Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра мережних технологій

Курсова робота

освітній ступінь – бакалавр

на тему імплементація механізму авторизації для Android застосунку “SMART UKMA”

Автор: _____
Калита Дарина Олександрівна
Освітня програма «Інженерія програмного забезпечення», 121

Керівник: _____
доктор технічних наук, доцент
Глибовець А.М.,_____

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК _____
«____» _____ 2023 р.

Київ – 2023

Зміст

Вступ

Android – широко використовувана мобільна операційна система, розроблена компанією Google. Ця операційна система легко налаштовується та відома своєю гнучкістю та можливостями кастомізації. Також ця операційна система сумісна з широким спектром пристроїв. Розробка додатків для Android вимагає спеціалізованих навичок та інструментів[1].

Розробка додатків для Android - це багатогранний і складний процес, який вимагає високого рівня навичок у роботі із різноманітними інструментами і ресурсами. Задля створення якнайбільш якісних продуктів необхідно добре продумувати та планувати процес розробки, а також вміло вибирати технології, які будуть використані. Зокрема важливим є вмілий вибір та імплементація корректного типу авторизації.

Несанкціонований доступ до конфіденційної інформації через відсутність сучасної та ефективної авторизації може призвести до значних негативних наслідків, включаючи такі, як втрата даних, порушення конфіденційності та фінансові і репутаційні збитки. Для запобігання несанкціонованому використанню чутливої інформації була розроблена концепція авторизації, метою якої є контроль доступу користувача до наявних даних та ресурсів.

Авторизація – процес перевірки особи користувача та надання або заборони доступу до ресурсів на основі заздалегідь визначеного набору правил або політик. Цей надзвичайно важливий механізм безпеки відіграє вирішальну роль у захисті комп'ютерних систем і додатків від несанкціонованого доступу та шахрайських атак.

Авторизація – складний процес, який містить декілька послідовних кроків. Вона може бути реалізована за допомогою декількох методів, що відрізняються за своєю суттю та кожен з яких є найбільш підходящим для різних типів продуктів.

Процес реалізації авторизації має бути ретельно продуманий, щоб забезпечити його ефективність та дієвість. Необхідно правильно вибирати спосіб імплементації заздалегідь, із урахуванням технічних особливостей та нагальних потреб продукту. Після створення вона повинна регулярно перевірятися та оновлюватися, щоб відображати зміни в системі та організації.

В Україні актуальність впровадження авторизації в Android-додатках є надзвичайно важливою через зростаючу цифрову економіку країни. IT-індустрія України постійно стрімко прогресує і велика кількість українських компаній надає послуги з розробки програмного забезпечення на міжнародному ринку. Український ринок є висококонкурентним, і клієнти очікують від мобільних додатків найвищого рівня безпеки та надійності,

оскільки це синонімічно із якістю. Український уряд також активно сприяє розвитку IT сектору, що призвело також до значного збільшення кількості і саме мобільних додатків, що розробляють та використовують в Україні. Із зростанням розповсюдженості їх використання спостерігається відповідний ріст частоти випадків кіберзлочинності, особливо у сферах крадіжки персональних даних, фінансового шахрайства та крадіжки даних.

Тому для бізнесів та розробників в Україні життєво необхідно впроваджувати належні заходи авторизації у своїх мобільних додатках, щоб захистити дані користувачів та запобігти несанкціонованому доступу до акаунтів. Впровадження належних заходів авторизації може допомогти бізнесу побудувати довіру з клієнтами та отримати значну перевагу над своїми суперниками.

Таким чином, впровадження авторизації в Android-додатках має важливе значення як у світі, так і в Україні для забезпечення безпеки та конфіденційності даних користувачів, запобігання кіберзлочинності та побудови довіри з клієнтами. В ході цієї роботи буде більш детально розглянуто приклад Android додатків на прикладі мобільного додатку Smart UKMA, а також приклад механізму авторизації на прикладі мобільного додатку Authorization Showcase, де її було імплементовано мною.

Загалом, основними завданнями є:

1) Розглянути операційну систему Android, процес розробки продуктів на базі цієї оперативної системи та технологій, що використовуються для цього. Проаналізувати додаток Smart UKMA.
2) Розглянути авторизацію як механізм загалом, надати опис її типів та методів реалізації. Проаналізувати власну імплементацію механізму авторизації через Microsoft акаунт.

Частина 1. Операційна система Android

Вперше операційна система Android була випущена в 2008 році і з тих пір стала однією з найпопулярніших операційних систем для смартфонів і планшетів у всьому світі. Архітектура Android побудована на ядрі Linux, що забезпечує стабільну та безпечну основу для цієї операційної системи. Android складається з набору стандартних додатків, включаючи годинник, календар, калькулятор, камеру, галерею та інші, які попередньо встановлені на пристроях. Крім того, платформа підтримує встановлення сторонніх додатків з Google Play Market, який має величезну колекцію додатків, ігор та іншого програмного забезпечення, доступпного для завантаження користувачами.

Ця операційна система відома своїм високим ступенем кастомізації та гнучкості, що дозволяє користувачам налаштовувати свій пристрій на свій смак. Наприклад, користувачі можуть налаштувати макет, кольори, шпалери, іконки та інші елементи користувацького інтерфейсу, щоб створити персоналізований досвід. Крім того, Android підтримує різні методи введення, такі як сенсорні екрани, клавіатури та голосові команди. Така багатоманістність дозволяє користувачам взаємодіяти зі своїми пристроями у спосіб, який їм найбільше підходить.

Android також здатна до підтримки широкого діапазону асортименті, в тому числі смартфони, планшети, смарт-годинники, телевізори та інші пристрої з різними технічними характеристиками та можливостями. Така різноманістність стала можливою завдяки модульній архітектурі платформи, яка дозволяє виробникам налаштовувати та модифікувати систему відповідно до своїх вимог до апаратного забезпечення та дизайну. У продукти, послуги чи системи, створені на Android, можна зручно вносити коригування, модифікації та загалом зміни, щоб краще узгодити їх з індивідуальними або унікальними специфікаціями, релевантними до конкретної мети розробника.

Підхід Android з відкритим вихідним кодом сприяв його широкому розповсюдженню і сприяв його постійній еволюції та розвитку, що дозволило розробникам і користувачам використовувати його платформу для різних цілей. Android широко використовується на широкому діапазоні пристроях, включаючи смартфони, планшети, смарт-годинники та телевізори.

Така гнучкість – одна з рис, що значно допомогла Android стати однією з найпоширеніших операційних систем у світі. Тепер вона працює на мільйонах пристроїв і забезпечує розмаїту екосистему додатків і сервісів.

1.1. Огляд мов для розробки Android додатків

Розробка в середовищі Android вимагає знання різних програмних інструментів і технологій, які дозволяють створювати надійні та високопродуктивні додатки. Процес розробки Android-додатків включає кілька етапів, зокрема проектування інтнрфейсу користувача, реалізацію логіки програми, тестування та розгортання. Одним з найбільш важливих аспектів розробки Android є вибір правильних інструментів і технологій, оскільки це може суттєво вплинути на продуктивність, стабільність і масштабованість кінцевого продукту[2]

Надважливою особливвстю розробки продуктів під Android є використання таких мов програмування, як Java та Kotlin. Ці мови надають розробникам цілий ряд бібліотек та API, які дозволяють створювати високоякісні, ефективні та надійні додатки для Android.

Kotlin стала офіційною мовою для розробки під Android, і багато розробників зараз використовують Kotlin для створення додатків для Android. Це більш сучасна мова програмування, була представлена у 2017 році.[2][3]

Java - одна з найпоширеніших мов програмування для розробки Android. Це об'єктно-орієнтована мова високого рівня, яка створена для того, щоб бути простою, ефективною та незалежною від платформи та надає розробникам широкий спектр інструментів та бібліотек для створення складних додатків. Широке застосування Java у розробці Android можна пояснити її широкою підтримкою сторонніх бібліотек та сумісністю з різзними операційними системами.

Kotlin - ще одна популярна мова програмування, яка використовується для розробки Android. Це сучасна мова зі статичною типізацією, яка розроблена для того, щоб бути більш лаконічною, виразною та легшою для читання, ніж Java, і багато розробників вважають її більш продуктивною. Kotlin надає розробникам такі можливості, як нульова безпека, функції розширення та підпрограми, які допомагають зменшити багатослівність коду та підвищити продуктивність Android-додатків. Також Kotlin надає можливість перемикатися між нею із Java, що робить роботу із мовою ще зручнішою та легшою.[3][4]

Використання таких мов програмування, як Java та Kotlin, є ключовою особливістю розробки для Android. Розповсюдженість цих мов дозволяє розробникам створювати якісні, ефективні та надійні додатки для різноманітних мобільних пристроїв. Крім того, наявність декількох каналів

розповсюдження дозволяє розробникам охопити широку аудиторію та ефективно монетизувати свої додатки.

Окрім Java та Kotlin, розробка Android також передбачає роботу з XML для проектування макетів та іншими технологіями, такими як SQL для управління базами даних та JavaScript для створення веб-додатків для Android.[5]

XML використовується для визначення макету та дизайну користувацьких інтерфейсів Android. Ця мова розмітки забезпечує структурований спосіб визначення компонентів інтерфейсу користувача та їх властивостей. Файли XML можна створювати за допомогою візуального редактора макетів або безпосередньо редагуючи XML файли.

Розробникам Android також можуть знадобитися знання C++, Python та інших мов для низькорівневого системного програмування та навички оптимізації продуктивності додатків.

C++ - це високопродуктивна мова, яка широко використовується для розробки мобільних ігор та інших додатків, для яких критично важлива продуктивність. З іншого боку, Python - це динамічна і проста у вивченні мова, яка часто використовується для розробки моделей машинного навчання та інших додатків, орієнтованих на роботу з даними. [4]

1.2. Огляд середовищ та інструментів для розробки Android додатків

Під час розробки додатків, які базуються на операційній системі Android, використовується Android Software Development Kit (SDK).

Android SDK – це набір інструментів і ресурсів, які розробники використовують для створення додатків для Android. SDK вміщує інструменти для налагодження, тестування та профілювання додатків, а також такі ресурси, як приклади коду, документація та навчальні посібники, які полегшують розробку високоякісних додатків для Android. Одним з ключових компонентів SDK є Android Framework, який надає розробникам набір інтерфейсів прикладного програмування (API) для створення Android-додатків; також SDK містить Android Emulator і Android Debug Bridge (ADB).[6][7]

Android надає широкий спектр API, які розробники можуть використовувати для додавання функціональності до своїх додатків. Ці інтерфейси охоплюють широкий спектр тем, в тому числі такі, як дизайн інтерфейсу користувача, зберігання даних, роботу в мережі та інші. Деякі з найбільш часто використовуваних API включають в себе інструментарій Android User Interface (UI), який надає розробникам набір компонентів користувацького інтерфейса для створення додатків; Android Content Provider API, який дозволяє додаткам обмінюватися даними з іншими додатками[9]; і Android Location API, який дозволяє додаткам отримувати доступ до даних про місцезнаходження з пристрою.

Android Emulator – емулятор, який дозволяє розробникам тестувати свої програми на віртуальних пристроях Android.[8] Цей емулятор підтримує ряд пристроїв з різними технічними характеристиками і конфігураціями, що дозволяє розробникам моделювати різні сценарії, а також зручно виявляти і виправляти будь-які проблеми, що виникають у процесі розробки.

Android Debug Bridge (ADB) – незамінна утиліта командного рядка для відладки та тестування Android-додатків. Вона пропонує широкий набір інструментів, які дозволяють розробникам професійно встановлювати, налагоджувати і керувати Android-додатками як на віртуальних, так і на фізичних пристроях.[6] [7] На додаток до цих основних функцій, ADB також має можливість віддаленого відлагодження, що полегшує роботу із програмами, які працюють на дистанційному пристрої. Ця функція віддаленого доступу в поєднанні з арсеналом команд робить ADB інструментом для розробників, які прагнуть забезпечити безперебійне розгортання та виконання Android-додатків.[9]

Основним інтегрованим середовищем розробки (IDE), що використовує Android Software Development Kit є Android Studio. Цей редактор коду побудований на платформі IntelliJ IDEA та є офіційним середовищем для розробки Android продуктів.

Android Studio – потужне і багатофункціональне IDE, яке надає розробникам повний набір інструментів і ресурсів для створення високоякісних Android-додатків. Це IDE призначене для підтримки розробників протягом усього процесу розробки додатків, від створення проекту до деплоймента.[10] Це основне інтегроване середовище розробки (IDE) для розробки Android-додатків. Воно пропонує вичерпний набір інструментів, які дозволяють розробникам ефективно проектувати, розробляти та тестувати додатки для Android.[10][7] Наприклад, Android Studio відома своїм універсальним візуальним редактором макетів, який полегшує розробку користувацьких інтерфейсів для Android-додатків. Редактор пропонує можливість мануального пересування компонентів інтерфейсу, і ця функція значно спрощує процес створення складних макетів і полегшує побудову зручних інтерфейсів. Ця IDE також включає в себе інші функції. Наприклад, аналіз коду, налагодження та інструменти тестування, які дозволяють розробникам створювати високоякісні Android-додатки.[10]

Також підсвічування синтаксису, завершення коду та навігація по коду, які допомагають розробникам писати чистий та ефективний код, інструменти профілювання і підтримка різних сторонніх бібліотек і фреймворків, які надають розробникам широкі можливості для створення інноваційних і багатофункціональних Android-додатків. Крім того, налагоджувач IDE дозволяє розробникам виявляти і вирішувати проблеми в їхньому коді, забезпечуючи комплексний досвід налагодження.

Ці можливості в поєднанні з відкритим вихідним кодом Android зробили Android Studio популярним вибором для розробників, які прагнуть створювати високоякісні, надійні та масштабовані додатки для Android.

Хоча Android Studio є офіційним IDE для розробки Android, існує кілька популярних альтернатив, які розробники можуть використовувати для створення Android-додатків. Прикладами таких аналогів є Eclipse, IntelliJ IDEA, Visual Studio, Xamarin Studio та React Native.[11]

Eclipse – це IDE з відкритим вихідним кодом, що підтримує декілька мов програмування, в тому числі Java та C++. Вона широко використовувалася для розробки Android до виходу Android Studio, і деякі розробники використовують її й по цей день. [7]

IntelliJ IDEA – популярне середовище розробки Java, розроблене компанією JetBrains, тією ж компанією, яка створила Android Studio. Хоча Android Studio базується на IntelliJ IDEA, IntelliJ IDEA пропонує більше функціональних можливостей і підтримує додаткові мови програмування. [7][11]

Visual Studio від Microsoft – це потужне середовище розробки, яке підтримує низку мов програмування, зокрема Java та C++. Visual Studio надає інструменти для створення Android-додатків з використанням Xamarin, фреймворку призначеного для розробки крос-платформних додатків.

Xamarin Studio – це кросплатформенна IDE, яка дозволяє розробникам створювати додатки для Android, iOS та Windows Phone, використовуючи одну кодову базу. Вона надає ряд інструментів і ресурсів для розробки під Android, включаючи засоби налагодження і тестування. [7]

React Native – популярний фреймворк для створення мобільних додатків з використанням JavaScript та React. Так само, як і Xamarin Studio, він дозволяє розробникам створювати додатки для Android, використовуючи єдину кодову базу, а також надає інструменти для налагодження, тестування та профілювання додатків. [12]

Кожна з цих IDE має свої унікальні особливості та переваги, і розробники повинні вибрати ту, яка найкраще відповідає їхнім потребам та вподобанням. Однак варто зазначити, що Android Studio є найбільш поширеною та рекомендованою IDE для розробки під Android.

Існують також інші технології, які використовуються у процесі розробки під операційну систему Android. Серед них: Gradle та Firebase.

Gradle – це інструмент збірки за замовчуванням з відкритим вихідним кодом, який широко використовується у розробці та розгортання додатків для Android. Він надає комплексне рішення для управління всім процесом збірки, включаючи вибір варіанту збірки, а також управління залежностями, автоматичну компіляцію вихідного коду та пакування кінцевого додатку. Популярність Gradle зумовлена його ефективною та гнучкою системою конфігурації на основі скриптів, яка дозволяє розробникам легко конфігурувати та налаштовувати складні процеси збірки. Крім того, Gradle пропонує широку підтримку плагінів, що дозволяє розробникам розширити процес збірки за допомогою таких функцій, як обфускація коду та підписання кінцевого додатку. Використовуючи ці потужні можливості, розробники можуть спростити процес розробки та покращити загальну якість і продуктивність своїх Android-додатків.[13]

Firebase, надійна платформа для розробки мобільних та веб-додатків, пропонує розробникам широкий спектр послуг, що полегшують створення та

розгортання додатків. Її широкий набір функцій включає, але не обмежується автентифікацією, базою даних в режимі реального часу, хмарним сховищем та хмарним обміном повідомленнями. Завдяки бездоганній інтеграції Firebase в Android Studio, розробники можуть без особливих зусиль включати ці функції в свої додатки, спрощуючи процес розробки і розширюючи їх загальну функціональність.[14] Комплексний набір інструментів Firebase робить її цінним ресурсом для розробників, які прагнуть оптимізувати свої робочі процеси розробки та розгортання додатків[15]

У підсумку, процес розробки під Android вимагає ознайомленності з великою кількістю програмних інструментів і технологій, включаючи, але не обмежуючись такими, як: Android Studio, Java, Kotlin, Gradle, Android Debug Bridge і Firebase. Доведено, що використання цих інструментів і технологій значно полегшує процес розробки та сприяє створенню високопродуктивних, надійних і масштабованих додатків. Тим не менш, розробники повинні завжди бути в курсі останніх досягнень в екосистемі Android, щоб переконатися, що вони використовують найбільш оптимальні та найсучасніші інструменти для розробки своїх додатків. Знання цих технологій, що розвиваються, може підвищити продуктивність та ефективність робочого процесу розробки.[6][7][16][17][18]

1.3. Етапи розробки Android додатків

Розробка додатків для Android вимагає спеціалізованих навичок та інструментів. Розробники повинні мати глибоке розуміння мов програмування, які вони збираються використовувати, а також фреймворків та API, що будуть примінені під час роботи. Окрім цього необхідно володіти навичками дизайну користувацького інтерфейсу задля забезпечення позитивного досвіду для майбутніх користувачів.[19][20]

Процес створення додатку для Android складається з численних етапів, список яких включає в себе наступні кроки: збір вимог, аналіз та проектування, кодування, тестування та розповсюдження. Детальніше про ці етапи нижче:

1) Збір вимог: задля вдалого створення продукту необхідно перед початком роботи чітко та вичерпно визначити, якими є вимоги до нього. Це передбачає визначення потреб користувачів, мети програми та технічних вимог, які необхідно задовольнити. Кроки, які необхідно пройти задля успішного виконання цього етапу, зазвичай включають визначення мети та сфери застосування продукту, проведення користувацьких досліджень, визначення технічних вимог, планування користувацького інтерфейсу та створення користувацьких історій, тобто опису того, як користувачі будуть взаємодіяти з додатком. Ці описи можуть допомогти навігувати процесом розробки, забезпечуючи чітке розуміння потреб і цілей користувача.[20]
2) Аналіз та проектування: передбачає аналіз вимог, зібраних на попередньому етапі, з метою визначення можливості реалізації проекту, виявлення будь-яких ймовірних ризиків або обмежень, а також формування плану реалізації проекту. Нижче наведені деякі ключові процеси, які, як правило, відбуваються на етапі аналізу: техніко-економічне обґрунтування, тобто аналіз вимог і визначення того, чи є проект технічно здійсненним і фінансово вигідним, оцінка ризиків та розробка планів зменшення їх впливу, проектування архітектури системи, включаючи базу даних, інтерфейс користувача та інші компоненти, за допомогою їх візуальних репрезентації, створення попередньої версії додатку для тестування його функціональності та користувацького досвіду та розробка детального плану проекту, включаючи терміни виконання, бюджет і потреби в ресурсах.[19]
3) Написання коду: після пройдення етапу збора вимог, вичерпного аналізу на основі зібранної інформації а також ви можете почати писати код для

реалізації функціональності вашого додатку. Ви будете писати код на Java або Kotlin, залежно від ваших уподобань.

4) Тестування: протягом усього процесу розробки необхідно ретельно тестувати додаток, щоб переконатися, що він працює належним чином. Тестування можна проводити за допомогою емуляторі Android або на фізичному пристрої. Типами тестування можуть бути функціональне тестування, тестування продуктивності, юзабіліті-тестування та тестування безпеки. Для того, щоб гарантувати, що додаток відповідає найвищим стандартам якості, розробники повинні також проводити ретельне відлагодження.[19]
5) Налагодження та оптимізація: Під час тестування вашого додатку є шанс зіткнутися з помилками або проблемами продуктивності. Для того, щоб виявити і виправити ці проблеми, необхідними будуть інструменти налагодження Android, наприклад раніше згаданий ADB; Android Studio Debugger, який дозволяє розробникам відстежувати виконання додатку, встановлювати точки зупину, перевіряти значення змінних тощо; [20] Traceview, який дозволяє аналізувати, як програма використовує ресурси пристрою, ідентифікувати буття помилки в процесі виконання, а також відстежувати взаємодію між компонентами додатку; Android Monitor, який дозволяє розробникам переглядати інформацію про використання пам'яті, процесора та інших ресурсів; та Network Profiler, який дозволяє розробникам відстежувати відправку та отримання даних через мережу, а також аналізувати їх швидкість передачі.[19]
6) Публікація: Після того, як ваш додаток буде завершено і протестовано, ви можете опублікувати його в Google Play Store, який є найпопулярнішим каналом розповсюдження додатків для Android, або іншому магазині додатків, наприклад Amazon Appstore. Для цього необхідно створити список додатків, встановити ціну, якщо це можливо, і завантажити файл Android Package Kit (APK). Розробники також можуть поширювати свої додатки безпосередньо користувачам, надаючи їм файл APK, який можна встановити на їхні пристрої. [20][21]

1.4. Аналіз Android додатку “Smart UKMA”

Android додаток “Smart UKMA” написано мовою програмування Kotlin.

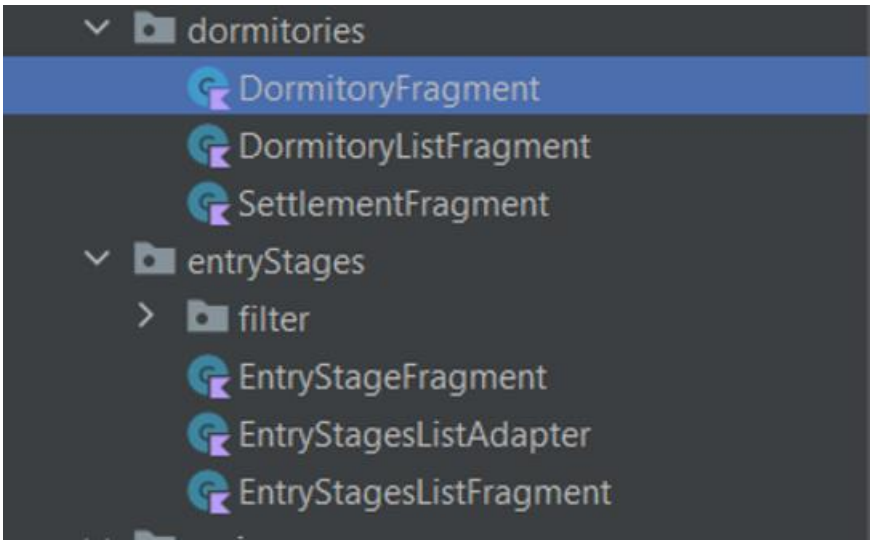

Рис. 1. Kotlin у додатку Smart UKMA

Ця мова більш сучасна та, як вже було сказано в цій роботі раніше, є офіційною мовою розробки додатків на базі операційної системи Android. Ця мова також має чистий і лаконічний синтаксис, що відчутно покращує досвід роботи із нею. Також, Kotlin було створено із особливою увагою до безпеки та надійності. У Kotlin передбачені вбудовані механізми захисту від нульових значень, які допомагають усунути жахливі помилки NullPointerException (NPE), що часто трапляються в Java. Система типів Kotlin розрізняє типи, які можна обнулити, та типи, які не можна обнулити, що зменшує ймовірність виникнення NPE та забезпечує більш надійний код. Ця мова також включає такі функції, як виведення типів, незмінність за замовчуванням та комплексні стандартні бібліотечні функції, і все це значно зменшує ризик помилок.
У проєкті використано стандартний для додатків на базі Android OS інструмент автоматизації збірки – Gradle.

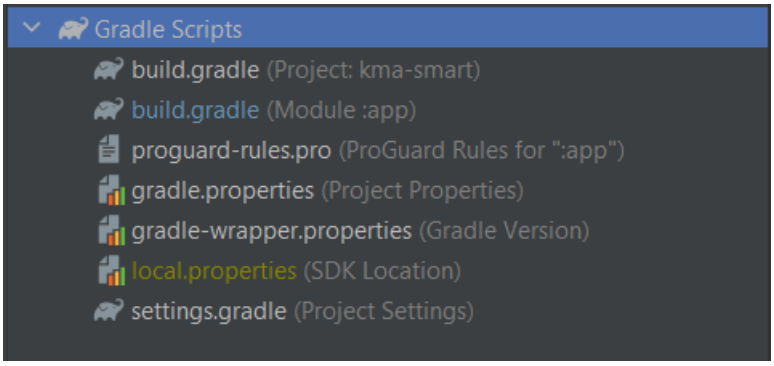

Рис 2. Gradle у додатку Smart UKMA

Gradle використовує Kotlin як мову сценаріїв для визначення сценаріїв збірки, якими описуються завдання, залежності та конфігурації, необхідні для збірки та пакування проекту. Gradle також вельми гнучкий і має можливість використовувати велику кількість різноманітних плагінів та залежностей, наприклад Android плагін, що дозволяє створювати та пакувати додатки для Android.

У даному проєкті використано багато плагінів, у тому числі плагін Firebase.

```
plugins {
    id 'com.android.application' version '7.2.0' apply false
    id 'com.android.library' version '7.2.0' apply false
    id 'org.jetbrains.kotlin.android' version '1.7.0' apply false
    id 'org.jetbrains.kotlin.plugin.parcelize' version '1.7.0' apply false
    id 'org.jetbrains.kotlin.kapt' version '1.7.0' apply false
    id 'com.google.gms.google-services' version '4.3.10' apply false
    id 'com.google.firebase.appdistribution' version '3.0.1' apply false
    id 'com.google.firebase.crashlytics' version '2.9.0' apply false
    id 'androidx.navigation.safeargs.kotlin' version '2.4.2' apply false
    id 'com.google.android.libraries.mapsplatform.secrets-gradle-plugin' version '2.0.1' apply false
}
```

Хмарна база даних NoSQL Firebase забезпечує синхронізацію даних між клієнтами в режимі реального часу. Вона використовує модель даних на основі JSON, що дозволяє створювати додатки без складної логіки для сервера.

Для створення графічної частини проєкту було використано мову маркування XML.

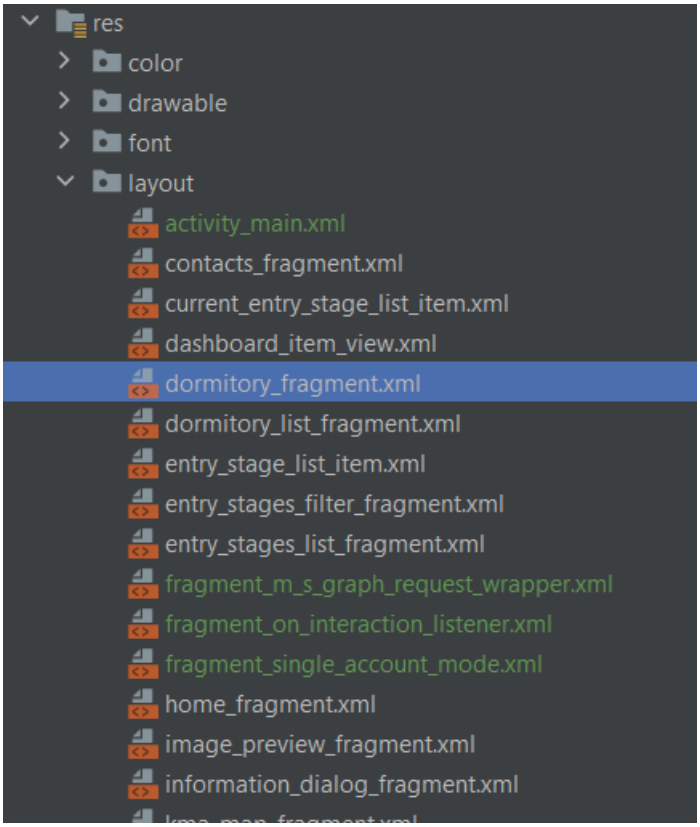

Рис. 3. XML у додатку Smart UKMA

XML темпелейти дуже прості у використанні, а сама мова легко читається, що робить написане нею простим для розуміння людини. Також XML сприяє чіткому розмежуванню між візуальною складовою та основною логікою додатку і дозволяє дуже зручно контролювати ресурси, як вже готові, так і власне створені розробником для персональних потреб конкретного проєкту. Структура проєкту є інтуітивно зрозумілою, а ресурси в ньому дуже добре розташовані. Кожна категорія класів знаходиться і власній піддиректорії із зрозумілою та чіткою назвою, що робить аналіз коду значно легшим.

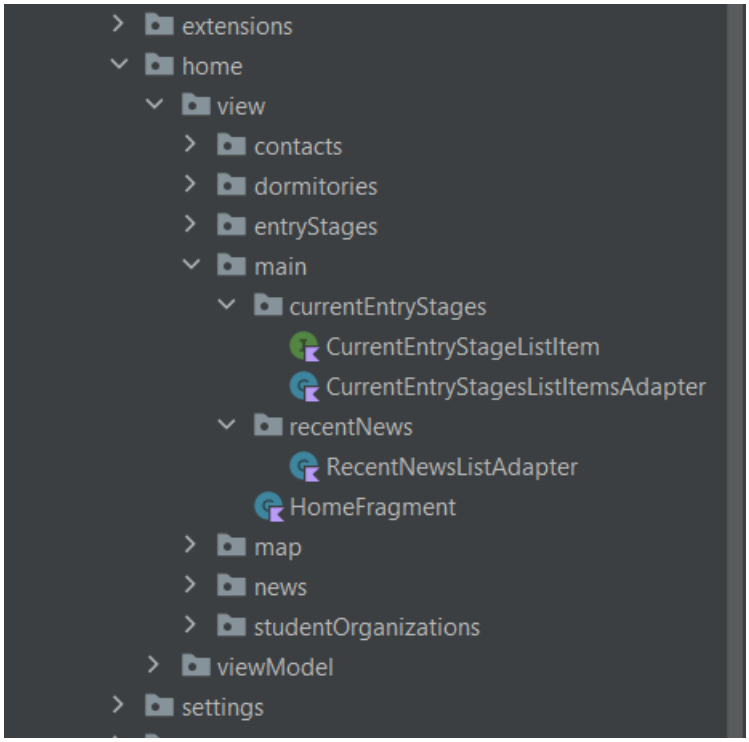

Рис. 4. Розташування класів у додатку Smart UKMA

Для запуску та аналізу додатку мною було використано стандартну IDE для розробки Android додатків Android Studio. Android Studio є дуже легкою у використанні, особливо для розробників, які звикли до IntelliJ IDEA.

Частина 2. Механізми авторизації

Авторизація – вкрай важливий компонент інформаційної безпеки. Особливо для сучасник систем та додатків. Цей механізм передбачає перевірку особи користувача і визначення того, чи мають цей користувач належні повноваження для одержання доступу до певних ресурсів. Авторизаційні механізми ґрунтуються на сукупності заздалегідь визначених правил, які регламентують, хто і при яких обставинах може отримати доступ до яких ресурсів, а також які умови будуть в цього доступу. Для того, щоб визначити, чи слід надавати доступ, чи відмовляти в ньому, ці правила можуть враховувати різноманітні чинники. Наприклад, роль користувача, належність до груп, пора доби, місцезнаходження користувача чи інші атрибути. Ефективні механізми авторизації є вкрай необхідними, оскільки вони допомагають запобігти багатьом серйозним проблемам. Наприклад, несанкціонованому доступу до чутливих ресурсів, порушенню цілісності і конфіденційності даних та зменшити ризик порушення безпеки.

Процес авторизації повинен бути ретельно розроблений і регулярно перевірятися, щоб забезпечити його ефективність і надійність у запобіганні несанкціонованому доступу до ресурсів.[22][23]

2.1 **Аутентифікація**

Авторизація зазвичай включає два ключові компоненти: автентифікацію та контроль доступу.

Автентифікація - це процес ідентифікації користувача або програми, що запитує доступ до ресурсу, для котрого налаштована авторизація. Зазвичай це робиться шляхом надання користувачем певного типу інформації, необхідної для ідентифікації його особи. Наприклад, автентифікація може проводитися за допомогою форми, що потребує введення імені та пароля користувача.

Процес авторизації складається з послідовних кроків, які мають пройти коли користувач або додаток намагаються авторизуватися в деякій системі. Ці кроки йдуть в такому порядку:

1. Аутентифікація: перш за все користувач, пристрій або програма мають пройти автентифікацію. Тобто, надати дійсне ім'я користувача та пароль, токен або іншу форму облікових даних, які вимагає дана форма авторизації.
2. Запит на авторизацію: після успішного проходження автентифікації користувач, пристрій або програма мають можливість надіслати запит про дозвіл на допущення до певного ресурсу або виконання певної дії.
3. Перевірка авторизації: система авторизації перевіряє, чи має користувач, пристрій або програма, які було аутентифіковано, необхідні дозволи для доступу до запитуваного ресурсу або виконання запитуваної дії.
4. Рішення про дозвіл на авторизацію: пересвідчившись у правильності авторизації та наявності необхідних дозволів, система авторизації надає доступ до запитуваного ресурсу або можливість виконання запитуваної дії. У випадку ж, якщо авторизація не пройдена, то запитані повноваження не надаються.
5. Доступ: Якщо доступ надано, користувачеві, пристрою або програмі дозволяється отримати доступ до запитуваного ресурсу або виконати запитувану дію.
6. Аудит: Система авторизації реєструє рішення про надання доступу. Це рішення може бути використано для ревізії або для розслідування будь-яких спроб несанкціонованого доступу.

[24][25][26][27]

Існує декілька методів, якими можливо імплементувати автентифікацію. Зокрема такими методами є:

- Автентифікація на основі паролю: для того, щоб підтвердити свою особу, користувач повинен ввести правильне ім'я користувача та відповідний

пароль. Перевагами цього типу автентифікації можна назвати такі фактори:

- Звичність і простота: паролі є вже давно розповсюдженим і тому звичним методом автентифікації для більшості користувачів. Принцип паролів легко зрозуміти, їх легко використовувати і вони вимагають мінімальної підготовки або технічних знань. Зазвичай вистачає лише короткого уточнення щодо вимог паролю для конкретного ресурсу, і немає необхідності в додатковій інформації.
- Економічна ефективність: зазвичай імплементація автентифікації на основі паролів не вимагає додаткового обладнання або спеціалізованої інфраструктури. Така простота впровадження робить її економічно вигідним варіантом.
- Користувацький контроль: при використанні цього типу автентифікації користувач сам створює та має контроль над своїм паролем, що дозволяє відчувати більшу владу над своїм обліковим записом, а значить і більшу захищеність цього запису.
- Розповсюджена підтримка: більшістю операційних систем, додатків та онлайн-сервісів підтримують автентифікацію на основі пароля, завдяки чому цей тип автентифікації може використовуватися майже всюди.

Недоліками ж можна вважати наступні фактори:

- Вразливість до підбору паролів: користувачі часто обирають паролі, які легко вгадати, наприклад, загальні слова або особисту інформацію, що може бути використано зловмисниками. Через це автентифікація на основі паролів може бути вразливою до атак на підбір паролів проведених в ручну або за допомогою автоматизованих інструментів для брут-форс атак.
- Повторне використання паролів: користувачі нерідко можуть повторно використовувати однакові паролі в різних акаунтах. Це значно підвищує ризик несанкціонованого доступу, якщо один з цих акаунтів буде скомпрометовано.
- Проблеми через невміле керування паролями: без належних знань та навичок у керуванні паролями користувачі можуть вдаватися до поганих практик зберігання паролів, що може призвести до полегшення зламу акаунтів зловмисниками.
- Соціальна інженерія: зловмисники можуть проводити атаки, в ході яких змушують користувачів розкрити свої паролі, і таким чином із легкістю зламувати автентифікацію на базі паролю.

- Автентифікація на основі біометричних даних: для підтвердження своєї особи користувач повинен надати відбитки своїх пальців, пройти процес розпізнавання обличчя або надати інші біометричні дані. Перевагами цього типу автентифікації можна вважати наступні фактори:
  - Підвищена безпека: біометричні дані, такі як відбитки пальців або малюнок райдужної оболонки ока, невід'ємно пов'язані з конкретною людиною і тому їх важко відтворити або підробити. Порівняно з традиційною автентифікацією на основі пароля це забезпечує вищий рівень безпеки.
  - Зручність та досвід користувача: впровадження біометричної автентифікації позбавляє користувачів необхідності запам'ятовувати паролі та керувати ними, що робить такий досвід простішим та приємнішим.
  - Ненав'язливість: автентифікації, імплементована саме у цій формі, може бути не інтрузивною і вимагати від користувача мінімальних зусиль, наприклад, усього лише зсунення пальців на сенсор задля сканування відбитків або переведення погляду в камеру для сканування малюнку райдужної оболонки ока.

Недоліками ж можна вважати такі фактори:

- Питання конфіденційності: біометричні дані є дуже особистою інформацією і їх використання може викликати занепокоєння щодо конфіденційності. Зберігання та обробка біометричної інформації вимагає суворих заходів безпеки для запобігання несанкціонованому доступу або зловживанню.
- Незамінність: На відміну від паролів, які можна змінити у випадку необхідності, біометричні дані є незмінними. Це ускладнює відновлення біометричного шаблону, який було тим чи іншим чином скомпрометовано.
- Неточність роботи: біометричні системи іноді можуть помилково помилково автентифікувати неавторизованих користувачів або, навпаки, не автентифікувати авторизованих користувачів. Тобто, надійність системи та якість обслуговування користувачів можуть бути доволі низькими.
- Вартість та вимоги до інфраструктури: імплементація біометричної автентифікації може вимагати спеціалізованого обладнання, програмного забезпечення та значних зусиль для правильної інтеграції. Процес може виявитися коштовним і

трудомістким, особливо у випадку, коли система має масштабне покриття.

- Двофакторна автентифікація, або 2FA (two-factor authentication): задля успішного проходження автентифікації цього типу користувач повинен пройти два рівні автентифікації, тобто надати два різних типу ідентифікаційної інформації. Наприклад, свій пароль і токен безпеки, або пароль та відбиток пальця. Перевагами цього типу автентифікації можна вважати наступні фактори:
  - Підвищена безпека: 2FA істотно підсилює безпеку профілю, поєднуючи те, що користувач знає (наприклад, пароль), з тим, чим користувач володіє (наприклад, мобільним пристроєм), або з тим, що безпосередньо властиве користувачу (наприклад, відбитком пальця).
  - Зниження вразливості паролів: оскільки при використанні 2FA зловмиснику для входу потрібен і пароль, і другий фактор, цей метод автентифікації знижує ризик несанкціонованого доступу, навіть якщо паролі скомпрометовані.
  - Підвищена надійність автентифікації: 2FA створює додатковий бар'єр для зловмисників завдяки використанню двох різних фактори автентифікації. Це гарантує вищий рівень впевненості в ідентифікації особи користувача.
  - Універсальність і гнучкість: Двофакторна автентифікація може використовувати різні другі способи ідентифікації, включаючи надсилання SMS-кодів, використання мобільних додатків, біометричних даних та апаратних токенів, або перевірку електронної пошти. Така різноманітність доступних для використання факторів дозволяє користувачам самостійно обирати найбільш підходящий для них метод.
  - Галузевий стандарт і широка підтримка: 2FA стала галузевим стандартом безпеки, і багато онлайн-платформ, сервісів та додатків пропонують вбудовану підтримку 2FA.
- Недоліками ж можна вважати такі фактори:
  - Недостатньо комфортний користувацький досвід: імплементація 2FA може негативно вплинути на зручність та комфорт користувачів, оскільки вона може призвести до збільшення кількості кроків, які необхідно пройти, або до появи додаткових складнощів під час входу в систему.

- Залежність від другого фактору: у випадку, якщо пристрій загублено, викрадено або він недоступний, наявність саме 2FA може принести проблеми, оскільки для її проходження користувачі повинні мати доступ до другого фактору (наприклад, мобільного пристрою для отримання коду чи апаратного токену).
- Можливість затримок та перебоїв зв'язку: залежність від зовнішніх факторів (наприклад, необхідність отримання SMS-коду або потреба в підключенні до Інтернету) задля проходження другого кроку автентифікації може призвести до затримок або труднощів у процесі автентифікації, що вплине на доступність для користувачів.
- Хибне відчуття безпеки: 2FA забезпечує додатковий рівень захисту, що може надавати оманливе відчуття повної захищеності. Це може призвести до послаблення обережності. Проте зловмисники все одно можуть застосовувати методи соціальної інженерії або використовувати вразливості процесу автентифікації для своїх цілей.

[24][28][29][30]

2.2. Контроль доступу

Після перевірки особи користувача система може перейти до наступного етапу процесу авторизації, а саме – контролю доступу.

Контроль доступу - це процес надання або відмови у наданні дозволів автентифікованим користувачам або організаціям на основі заздалегідь визначених правил, політик або атрибутів. Він передбачає визначення того, які дії або операції може виконувати користувач, до яких ресурсів він може отримати доступ і який рівень доступу він має (читання, запис, виконання тощо). Механізми контролю доступу забезпечують виконання цих дозволів, щоб гарантувати, що користувачі мають належний і санкціонований доступ до ресурсів.

Існує декілька основних методів імплементації контролю доступу, наприклад: обов'язковий контроль доступу (MAC), дискреційний контроль доступу (DAC), контроль доступу на основі ролей (RBAC) і контроль доступу на основі атрибутів (ABAC) та контроль доступу на основі правил. Розглянемо їх детальніше далі.

- Обов'язковий контроль доступу (MAC):
  Скорочення MAC розшифровується як Mandatory Access Control. У цій моделі доступу доступ до ресурсів ґрунтується на списку заздалегідь визначених правил та політик. Самі ці політики визначає єдине керівництво, а рішення щодо контролю доступу приймає не користувач, а система, і таким чином забезпечується неухильне та ретельне дотримання цих політик в межах всієї системи. MAC – жорстка модель контролю, і користувачам надається доступ лише до тих ресурсів, які їм потрібні для виконання своїх посадових обов'язків, не більше і не менше. Цей метод контролю доступу також має свої недоліки, наприклад складність імплементації та необхідність жорсткої адміністрації, особливо при наявності великої кількості користувачів. Складність систем MAC створює ризик некоректних конфігурацій або непередбачуваних наслідків. Неправильно налаштовані політики можуть призвести до надмірного обмеження доступу, блокуючи вповноваженим користувачам доступ до ресурсів, або до недостатнього обмеження доступу, що може призвести до несанкціонованого доступу до конфіденційних даних.
  MAC часто використовується в середовищах, де необхідний підвищений рівень безпеки, наприклад в урядових або військових системах. Прикладом системи, яка використовує для авторизації переважно

обов'язковий контроль доступу (MAC), є операційна система SELinux (Security-Enhanced Linux). SELinux забезпечує додатковий рівень контролю доступу за межами традиційного дискреційного контролю доступу в Linux (DAC). Він привласнює мітки безпеки, відомі як контексти, процесам і об'єктам, наприклад файлам, каталогам і мережевим портам. Ці мітки містять інформацію про чутливість і категорію ресурсу. У SELinux правила керування доступом засновані на мітках безпеки та визначаються політиками MAC. Політики визначають дії, які процес з певною міткою безпеки може виконувати над ресурсами з певними мітками. Рішення про доступ приймає сама система, для цього строго дотримуючись політик і запобігаючи виконанню несанкціонованих дій.

- Дискреційний контроль доступу (DAC):
  Акронім DAC розшифровується як Discretionary Access Control. У цій моделі контролю доступу доступ до ресурсів надається власне на розсуд власника ресурсу. Він має повноваження забороняти або дозволяти доступ до ресурсів, захищених за допомогою DAC, на основі критеріїв, які сам же і вирішує. DAC зазвичай основується на принципі найменших привілеїв, тобто користувачам надається лише мінімальний рівень доступу, необхідний для виконання їхніх робочих функцій. Основним недоліком цього типу контролю є те, що задля його надійної роботи необхідно, щоб сам власник правильно та розсудливо налаштував дозволи, і його помилки можуть призвести до суперечностей або вразливостей.
  Цей тип контролю доступу часто використовується в невеликих, менш складних системах. Прикладом такої системи, яка використовує дискреційний контроль доступу для авторизації, є операційна система Linux. У Linux DAC реалізовано за допомогою дозволів на доступ до файлів та на право володіння ними. Власник файлу або каталогу може на власний розсуд встановлювати дозволи для цього конкретного ресурсу, як для себе так і для інших користувачів.
    - Існує окремий механізм, що використовується для імплементацій систем DAC, під назвою ACL (Access Control List, тобто список управління доступом). ACL – це структури даних, які зазвичай використовуються для реалізації DAC. Вони являють собою списки записів про контроль доступу (access control entries, або ACE), що визначають дозволи, які власник якогось ресурсу надав певним користувачам або, навпаки, обмежив їм. Кожен ACE в

ACL містить інформацію про користувача, тип доступу (наприклад, читання, запис, виконання) і відповідний дозвіл (дозволити або заборонити). Механізм DAC може забезпечувати контроль доступу до певного ресурсу на основі дозволів, зазначених у прив’язаному до цього ресурсу ACL. Коли користувач намагається отримати доступ до ресурсу, система перевіряє записи у відповідному ACL, і за допомогою записів у ньому визначає, чи має цей користувач дозвіл на дію, яку він хоче зробити.

- Контроль доступу на основі ролей (RBAC):
  Акронім RBAC розшифровується як role-based access control. У цій моделі контролю доступу рішення щодо контролю доступу ґрунтуються на ролях, приписаним користувачам або програмам. Кожна роль має власний набір дозволів, і ці дозволи визначають, які дії користувачеві або додатку дозволено виконувати. RBAC зазвичай базується на принципі розподілу обов'язків, тобто користувачам надається доступ лише до тих ресурсів, які необхідні для виконання їхніх конкретних посадових обов'язків, подібно до системи MAC. Проте цей тип контролю доступу має свої недоліки. Впровадження RBAC може бути непропорційно складним, особливо у великих організаціях або середовищах з різноманітними ролями та дозволами. Розробка та визначення ролей, прив'язка дозволів до ролей та управління призначеннями ролей вимагають ретельного планування та координації. Також у складних середовищах кількість ролей, необхідних для точного представлення різних посадових функцій і вимог до доступу, може значно зрости. Це може призвести до явища, відомого як "вибух ролей", тобто випадків, коли кількість ролей стає некерованою і напряму перешкоджає ефективності RBAC. Також, із часом при еволюції позицій та їх пов’язаних із ними обов’язків з’являється ризик того, що користувачі матимуть доступи, яких не мають мати у зв’язку із нинішньою своєю роллю. Нарешті, RBAC може не бути в змозі забезпечити достатню точність, необхідну для ретельного контролю доступу. Хоча ролі визначають широкі набори дозволів, може бути важко або навіть майже неможливо надати конкретні дозволи окремим користувачам у межах ролі, наприклад у випадках, користувачам, що мають одну роль, потрібні різні права доступу.
  RBAC часто використовується у великих, складніших системах, оскільки дозволяє адміністраторам керувати доступом до ресурсів на

більш детальному рівні. Одним із прикладів системи, яка широко використовує контроль доступу на основі ролей (RBAC), є система Microsoft Active Directory (AD). AD - це служба каталогів, яка широко використовується в корпоративних середовищах для централізованого управління користувачами, автентифікації та контролю доступу. В AD RBAC використовується для призначення дозволів і прав доступу користувачам на основі їхніх ролей або посадових обов'язків в організації. Система включає заздалегідь визначені ролі, відомі як групи безпеки, такі як адміністратори доменів, оператори облікових записів або служба підтримки, які представляють загальні робочі ролі в IT-інфраструктурі. Адміністратори можуть створювати власні ролі, визначаючи групи безпеки та призначаючи певні дозволи для цих груп. Дозволи можна надавати на різних рівнях, наприклад, на рівні домену, організаційного підрозділу (OU) або конкретних ресурсів, таких як файли або принтери. RBAC в AD дозволяє ефективно керувати дозволами користувачів, призначаючи користувачів до відповідних груп безпеки на основі їхніх ролей. Це спрощує процес адміністрування, оскільки адміністратори можуть призначати дозволи і права доступу групі, а не окремим користувачам. Будь-який користувач, доданий до групи, автоматично успадковує відповідні дозволи. RBAC в AD також підтримує вкладеність ролей, коли група безпеки може бути членом іншої групи, успадковуючи дозволи від батьківської групи. Таке вкладання забезпечує ієрархічний контроль доступу і полегшує управління складними вимогами до доступу.

- Контроль доступу на основі атрибутів (ABAC):
  Англійською це скорочення розшифровується як attribute-based access control. У цій моделі управління доступом рішення про доступ приймаються на основі набору атрибутів, які прив’язані до користувача або додатка. Серед цих атрибутів можуть бути такі, як посада користувача, відділ, до якого він належить, місцезнаходження та інші. Зазвичай ABAC базується на принципі динамічної авторизації, що означає, що рішення про контроль доступу приймаються в режимі реального часу на основі поточного контексту. Звісно, ця система також має недоліки. Наприклад, складність у впровадженні. ABAC вимагає визначення та управління численними атрибутами, пов'язаними з користувачами, ресурсами та умовами навколишнього середовища. Інша проблема - стандартизація та інтероперабельність: ABAC не має стандартизованих угод щодо іменування атрибутів і форматів значень

атрибутів, що може ускладнити інтероперабельність між різними системами або політиками ABAC. Без стандартизації організації можуть зіткнутися з труднощами при інтеграції ABAC в різні системи або при спільному використанні ресурсів із зовнішніми організаціями. Ще однією потенційною проблемою є збої в роботі: оцінка рішень щодо контролю доступу на основі декількох атрибутів і правил політики може призвести до додаткових накладних витрат на обробку даних, що потенційно впливає на продуктивність системи. І, нарешті, потенційна неузгодженість атрибутів: Розбіжності у значеннях атрибутів або механізмах пошуку атрибутів у різних системах або джерелах даних можуть призвести до суперечливих рішень щодо управління доступом. Організації повинні забезпечити точність і узгодженість значень атрибутів у всіх відповідних системах і джерелах, щоб підтримувати цілісність політик ABAC.

ABAC часто використовується в динамічних середовищах, де вимоги до доступу часто змінюються. Прикладом системи, що використовує ABAC як тип авторизації, є Google Cloud IAM (Identity and Access Management). Google Cloud IAM - це централізована служба управління доступом до ресурсів Google Cloud Platform. Вона дозволяє організаціям визначати політики доступу на основі атрибутів, пов'язаних з користувачами, групами, ресурсами та іншою контекстною інформацією. Політики IAM в Google Cloud забезпечують надійний контроль доступу до ресурсів за допомогою таких атрибутів, як ідентифікатор користувача, IP-адреса, тип ресурсу тощо.

- Контроль доступу на основі правил:

  У моделі управління доступом на основі правил рішення про контроль доступу приймаються на основі набору правил, визначених системним адміністратором. Ці правила можуть визначати, яким користувачам або програмам дозволено доступ до яких ресурсів і за яких обставин. Звісно, маючи свої власні переваги, вона також має і недоліки. Контроль доступу на основі правил часто базується на попередньо визначених статичних правилах, які диктують рішення щодо контролю доступу. Такий брак гнучкості може ускладнити обробку складних або динамічних вимог до доступу. Інша проблема полягає в тому, що контроль доступу на основі правил зазвичай зосереджується на призначенні дозволів на грубому рівні, такому як ролі або групи, а не на тонкому рівні. Це може призвести до ситуацій, коли користувачі мають більше дозволів, ніж їм насправді потрібно, або коли дозволи не

відповідають індивідуальним вимогам користувачів. Подібно до методу контролю доступу на основі ролей, використання контролю доступу на основі правил може призвести до "вибуху ролей", тобто в складних середовищах може статися проліферація ролей, оскільки організації намагаються представити різні робочі функції і вимоги до доступу, що робить управління системою більш складним і ресурсномістким. Брак контекстної інформації також може бути проблемою: Контроль доступу на основі правил часто зосереджується виключно на ролі користувача і не бере до уваги іншу контекстну інформацію, яка може вплинути на рішення щодо контролю доступу. Такі фактори, як час доби, місцезнаходження, тип пристрою або історія транзакцій, зазвичай не враховуються в традиційних моделях контролю доступу на основі правил. Також можуть виникнути труднощі з міжорганізаційною співпрацею. Контроль доступу на основі правил в першу чергу призначений для використання в межах однієї організації, і його нелегко поширити на міжорганізаційну співпрацю. При спільному використанні ресурсів або систем із зовнішніми організаціями узгодження моделей і політик RBAC може бути складним, особливо якщо різні організації мають різні рольові структури або домовленості щодо іменування.

Контроль доступу на основі правил часто використовується в системах, де рішення про контроль доступу потрібно приймати швидко і без втручання людини. Одним з прикладів системи, яка використовує управління доступом на основі правил, є брандмауер. Брандмауери широко застосовуються для контролю вхідного і вихідного мережевого трафіку. У брандмауері рішення про контроль доступу приймаються на основі набору правил, які визначають який трафік дозволено або заборонено на основі різних критеріїв. Цими критеріями можуть бути IP-адреса джерела, IP-адреса призначення, номери портів, протоколи тощо. Ці правила діють як статичні політики, які визначають, як саме брандмауер має обробляти мережевий трафік. Адміністратор брандмауера налаштовує правила для визначення який трафік може пройти через брандмауер, а який буде заблоковано. Наприклад, правило може вказувати, що вхідний трафік з певного діапазону IP-адрес необхідно заблокувати, а трафік з іншого діапазону - пропустити. Правила також можуть визначати винятки і конкретні умови, за яких певний трафік необхідно обробляти по-іншому. Брандмауер послідовно оцінює мережеві пакети на відповідність правилам. Він виконує дії на основі відповідності правилам. Тобто, пакет може пройти через

брандмауер у випадку, якщо відповідає правилу, яке його дозволяє. Якщо ж пакет підпадає під правило, яке його забороняє, то він блокується. Брандмауер послідовно застосовує ці правила, і таким чином мережевий трафік гарантовано дотримується попередньо визначених політик контролю доступу.

[31][32][33][34][35][36][37][38][39][40]

Кожен з типів контролю доступу має свою галузь застосування. Зазвичай програми Android використовують комбінацію різних механізмів контролю доступу для забезпечення безпеки. Основною моделлю контролю доступу, яка використовується в програмах Android, є обов'язковий контроль доступу (MAC), як зазначено в офіційній документації Android [41]. Хоча точніше було б сказати, що суміш принципів MAC і DAC використовується найчастіше, іноді також використовуються інші типи контролю доступу.

- Використання MAC: Android використовує MAC за допомогою механізму під назвою «пісочниця». Для запуску кожної програми Android використовується окреме середовище ізольованого програмного середовища з унікальним ідентифікатором користувача. Завдяки цій ізоляції можна гарантувати, що програми не зможуть отримати прямий доступ або втручатися в інші програми або базові системні ресурси. Операційна система Android підтримує MAC, обмежуючи зв'язок між програмами та накладаючи суворий контроль доступу до системних ресурсів.
- Використання DAC: Android включає DAC як основний механізм контролю доступу. DAC дозволяє розробникам додатків визначати права доступу до різних ресурсів, таких як файли, мережеві підключення, датчики та системні служби. Дозволи оголошено у файлі AndroidManifest.xml і можуть запитуватися програмою під час виконання. Користувачі мають можливість надавати або забороняти ці дозволи під час встановлення або використання програми.[42]
- Модель дозволів виконання: починаючи з Android 6.0 (Marshmallow), Android представив модель дозволів часу виконання, розширюючи можливості існуючої системи дозволів. Завдяки цій моделі певні конфіденційні дозволи, як-от доступ до камери, мікрофона чи розташування, вимагають явного схвалення користувача під час виконання, навіть якщо вони оголошені в маніфесті. Це додає додатковий рівень контролю доступу користувачів, дозволяючи користувачам отримати більше контролю над власними пристроями та встановленим на них програмним забезпеченням.

[https://docs.unity3d.com/Manual/android-RequestingPermissions.html]

- Використання RBAC: ще один набір принципів, які також використовує Android у своїй структурі, це RBAC. Цей тип керування доступом надає попередньо визначені ролі та дозволи, такі як «система», «інсталятор» або «програма», які призначаються різним об’єктам у межах система. Ці ролі визначають рівень привілеїв, наданих кожній сутності, таким чином покращуючи можливості контролю доступу в екосистемі Android. [42]

Поєднуючи MAC, DAC, дозволи під час виконання та RBAC, програми Android досягають багатогранного підходу до контролю доступу. Це гарантує, що програми працюють у власних середовищах ізольованого програмного середовища, не заважаючи одна одній, дотримуються наданих користувачами дозволів і дотримуються політик контролю доступу на рівні системи. Таким чином можна отримати безпечне та контрольоване середовище для користувачів програми Android.

2.3. Автентифікація на прикладі власного Android додатку “Authorization Showcase”

Задля можливості розглядання авторизації на практичному прикладі мною було написано власний Android додаток “Authorization Showcase”. Далі буде надано розібрано систему авторизації на прикладі саме цього додатку.
Даний додаток інтегрується з Azure Active Directory (Azure AD) для того, щоб автентифікувати користувачів та отримувати токени доступу, які необхідні для виклику Microsoft Graph API. Завдяки використанню Azure AD цей додаток здатен приймати спроби входу з особистих облікових записів Microsoft (зокрема outlook.com, live.com та інших), а також робочих або навчальних облікових записів будь-якої компанії чи організації, яка використовує Azure AD.
Додаток “Authorization Showcase” використовує бібліотеку автентифікації Microsoft Authentication Library (MSAL) для Android для реалізації автентифікації. Власне MSAL підтримує принципи RBAC. Тобто, він дозволяє розробникам самостійно визначати ролі та дозволи у своїх додатках або на основній платформі для ідентифікації Azure Active Directory (Azure AD). RBAC забезпечує гнучкий спосіб керування доступом до ресурсів на основі ролей, призначених користувачам. За допомогою RBAC розробники можуть призначати користувачам певні ролі (наприклад, адміністратор, менеджер або учасник), які визначають права та привілеї цих учасників в рамках додатку.[43]
При запуску додатку користувач опиняється на екрані автентифікації, де має натиснути кнопку sign-in для того, щоб пройти далі.

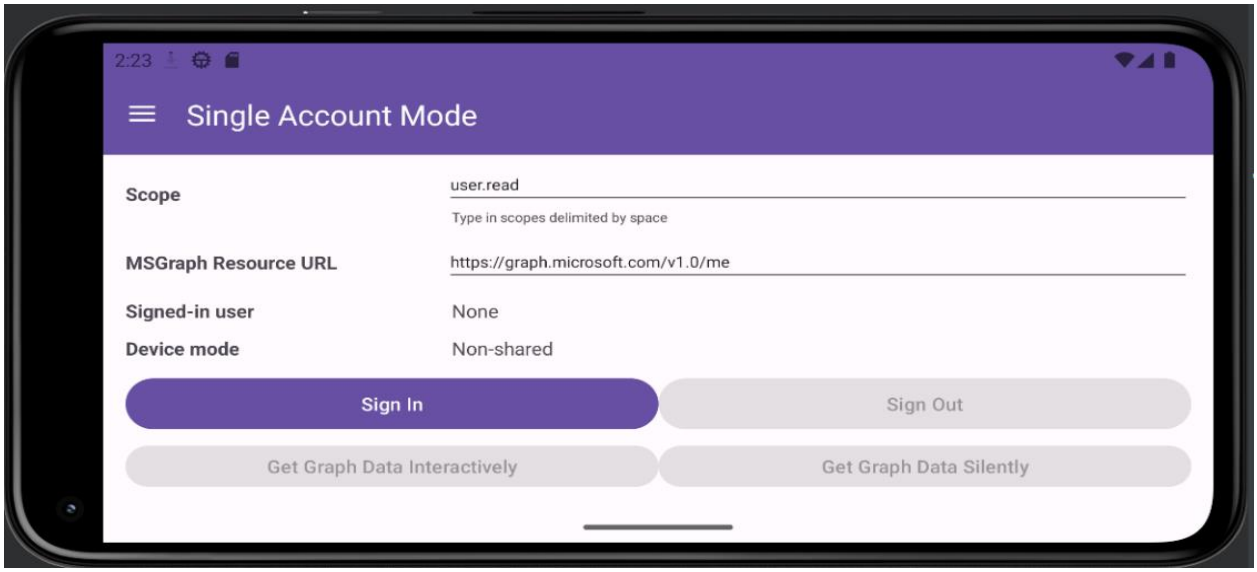

Рис. 5. Перший екран додатку "Authorization Showcase".
Стадія початку автентифікації

Після цього користувача перенаправлено до вітальної сторінки для користування Chrome.

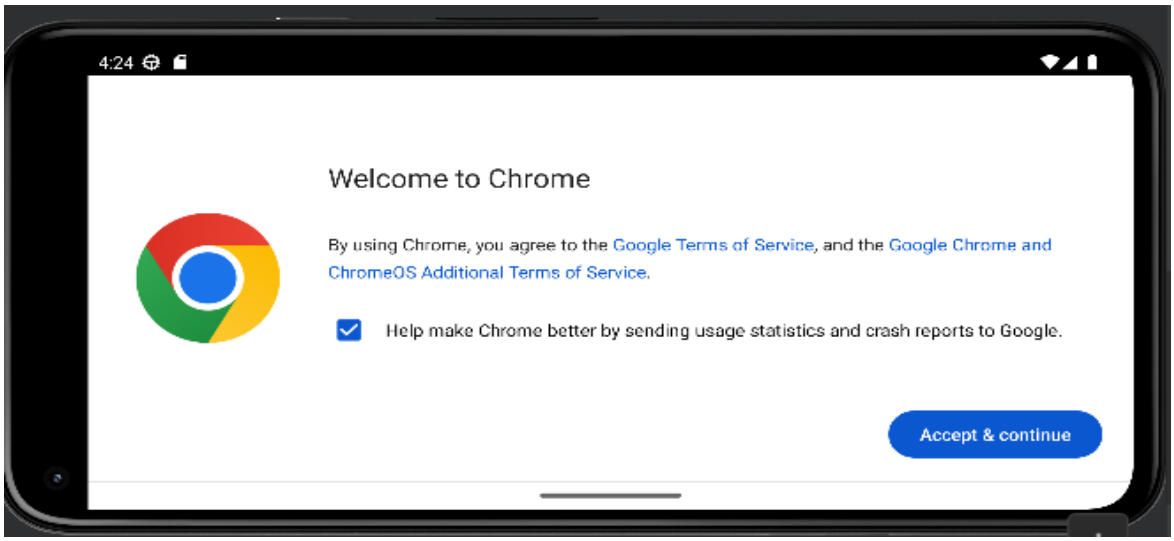

Рис. 6. Вітальна сторінка Chrome

При виборі цієї опції користувача буде переадресовано на наступний екран, де знаходиться Microsoft форма реєстрації, а саме – форма входу у акаунт Google.

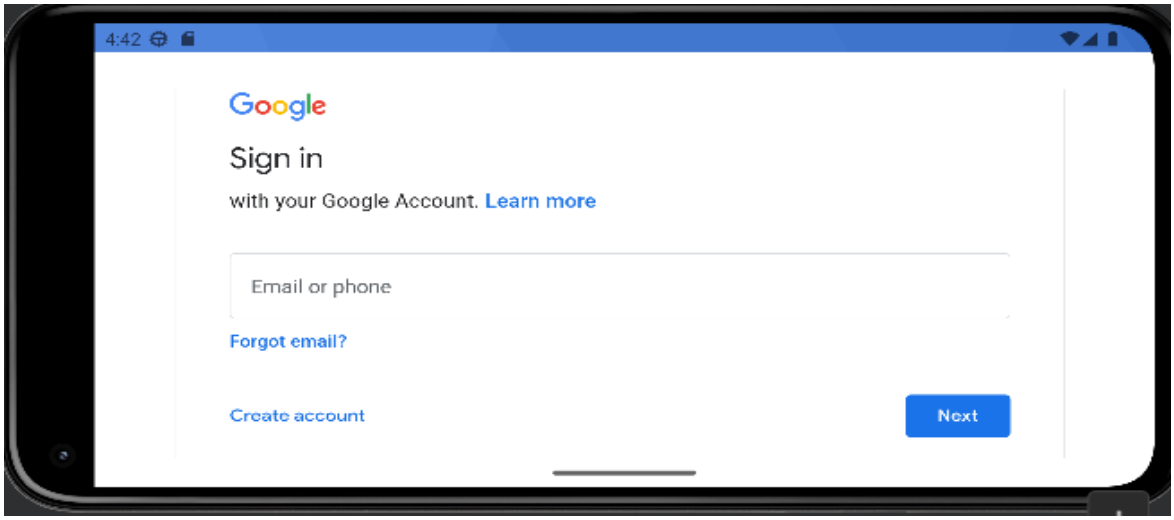

Рис. 7. Вхід у Google акаунт.

Наступним кроком є введення паролю. У даному випадку використовується класична автентифікація на основі паролю.

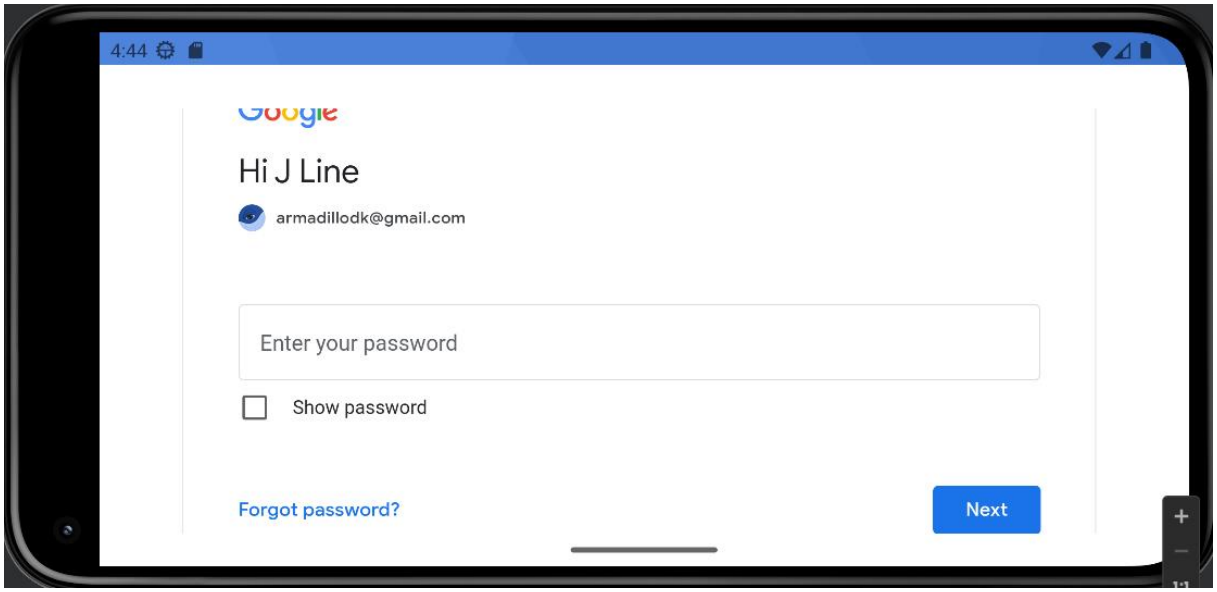

Рис. 8. Запит на введення паролю

Після введення правильного паролю Google акаунт відзначає, що користувач успішно увійшов у свій акаунт.

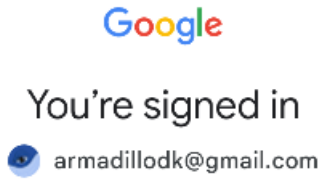

Рис. 9. Користувач успішно увійшов

Після цього користувач може продовжувати роботу із пристроєм тепер вже під своїм власним акаунтом, або ж вийти з цього акаунта. Якщо не виходити із облікового запису, то при повторному запуску програми при перевірці акаунту буде відмічено, що на даний момент в системі знаходиться користувач, який ввійшов минулого разу.

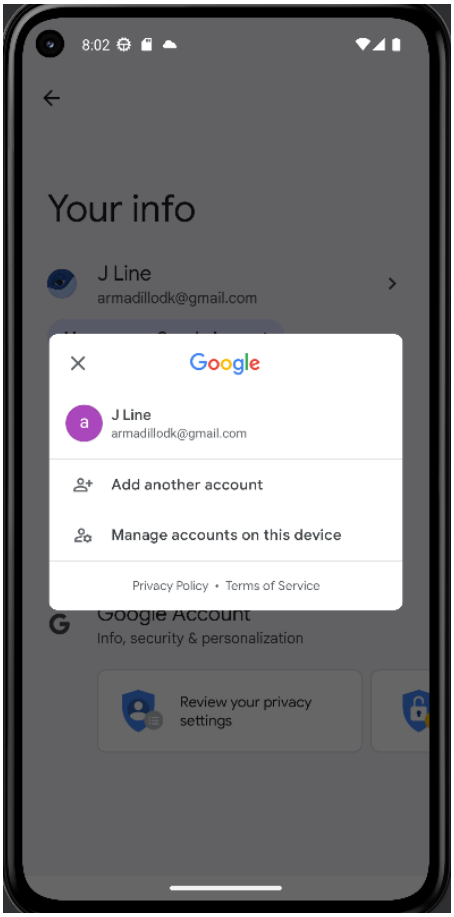

Рис. 10. При повторному вході система пам’ятає користувача.

В деяких випадках додаток, після успішного прохождення авторизації, залишається на непровантаженній сторінці com.microsoftonline.com.

Рис. 11. Зависання на com.microsoftonline.com

Така проблема є дуже розповсюдженою для додатків, що використовують авторизацію через Microsoft. Процесс часто затягується, незалежно від особливостей системи, що намагається виконати вхід. В моєму випадку, нажаль, час, який може зайняти вхід, вочевидь довше за 30 хвилин. Після того, як цей час спливає, мій ноутбук зупиняє процес самостійно, оскілки в іншому випадку може серйозно перегрітися.

Посилання на джерела

1 *Android App Development: Which Language to Choose*. (n.d.). Retrieved from codemotion: https://www.codemotion.com/magazine/frontend/mobile-dev/android-app-development-which-language-to-choose/

2 *What is an Android Operating System & Its Features*. (n.d.). Retrieved from elprocus.com: https://www.elprocus.com/what-is-android-introduction-features-applications/

3 *Which is better, Kotlin vs. java? Let's settle the debate*. (n.d.). Retrieved from DAC.digital: https://dac.digital/kotlin-vs-java/

4 *Top Programming Languages for Mobile App Development*. (n.d.). Retrieved from nativapps: https://nativapps.com/top-programming-languages-for-mobile-app-development/

5 *Modern web application architecture to build a high-performance app*. (n.d.). Retrieved from acropolium: https://acropolium.com/blog/modern-web-app-architecture/

6 *What is Java IDE?* (n.d.). Retrieved from javatpoint: https://www.javatpoint.com/java-ides

7 *Android*. (n.d.). Retrieved from archlinux: https://wiki.archlinux.org/title/android

8 *Run apps on the Android Emulator*. (n.d.). Retrieved from developer.android.com: https://developer.android.com/studio/run/emulator

9 *Application Programming Interface: Design, Features, and Functionalities*. (n.d.). Retrieved from www.turing.com: https://www.turing.com/kb/application-programming-interface

10 *Android Testing: A Beginner's Guide to Getting Started*. (n.d.). Retrieved from www.testim.io: https://www.testim.io/blog/android-testing/

11 *integrated development environment (IDE)*. (n.d.). Retrieved from www.techtarget.com: https://www.techtarget.com/searchsoftwarequality/definition/integrated-development-environment

12 *What Is React Native? Complex Guide for 2022*. (n.d.). Retrieved from www.netguru.com: https://www.netguru.com/glossary/react-native

13 *A Complete Guide to Android App Development*. (n.d.). Retrieved from www.explorateglobal.com: https://www.explorateglobal.com/blog/android-app-development/

14 *What is Firebase? All the secrets unlocked*. (n.d.). Retrieved from blog.back4app.com: https://blog.back4app.com/firebase/

15 *What is Firebase?* (n.d.). Retrieved from appmaster.io: https://appmaster.io/blog/what-is-firebase

16 *Top 20 Mobile App Development Tools*. (n.d.). Retrieved from blog.back4app.com: https://blog.back4app.com/mobile-app-development-tools/

17 *The Pros and Cons of Android Studio and App Tools*. (n.d.). Retrieved from www.pangea.ai: https://www.pangea.ai/dev-mobile-app-resources/the-pros-and-cons-of-android-studio-and-app-tools/

18 Accessing resources—from sample code to tests. (n.d.). Retrieved from www.ibm.com: https://www.ibm.com/topics/android-development

19 Essential Skills Needed To Become a Perfect Android Developer. (n.d.). Retrieved from www.sysbunny.com: https://www.sysbunny.com/blog/essential-skills-needed-to-become-a-perfect-android-developer/

20 8 Must-Have Skills for Becoming an Android App Developer. (n.d.). Retrieved from www.geeksforgeeks.org: https://www.geeksforgeeks.org/8-must-have-skills-for-becoming-an-android-app-developer/

21 Mobile App Development Process: Ultimate Guide to Build an App. (n.d.). Retrieved from www.velvetech.com: https://www.velvetech.com/blog/mobile-app-development-process/

22 Security Authentication vs. Authorization | A Quick Guide. (n.d.). Retrieved from swoopnow.com: https://swoopnow.com/security-authentication-vs-authorization/

23 What Is The Purpose Of Authorisation? (n.d.). Retrieved from security-systems.net.au: https://security-systems.net.au/what-is-the-purpose-of-authorisation/

24 Authentication: Methods, Protocols, and Strategies. (n.d.). Retrieved from frontegg.com: https://frontegg.com/blog/authentication

25 Understanding Authentication, Authorization, and Encryption. (n.d.). Retrieved from www.bu.edu: https://www.bu.edu/tech/about/security-resources/bestpractice/auth/

26 Authentication and Authorization. (n.d.). Retrieved from realtimelogic.com: https://realtimelogic.com/ba/doc/en/C/authentication.html

27 Why Do We Need Authorization and Authentication? (n.d.). Retrieved from dev.to: https://dev.to/mariamarsh/why-do-we-need-authorization-and-authentication-13d9

28 User Authentication: Understanding the Basics & Top Tips. (n.d.). Retrieved from swoopnow.com: https://swoopnow.com/user-authentication/

29 Which Methods Can Be Used to Implement Multifactor Authentication? (n.d.). Retrieved from www.cgaa.org: https://www.cgaa.org/article/which-methods-can-be-used-to-implement-multifactor-authentication

30 A Guide to the Types of Authentication Methods. (n.d.). Retrieved from www.veriff.com: https://www.veriff.com/blog/types-of-authentication-methods

31 Authorization Cheat Sheet. (n.d.). Retrieved from cheatsheetseries.owasp.org: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

32 Access Control Models: Review of Types and Use-Cases. (n.d.). Retrieved from medium.com: https://medium.com/yellow-universe/access-control-models-review-of-types-and-use-cases-1f4c427b0cc2

33 What is MAC (Mandatory Access Control)? (n.d.). Retrieved from goteleport.com: https://goteleport.com/learn/what-is-mac/

34 Maximizing Data Security with Role-Based Access Control. (n.d.). Retrieved from bigid.com: https://bigid.com/blog/what-is-role-based-access-control-rbac/

35 Mandatory Access Control (MAC): how does it work? (n.d.). Retrieved from www.ionos.com: https://www.ionos.com/digitalguide/server/security/what-is-mandatory-access-control-mac/

36 Access Control Models. (n.d.). Retrieved from thecybersecurityman.com: https://thecybersecurityman.com/2017/12/26/access-control-models/

37 BROKEN ACCESS CONTROL. (n.d.). Retrieved from cqr.company: [https://cqr.company/web-vulnerabilities/broken-access-control/]

38 Role-based access control (RBAC). (n.d.). Retrieved from www.sailpoint.com: https://www.sailpoint.com/identity-library/what-is-role-based-access-control/

39 Role-Based Access Control (RBAC). (n.d.). Retrieved from www.imperva.com: https://www.imperva.com/learn/data-security/role-based-access-control-rbac/

40 Access Control List (ACL). (n.d.). Retrieved from www.imperva.com: https://www.imperva.com/learn/data-security/access-control-list-acl/

41 Android Security Features. (n.d.). Retrieved from ource.android.com: https://source.android.com/docs/security/features

42 Android Runtime Permissions Example. (n.d.). Retrieved from www.digitalocean.com: https://www.digitalocean.com/community/tutorials/android-runtime-permissions-example

43 Implement role-based access control. (n.d.). Retrieved from learn.microsoft.com: https://learn.microsoft.com/en-us/azure/active-directory/develop/howto-implement-rbac-for-apps