴ هوک‌های سفارشی (Custom Hooks)

- `hooks/useDashboardData.js`: یک هوک سفارشی در React است که مدیریت کامل داده‌ها و حالت‌های مرتبط با داشبورد را بر عهده دارد. این هوک ابتدا با استفاده از React Query، داده‌های اندازه‌گیری را از `DashBoardManager` واکنشی می‌کند و وضعیت بارگذاری و خطا را کنترل می‌نماید. سپس داده‌ها را بر اساس فیلترهای کاربر مانند نوع شبکه و بازه زمانی به صورت پویا فیلتر می‌کند. علاوه بر این، این هوک مجموعه‌ای از گزینه‌های پیکربندی شده برای انواع نمودارهای مختلف را با توجه به داده‌های فیلتر شده و تم رابط کاربری تولید می‌کند. استفاده از این هوک باعث جداسازی کامل منطق داده و آماده‌سازی آن برای نمایش در کامپوننت‌ها شده و کد رابط کاربری را ساده‌تر و قابل نگهداری‌تر می‌کند. همچنین بازخوانی داده‌ها به صورت خودکار و با فاصله زمانی مشخص انجام می‌شود تا داشبورد همواره به‌روز باشد. در نهایت، این هوک توابع و متغیرهای مورد نیاز برای فیلترینگ و تغییر حالت داده‌ها را نیز ارائه می‌دهد که به راحتی در کامپوننت‌های مختلف قابل استفاده است.

### ۳.۳.۴ لایه رابط کاربری (UI Layer)

- **pages/UserLayout/index.jsx**: این کامپوننت اصلی لایه کاربری (User Layout) برنامه را پیاده‌سازی می‌کند و ساختار کلی صفحات را با یک نوار کناری (Sidebar) و نوار بالایی (AppBar) فراهم می‌آورد. این لایه به صورت واکنش‌گرا طراحی شده و در دستگاه‌های موبایل، منوی کشویی بازشو دارد، در حالی که در صفحه‌نمایش‌های بزرگ‌تر یک منوی ثابت با قابلیت جمع‌شدن (Collapse) نمایش داده می‌شود. منوی کناری شامل آیتم‌های ناوبری مانند داشبورد، نقشه، لیست کاربران (در صورت مدیر بودن کاربر) و گزینه خروج است. کاربر می‌تواند با کلیک روی هر آیتم به صفحه مربوطه هدایت شود و گزینه خروج، کوکی‌های احراز هویت را حذف کرده و کاربر را به صفحه ورود می‌برد. طراحی این کامپوننت با استفاده از کتابخانه Material-UI انجام شده و از امکاناتی مانند AppBar، Drawer، آیکون‌ها و واکنش‌پذیری بهره می‌برد. همچنین وضعیت باز یا بسته بودن منو و حالت جمع‌شده در state محلی نگهداری می‌شود و با تغییر اندازه صفحه نمایش رفتار منو نیز تغییر می‌کند. ناوبری از طریق هوک useNavigate انجام می‌شود و لایه احراز هویت با استفاده از context مدیریت می‌گردد. این کامپوننت همچنین فضای لازم برای بارگذاری صفحات فرزند را با Outlet فراهم می‌کند تا بتوان صفحات مختلف را به صورت درون‌خطی نمایش داد. به طور کلی، این فایل مسئول ایجاد ساختار پایه‌ای و یکپارچه برای صفحات مختلف کاربری با قابلیت ناوبری و مدیریت دسترسی است.
- **pages/Dashboard/index.jsx**: این کامپوننت صفحه اصلی داشبورد شبکه را پیاده‌سازی می‌کند و شامل نمایش داده‌ها، فیلترها و نمودارهای متنوع است. با استفاده از هوک سفارشی useDashboardData داده‌ها واکشی و فیلتر شده و وضعیت بارگذاری کنترل می‌شود. داشبورد شامل چندین تب با عناوین مختلف است که هر تب داده‌ها و نمودارهای خاص خود را نمایش می‌دهد، مانند Overview، SMS، Measurements، Web و سایر موارد. در تب Overview نمودارهای کلی تکنولوژی شبکه، توزیع ARFCN، باند فرکانسی، پراکندگی RSRP و RSRQ و جدول داده‌ها ارائه می‌شود. در تب‌های تخصصی‌تر مانند SMS یا Web، نمودارهای خطی، جعبه‌ای (BoxPlot) و توزیع داده‌ها همراه با جدول آمارهای عددی نمایش داده می‌شوند. این کامپوننت برای نمایش واکنش‌گرا طراحی شده و در دستگاه‌های موبایل تنظیمات فاصله و چینش متفاوت دارد. مدیریت وضعیت تب فعال و رندر نمودارها با استفاده از state و conditional rendering انجام می‌شود. علاوه بر نمودارها، جدول داده‌ها با قابلیت پاسخگویی به تغییر تب‌ها، به‌روزرسانی می‌شود تا داده‌ها را به صورت جزئی‌تر نمایش دهد. این ساختار باعث می‌شود داشبورد به صورت جامع و کاربرپسند اطلاعات شبکه را ارائه دهد و تحلیل‌های متنوعی را امکان‌پذیر سازد.
- **pages/Dashboard/components/ColumnConfig.js**: این فایل پیکربندی ستون‌های جدول



داشبورد را تعریف می‌کند. هر ستون شامل برچسب (label) تابع رندر برای نمایش داده‌ها و سبک ظاهری اختصاصی است. برای مثال ستون زمان (timestamp) با استفاده از تابع `formatDateTime` فرمت شده و به صورت خوانا نمایش داده می‌شود. سایر ستون‌ها مانند موقعیت جغرافیایی، نوع شبکه، باند فرکانسی، ARFCN و اطلاعات سلولی نیز به صورت مناسب فرمت و رندر می‌شوند. این ساختار به جداسازی منطق نمایش داده‌ها در جدول کمک کرده و قابلیت گسترش و نگهداری را بالا می‌برد. مقادیر نال یا نامشخص با استفاده از “-” نمایش داده می‌شوند تا جدول مرتب و قابل فهم باقی بماند. ستون‌هایی مانند سرعت دانلود و آپلود، پینگ و پاسخ DNS با دقت عددی مشخص شده نمایش داده می‌شوند. به طور کلی، این فایل قالب‌بندی و نمایش داده‌های خام را برای جدول داشبورد استاندارد می‌کند.

• **pages/Dashboard/components/DashboardCharts.jsx**: این فایل مجموعه‌ای از کامپوننت‌های

نمودار را برای داشبورد تعریف می‌کند که با استفاده از `echarts-for-react` پیاده‌سازی شده‌اند. هر نمودار درون یک `ChartContainer` قرار گرفته که قابلیت نمایش تمام صفحه (Fullscreen) را با استفاده از `Dialog` فراهم می‌کند. کامپوننت‌های مختلفی مانند نمودار تکنولوژی شبکه، توزیع ARFCN، نمودار باند فرکانسی، پراکندگی RSRP/RSRQ و نمودارهای سیگنال به مرور زمان در این فایل موجود است. این کامپوننت‌ها به صورت واکنش‌گرا طراحی شده و ارتفاع آن‌ها در موبایل و دسکتاپ متفاوت تنظیم می‌شود. استفاده از `useTheme` باعث می‌شود تم نمودارها با حالت کلی رابط کاربری هماهنگ باشد (حالت تاریک یا روشن). آیکن Fullscreen برای هر نمودار اجازه می‌دهد تا نمودارها به صورت بزرگ و واضح‌تر مشاهده شوند. این ساختار باعث یکپارچگی در نمایش نمودارها و تجربه کاربری بهتر در داشبورد می‌شود. همچنین کامپوننت‌های نمودار مشابه، با رابط یکسان قابل استفاده مجدد هستند و کد را تمیز و منظم نگه می‌دارند.

• **pages/Dashboard/components/DashboardConfig.js**: این فایل شامل توابعی برای

ساخت گزینه‌ها (option) و تنظیمات نمودارهای مختلف داشبورد است که با کتابخانه `ECharts` ساخته می‌شوند. برای هر نوع نمودار مانند نمودار دایره‌ای شبکه‌های مخابراتی (Network Tech)، توزیع ARFCN، نمودار میله‌ای باند فرکانسی و نمودار پراکندگی RSRP و RSRQ، تنظیمات مربوط تعریف شده است. توابع داده‌های ورودی را پردازش کرده و شمارش یا فیلترهای لازم را انجام می‌دهند تا داده‌ها به فرمت مناسب نمودار تبدیل شوند. مثلاً در نمودار دایره‌ای نوع شبکه، مواردی که مقدارشان “UNKNOWN” است نادیده گرفته می‌شوند. نمودارهای خطی مانند قدرت سیگنال در طول زمان، تعداد اندازه‌گیری‌ها بر اساس ساعت، و نمودارهای باکس‌پلات و توزیع داده‌ها نیز با محاسبات آماری دقیق آماده می‌شوند. تابع `percentile` برای محاسبه درصدک‌ها در نمودار باکس‌پلات استفاده می‌شود تا بازه‌های آماری مانند چارک‌ها و رنج بین چارکی محاسبه شود. تنظیمات محورهای

X و Y، عنوان‌ها، رنگ‌بندی‌ها و استایل‌ها با توجه به تم کلی برنامه و پارامترهای ورودی تنظیم می‌شوند. این فایل با جداسازی منطق ساخت گزینه‌های نمودار، به مدیریت بهتر و قابلیت نگهداری کد کمک می‌کند. همچنین انعطاف‌پذیری برای اضافه کردن نمودارهای جدید یا تغییر ظاهر آن‌ها را ساده‌تر می‌سازد. به طور کلی، این کامپوننت هسته تولید تنظیمات نمودارهای داشبورد است که باعث یکپارچگی و سازگاری نمایی می‌شود.

- **pages/Dashboard/components/DashboardFilters.jsx**: این کامپوننت مسئول نمایش فیلترهای داشبورد است که به کاربر امکان انتخاب نوع شبکه و بازه زمانی داده‌ها را می‌دهد. از TextField‌های MUI برای انتخاب نوع شبکه و تاریخ شروع و پایان استفاده شده است که در حالت موبایل به صورت واکنش‌گرا تنظیم می‌شوند. لیست انواع شبکه شامل گزینه‌های مختلف مانند GSM، LTE و غیره است که کاربر می‌تواند به دلخواه فیلتر کند. برای فیلدهای تاریخ، استایل‌های سفارشی اعمال شده تا ظاهر دکمه انتخاب تاریخ (calendar picker) در حالت تاریک و روشن متناسب باشد. تغییرات در فیلدهای تاریخ با استفاده از تابع کمکی LocalizeDateTime به فرمت محلی تبدیل و در state والد ذخیره می‌شوند. کد به صورت واکنش‌گرا نوشته شده و بسته به اندازه صفحه، اندازه فیلدها و چینش آن‌ها تغییر می‌کند. این کامپوننت بدون حالت داخلی (stateless) است و تمام داده‌ها و مدیریت حالت از والد به صورت props دریافت می‌شود. استفاده از flexWrap و فاصله‌گذاری (gap) بین فیلدها باعث می‌شود چینش در موبایل و دسکتاپ مرتب و کاربرپسند باشد. این فیلترها نقش مهمی در محدود کردن داده‌های نمایش داده شده در نمودارها و جداول داشبورد دارند. در نهایت، این کامپوننت رابط کاربری ساده، تمیز و قابل گسترش برای فیلترکردن داده‌ها فراهم می‌کند.

- **pages/Dashboard/components/DashboardTable.jsx**: این کامپوننت جدول داده‌های اندازه‌گیری شده را نمایش می‌دهد و از قابلیت صفحه‌بندی (pagination) پشتیبانی می‌کند. ستون‌های جدول بر اساس تب فعال انتخاب می‌شوند و تعداد ردیف‌های نمایش داده شده در هر صفحه در موبایل و دسکتاپ متفاوت است. امکان تغییر صفحه و تعداد ردیف‌ها توسط کاربر فراهم شده و به‌روزرسانی داده‌ها با توجه به این تغییرات انجام می‌شود. دو دکمه برای صادر کردن داده‌ها به فرمت CSV و KML وجود دارد؛ CSV برای داده‌های جدولی و KML برای داده‌های جغرافیایی با مختصات مناسب. در فرآیند صادرات KML، فقط داده‌هایی که مختصات معتبر دارند در فایل قرار می‌گیرند و توضیحات مربوط به هر ردیف به صورت HTML داخل فایل درج می‌شود. از توابع کمکی برای قالب‌بندی داده‌ها و نمایش سفارشی هر ستون استفاده شده است. نمایش جدول با استفاده از کامپوننت‌های MUI انجام می‌شود تا تجربه کاربری بهتری در دستگاه‌های مختلف فراهم شود. امکان پیمایش افقی جدول نیز برای نمایش بهتر در صفحه‌های کوچک وجود دارد. این کامپوننت نقش مهمی در مدیریت، مشاهده و استخراج داده‌های اندازه‌گیری شده در داشبورد ایفا می‌کند. تمام منطق مرتبط با نمایش داده‌ها،

صفحه‌بندی و صادرات در این کامپوننت به صورت کامل پیاده‌سازی شده است.

- **pages/Dashboard/components/TabConfig.js**: این فایل تنظیمات ستون‌های نمایش داده شده در هر تب داشبورد را مشخص می‌کند. هر تب شامل لیستی از کلیدهای ستون‌ها است که تعیین می‌کند کدام داده‌ها در آن تب نمایش داده شوند. برای تب Overview همه ستون‌ها و برای تب‌های دیگر ستون‌های مرتبط با نوع داده خاص انتخاب شده‌اند.
- **pages/Dashboard/components/ValueStatisticTable.jsx**: این کامپوننت یک جدول آماری برای آرایه‌ای از مقادیر عددی نمایش می‌دهد. با استفاده از کتابخانه simple-statistics، شاخص‌هایی مانند میانگین، میانه، واریانس و انحراف معیار محاسبه می‌شود. همچنین بازه‌های اطمینان 68.2٪، 95٪ و 99.7٪ بر اساس انحراف معیار ارائه می‌شود. جدول نتیجه به شکل مرتب و با قالب‌بندی دو رقم اعشار نمایش داده می‌شود. در صورت نبود داده، کامپوننت چیزی رندر نمی‌کند. این کامپوننت برای نمایش خلاصه آماری داده‌ها و تحلیل سریع کاربرد دارد.

#### ۴.۳.۴ ابزارهای کمکی (Utility Functions)

- **utils/DatetimeUtility.js**: این فایل شامل دو تابع است: تابع `formatDateTime` برای قالب‌بندی تاریخ و زمان به صورت رشته‌ای با فرمت `YYYY-MM-DD HH:mm:ss`. تابع `Localize-Datetime` برای تبدیل تاریخ به زمان محلی با تنظیم اختلاف زمانی مشخص (کم کردن ۲۱۰ دقیقه).
- **utils/FormatDatetime.js**: این فایل تنها شامل تابع `formatDateTime` است که همانند نسخه قبلی، تاریخ و زمان را به فرمت `YYYY-MM-DD HH:mm:ss` تبدیل می‌کند. در این نسخه، تابع برای قالب‌بندی تاریخ بدون تغییر منطقه زمانی استفاده می‌شود.

#### ۴.۴ نقشه

##### ۱.۴.۴ مرور کلی

بخش Map مسئول نمایش اندازه‌گیری‌های میدانی (Measurement) بر روی یک نقشه تعاملی است. این بخش از کتابخانه `react-leaflet` برای رندر نقشه استفاده می‌کند. منطق واکنشی و عملیات مرتبط با API در یک ماژول مجزا (`MapManager`) قرار دارد و اجزاء نمایش‌دهنده، پیکربندی‌ها و توابع کمکی در پوشه `pages/Map` و `utils` پیاده‌سازی شده‌اند. هدف اصلی این بخش ارائه نمایش بصری داده‌های اندازه‌گیری با امکان تنظیم پویا توسط کاربر و حفظ پاسخ‌دهی مناسب است.

## ۲.۴.۴ ساختار کلی و نقش هر بخش

موارد اصلی و نقش هر یک:

- **MapManager**: لایه API است که مسئول همه تعاملها با سرور شامل واکنشی داده‌ها، فیلتر سمت سرور (در صورت وجود پارامترها) و عملیات حذف می‌باشد. MapManager نقطه متمرکزی برای مدیریت خطاها، افزودن هدر احراز هویت و یکسان‌سازی پاسخها فراهم می‌آورد؛ بنابراین بقیه اجزاء نیازی به دانستن جزئیات پروتکل شبکه ندارند و تنها با داده نرمال‌شده کار می‌کنند.
- **صفحه Map**: صفحه سطح بالا که مسئول ترکیب اجزاء است؛ ایجاد MapContainer، نگهداری stateهای محلی (مثل متریک انتخاب‌شده، پیکربندی رنگ‌ها و آستانه‌ها)، نمایش دیالوگ تنظیمات و مدیریت legend بر عهده این صفحه است. این صفحه همچنین نقش هماهنگ‌کننده را دارد: داده‌ها را از MapManager می‌گیرد، آن‌ها را به شکل موردنیاز برای نمایش آماده می‌کند و به نشانگرها تحویل می‌دهد.
- **SignalStrengthMarker**: جزئیات نمایش هر Measurement را بر عهده دارد — تعیین رنگ و آیکون مناسب، نمایش Popup با اطلاعات تکمیلی و پاسخ به تعاملات کاربر (کلیک، hover و ...) در این کامپوننت انجام می‌شود. این جز بهتر است تنها مسئول نمایش باشد و منطق پیچیده تبدیل داده را از لایه‌های دیگر دریافت کند.
- **default\_config**: مجموعه پیش‌فرض تنظیمات شامل دسته‌بندی متریک‌ها، واحدها، آستانه‌های اولیه و طیف رنگی پیشنهادی. این فایل مرجع اولیه برای هر تنظیم است و صفحه Map مقادیر فعلی را بر اساس آن مقداردهی می‌کند.
- **MapUtils**: توابع کمکی غیرمستقیم (مثل تعیین بازه مجاز اسلایدرها، اندازه گام تغییرات و محاسبه رنگ متن متضاد) که به یک نمایش سازگار و قابل‌پیش‌بینی کمک می‌کنند.

## ۳.۴.۴ نحوه واکنشی و مدیریت داده‌ها

واکنشگر اصلی واکنشی معمولاً از react-query و هوک useQuery استفاده می‌کند تا:

- کش مناسب و آستانه تازه‌سازی خودکار فراهم شود،
- خطاها و حالت‌های بارگذاری به صورت متمرکز مدیریت شوند،
- سیاست‌هایی مثل focus window on refetch یا polling در صورت نیاز فعال یا غیرفعال شوند.

در سطح منطقی، جریان داده چنین است: صفحه Map از MapManager.getMeasurements(allData)

برای گرفتن لیستِ Measurement ها استفاده می‌کند، پاسخ را بررسی و نرمال‌سازی می‌نماید و سپس به کامپوننت‌های نمایش‌دهنده پاس می‌دهد. MapManager مسئول افزودنِ هدرِ احراز هویت (مثلاً از طریق CookieManager یا ApiManager)، و بازگرداندنِ خطاهای معنادار به شکل پیام‌هایی برای نمایشِ کاربر است.

#### ۴.۴.۴ رندر نقشه و کنترل‌ها

MapContainer و TileLayer عنصرِ محوریِ رندرِ نقشه است و MapContainer منبعِ کاشی‌های نقشه (مثل OpenStreetMap) را فراهم می‌کند. مرجعِ نقشه در mapRef ذخیره می‌شود تا عملیات‌هایی مانند تغییرِ مرکز، بزرگ‌نمایی یا فراخوانیِ توابعِ کتابخانه به‌آسانی انجام شود. مرکزِ نقشه معمولاً بر اساسِ اولین Measurement یا انتخابِ کاربر مقداردهی می‌شود تا نمایشِ اولیه معنادار باشد.

کنترلِ مرکزِ نقشه برای حرکتِ نرم و هماهنگِ نقشه هنگامِ انتخابِ یک رکورد، یک کنترل‌کننده مبتنی بر هوکِ useMap تغییراتِ center را شنود می‌کند و از متدِ flyTo یا معادلِ آن برای انتقالِ روانِ نما استفاده می‌کند. این رفتار باعث می‌شود که کاربر هنگامِ کلیک بر روی لیستی از رکوردها یا جستجو تجربه دیداریِ پیوسته‌ای داشته باشد.

دیاالوگِ تنظیمات و legend دیاالوگِ تنظیمات نقطه ورودِ کاربر به سفارشی‌سازیِ نحوه نمایش است: انتخابِ دستهِ متریک (metricType)، انتخابِ متریکِ دقیق، تنظیمِ آستانه‌ها و ویرایشِ طیفِ رنگی. legend به‌عنوان راهنمای بصری در صفحه قرار می‌گیرد و بازخوردِ آنی از تنظیماتِ کاربر نشان می‌دهد؛ بنابراین کاربر بلافاصله تأثیرِ تغییرات را بر روی نشانگرها می‌بیند.

#### ۵.۴.۴ منطق تعیین رنگ و نمایش نشانگرها

هدفِ اصلیِ منطقِ رنگ‌دهی این است که مقدارِ عددیِ هر Measurement را به یک بازنمای بصریِ خوانا و معنی‌دار تبدیل کند. به‌طور مفهومی:

۱. انتخابِ مقدارِ معنادار: برای هر متریک، از میانِ فیلدهای موجود مقدارِ قابل‌اعتماد و مرتبطِ انتخاب می‌شود (مثلاً برای signal\_strength ممکن است چند فیلدِ جایگزین وجود داشته باشد).
۲. نرمال‌سازی نسبت به آستانه‌ها: مقدارِ خام نسبت به آستانه‌های max/mid/min سنجیده و به یک مقیاسِ معمولی (مثلاً 0 تا 1) تبدیل می‌شود تا منطقِ رنگ‌دهی برای همهٔ متریک‌ها قابلِ مقایسه شود.
۳. نگاشتِ عددی به طیفِ رنگی: نسبتِ نرمال‌شده روی طیفی بین رنگِ «ضعیف» تا «عالی» نگاشت

می‌شود. برای ایجاد تغییر نرم از یک رنگ به رنگ بعدی از یک مکانیزم میان‌یابی (interpolation) استفاده می‌شود تا مرزها مصنوعی و سخت نباشند.

۴. حالت‌های معکوس: برای متریک‌هایی که مقدار کمتر بهتر است (مثلاً latency) پیش از نرمال‌سازی معکوس‌سازی انجام می‌شود تا قانون یکپارچه رنگ‌دهی حفظ شود.

۵. مقادیر sentinel و داده ناموجود: مقادیر مخصوص (مثل 1- یا null) به‌عنوان داده ناموجود شناسایی شده و با یک رنگ/آیکون مشخص یا پنهان‌سازی هوشمند نمایش داده می‌شوند تا کاربر گمراه نشود.

#### ۶.۴.۴ توابع کمکی کلیدی

توابع کمکی جهت پشتیبانی از تجربه کاربری قابل‌تنظیم و سازگار وجود دارند؛ بدون آوردن پیاده‌سازی دقیق، شرح عملکرد هر کدام چنین است:

- `getMinRange`: مشخص می‌کند کمترین مقدار منطقی برای هر متریک چه قدر است—این مقدار پایه حدِ چپ اسلایدر آستانه است.
- `getMaxRange`: مشخص می‌کند بیشترین مقدار منطقی برای هر متریک چه قدر است—محدوده سمت راست اسلایدر.
- `getStepSize`: تعیین‌کننده کوچک‌ترین واحد تغییر ممکن در اسلایدر است (مثلاً برای برخی متریک‌ها گام ۱۰۰ منطقی‌تر است و برای برخی گام ۱).
- `getContrastColor`: تابعی که به‌صورت مفهومی تعیین می‌کند متن روی یک رنگ پس‌زمینه بهتر است تیره باشد یا روشن تا خوانایی حفظ شود.

#### ۷.۴.۴ جریان داده

۱. صفحه Map درخواست داده را از طریق MapManager ارسال می‌کند.
۲. پاسخ سرور توسط لایه MapManager بررسی و (در صورت نیاز) نرمال‌سازی می‌شود.
۳. داده‌های نرمال‌شده به مجموعه نشانگرها داده می‌شوند تا هرکدام بر اساس پیکربندی فعلی رنگ و محتوای Popup خود را تعیین کنند.
۴. تغییرات پیکربندی (آستانه/رنگ/متریک) که توسط کاربر انجام می‌شود، بلافاصله به نمای نقشه منعکس می‌گردد و در state محلی صفحه نگهداری می‌شود.

## ۵.۴ مدیریت کاربران و سطح دسترسی

### ۱.۵.۴ مدیریت منطق تجاری (Business Logic)

- `managers/UserListManager.js`: این ماژول رابط کاربری برای مدیریت کاربران است. تابع `getAll` لیست کامل کاربران را از API دریافت می‌کند. توابع `allow` و `ban` برای مسدود و فعال‌سازی کاربران با ارسال اطلاعات شماره یا ایمیل به API استفاده می‌شوند. در صورت خطا، پیام مناسب به صورت رشته‌ی JSON یا متن خطا برگردانده می‌شود.

### ۲.۵.۴ هوک‌های سفارشی (Custom Hooks)

- `hooks/useUserList.js`: این هوک با استفاده از `react-query` عملیات واکشی، مسدودسازی و رفع مسدودسازی کاربران را مدیریت می‌کند. از `useQuery` برای دریافت لیست کاربران و از `useMutation` برای اجرای عملیات مسدودسازی و رفع مسدودسازی استفاده شده است. در هر عملیات، داده‌های کش‌شده به‌روزرسانی و در صورت خطا، تغییرات برگشت داده می‌شود. پیغام‌های موفقیت و خطا با استفاده از `react-toastify` نمایش داده می‌شوند.

### ۳.۵.۴ لایه رابط کاربری (UI Layer)

- `pages/UserList/index.jsx`: این کامپوننت لیست کاربران را نمایش می‌دهد و از هوک `useUserList` برای دریافت داده‌ها استفاده می‌کند. دارای قابلیت جستجو بر اساس ایمیل، شماره تلفن و نام کاربری است و نتیجه را فیلتر می‌کند. نمایش کاربران در جدول همراه با وضعیت تأیید و مسدود بودن آن‌ها با سوئیچ تعاملی برای فعال/مسدود کردن. دارای صفحه‌بندی با تعداد سطر قابل تنظیم است که برای موبایل بهینه شده است. در هنگام بارگذاری داده، نمایش انیمیشن لودینگ و نمایش پیام خطا در صورت عدم موفقیت دریافت داده‌ها. استفاده از قابلیت واکنش‌گرایی برای نمایش مناسب در موبایل با استفاده از `Media Query`.