Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Пояснювальна записка

до магістерської кваліфікаційної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: Інтерактивна система інкрементного резервування даних

Виконав: студент II курсу, групи 1ПІ-18м
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Горовий Є. В.
(прізвище та ініціали)
Керівник: к.т.н., доц. Рейда О. М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. Колесницький О. К.
(прізвище та ініціали)

Вінниця – 2019

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Освітньо-кваліфікаційний рівень – магістр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Зав. каф. ПЗ, проф., д.т.н.
_____О. Н. Романюк
”___”_____ 20___ р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Горовому Євгенію Вікторовичу

1. Тема роботи: Інтерактивна система інкрементного резервування даних.

Керівник роботи: Рейда Олександр Миколайович, к. т. н., доцент, затверджені наказом вищого навчального закладу від “__”_____20__ року №__

2. Строк подання студентом роботи: _____

3. Вихідні дані до роботи: операційна система – Android; інтегроване середовище розробки – Android Studio; мова програмування – Java; тип резервування – інкрементне.

4. Зміст розрахунково-пояснювальної записки: вступ; обґрунтування доцільності розробки; аналіз засобів розробки системи; програмна реалізація системи; тестування; економічне обґрунтування доцільності розробки; висновки.

5. Перелік графічного матеріалу: актуальність обраної теми; основні задачі дослідження; наукова новизна одержаних результатів; порівняльний аналіз аналогів; функціональні відмінності типів резервування; графічний інтерфейс.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Рейда О. М к.т.н., доц. кафедри ПЗ		
5	Бальзан М. В., к.е.н., доц. кафедри ЕПВМ		

7. Дата видачі завдання: _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Обґрунтування доцільності розробки	07.10.2019 – 27.10.2019	Вик.
2	Аналіз засобів розробки системи	28.10.2019 – 8.11.2019	Вик.
3	Програмна реалізація системи	9.11.2019 – 20.11.2019	Вик.
4	Тестування	21.11.2019 – 1.12.2019	Вик.
5	Економічне обґрунтування доцільності розробки	1.12.2019 – 6.12.2019	Вик.

Студент _____ **Горовий Є.В.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Рейда О.М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

В магістерській дипломній роботі розроблено інтерактивну систему інкрементного резервування даних як мобільний додаток для операційної системи Android. Розроблений програмний продукт вирішує проблему резервування даних між хмарним сховищем та мобільним пристроєм та надає можливість його використання користувачами з різним рівнем навичок використання смартфона.

Для реалізації системи було проаналізовано технології розробки, обґрунтовано вибір засобів розробки, обрано архітектуру, розроблено алгоритми функціонування та спроектовано інтуїтивно-зрозумілий інтерфейс користувача.

Розробка відбувалась в середовищі Android Studio з використанням системи керування версіями Git та мови програмування Java.

Тестування підтвердило ефективність і правильність функціонування розробленої системи.

Також проведено економічні розрахунки витрат та прибутків від впровадження розробки, терміни окупності та визначено її комерційний потенціал.

ASBTRACT

In the master’s qualifying work an interactive system for incremental backup as mobile application for Android operating system was developed. The developed software product solves a problem of backup between the cloud storage and the mobile device and allows it to be used by users with different levels of smartphone usage skills.

For the implementation of the system, the technology of development was analyzed, the development tools choice was justified, an architecture was chosen, an algorithm of functioning was created and an intuitive user interface was designed.

The development was made in the Android Studio environment using the Git version control system and Java programming language.

Testing confirmed the efficiency and correctness the developed system functioning.

Economic calculations of costs and profits from the system implementation, payback period and its commercial potential were also conducted.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Мобільні пристрої, такі як смартфони та планшети стають тісніше пов’язаними і з нашим життям: в світі налічується близько 5 мільярдів користувачів мобільних телефонів, з них близько половини користуються смартфонами [1]. Ці пристрої пропонують широкий діапазон можливостей для підключення користувачів до мережі у будь-якій точці світу і у будь-який час. Однак мобільні пристрої мають суттєвий недолік – обмежений обсяг пам’яті для зберігання даних.

З покращенням швидкості, надійності та доступності мережі Інтернет за останні роки збільшився також і ринок провайдерів, які надають користувачам послуги хмарного зберігання даних. Але проблема резервування даних пристроїв залишається не вирішеною.

Як відомо, на даний момент жоден із відомих пристроїв зберігання інформації (в тому числі і Flash Memory, які застосовуються у мобільних пристроях) не є повністю захищеними від поломки, більш того, ми навіть не можемо достатньо точно передбачити момент виходу накопичувача з ладу. Для вирішення цієї проблеми в наш час запропоновано велику кількість програмних та апаратних засобів.

Актуальність роботи полягає у створенні системи, яка підвищує швидкодію і цілісність резервування для мобільних пристроїв за рахунок інкрементного методу оновлення даних у хмарному середовищі.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Мета полягає у підвищенні швидкодії і цілісності резервування.

Основними задачами дослідження є:

1. Обґрунтування доцільності розробки;

2. Аналіз технології розробки системи;

3. Розробка методу підвищення швидкодії резервування даних;

4. Програмна реалізація системи резервування даних;

5. Тестування системи;

6. Економічне обґрунтування доцільності розробки.

Об’єкт дослідження – процес розробки інтерактивної системи для резервування даних.

Предмет дослідження – методи та засоби розробки інтерактивної системи для резервування даних.

Методи дослідження. У процесі дослідження використовувалися: методи розробки мобільних додатків, методи резервування даних, методи передачі даних, методи створення баз даних у хмарному середовищі.

Наукова новизна отриманих результатів.

1. Вперше розроблено метод інкрементного резервування даних у хмарному середовищі за таких умов:

- наявність надійного та швидкого інтернет-з’єднання;
- наявність достатнього заряду акумулятора;
- низька завантаженість системи.

Практична цінність отриманих результатів. Розроблено інтерактивну систему інкрементного резервування даних для мобільних пристроїв на платформі Android. Проект призначено для підвищення швидкодії процесу резервування даних мобільних пристроїв з використанням хмарного сховища.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ

1.1 Аналіз предметної області

Оскільки мобільні пристрої починають отримувати нові та зручніші функції та починають відігравати більш значущу роль у нашому житті, від них потребується виконання більш складних задач. Однак ці пристрої є обмеженими в обчислюваних, накопичувальних та енергетичних ресурсах, тому повинні використовувати передові технології для того, щоб виконувати поставленні завдання на прийнятному рівні.

Мобільне хмарне сховище – це вид хмарного сховища, яке доступне для мобільних пристроїв таких, як ноутбуки, смартфони та планшети [2]. Провайдери послуг мобільного хмарного зберігання пропонують послуги, які дозволяють користувачам створювати та упорядковувати файли, папки, музику та фотографії. Послуги компаній можуть використовуватись як індивідуальними користувачами, так і компаніями. Більшість провайдерів пропонують обмежений безкоштовний об’єм сховища та платний об’єм при перевищенні вільного безкоштовного об’єму сховища. Зазвичай, витрати нараховуються як щомісячна платня та мають різні ставки в залежності від бажаного обсягу зберігання.

Деякі виробники мобільних пристроїв включають програми мобільного хмарного зберігання з їх продуктом, що полегшує резервування користувацьких файлів на різних платформах. Частина процесу налаштування нових мобільних пристроїв часто включає в себе налаштування служби хмарного зберігання для резервного копіювання файлів і даних пристрою. Пристрої Apple на базі операційної системи iOS постачаються з попередньо-завантаженими та налаштованими на використання службами зберігання даних Apple iCloud. Google пропонує подібну функцію в операційній системі Android, скориставшись резервною копією пристрою за допомогою облікового запису Google Drive. Смартфон Samsung Galaxy має партнерство з Dropbox, а Microsoft так само пропонує Microsoft OneDrive [3].

На думку дослідників з Технологічного Університету Нью-Джерсі, використання хмарних технологій для збереження даних на мобільних пристроях повинно задовольняти таким вимогам:

1. Можливості для збереження даних. Хмарні сервіси збереження даних на мобільних пристроях, замість спрощеної версії повинні мати схожий функціонал з тими, які використовуються на персональних комп'ютерах;

2. Ефективність. Хмарні сервіси повинні бути надійними, енергоефективними, та використовувати мережеві ресурси ефективно. Вони повинні бути спроектовані, базуючись на особливостях та обмеженнях мобільних пристроїв (обмежена кількість пам'яті та енергоресурсів, та обмежена і потенційно дорога вартість Інтернет-з'єднання);

Хмарне середовище як основне місце зберігання даних. Більшість служб для резервного зберігання даних будуть добре працювати на мобільних пристроях. На відміну від персональних комп'ютерів, мобільні пристрої більше залежать від хмарного сховища в плані збереження даних та проведення обчислень. Для цього бажано мати основне сховище на хмарі, яка зберігає дані та проводить обчислення [4].

Резервуванням називають метод підвищення надійності технічного засобу за рахунок введення надлишку. Під надлишком при цьому розуміють додаткові засоби і можливості окрім мінімально необхідних для виконання технічним засобом заданих функцій. Таким чином, задачею введення надлишку є забезпечення нормального функціонування технічного засобу після виникнення відмов у її елементах [5].

Відповідно до ГОСТ 13377-75 розрізняють три основних види резервування:

− структурне;

− інформаційне;

− тимчасове.

Структурне резервування (або апаратне) передбачає використання надлишкових елементів технічного засобу.

Інформаційне резервування передбачає використання надлишкової інформації.

Тимчасове резервування передбачає використання надлишкового часу.

В залежності від архітектури, функцій та інфраструктури мережі існує багато способів розділення інформаційного резервування по категоріям [6]. Види інформаційного резервування, розподілені за принципом передачі даних:

1. Повне резервування резервування виконується для усіх даних у системі, які підлягають резервуванню, незалежно від часу їхньої останньої модифікації. Ця категорія є основою для усіх інших категорій;

2. Інкрементне резервування виконується тільки для тих даних, які були змінені в моменту останнього резервування, незалежно від того було це повне або інкрементне резервування. Операція відновлення, зазвичай, включає останнє повне резервування плюс усі наступні додаткові резервування;

3. Диференціальне резервування виконується тільки для тих даних, які змінились з моменту останнього повного резервування. У випадку відновлення, включає в себе повне резервування плюс останнє диференціальне резервування;

4. Синтетичне резервування є результатом процесу створення повного резервування з попередніх повних резервувань, поєднаних з інкрементними резервуваннями. Може бути використане при обмеженій пропускній здатності мережі. Процедура називається синтетичною, оскільки резерв даних не є створеним зі справжніх даних.

У неведеному вище списку описані основні типи резервних копіювань таких, як повне інкрементне, диференціальне та синтетичне. Цей список не повний, оскільки на сьогоднішній день різні програмні продукти пропонують різні комбінації основних типів резервування. У таблиці 1.1 приведено класифікацію інформаційних видів резервування в залежності від функціональних відмінностей.

Таблиця 1.1 – Функціональні відмінності типів резервування

Тип резервування	Данні, що резервуються	Швидкість резервування	Швидкість відновлення	Використання дискового простору
Повне	Всі	Низька	Висока	Високе
Інкрементне	Нові або змінені	Висока	Середня	Низьке
Диференціальне	Нові або змінені з моменту повного резервування	Середня	Висока	Середнє
Синтетичне	Нові або змінені	Висока	Висока	Високе

Актуальність обумовлена потребою у підвищенні ефективності резервування даних мобільних пристроїв з використанням хмарного сховища, оскільки вони є обмеженими в обчислюваних, накопичувальних та енергетичних ресурсах.

1.2 Порівняльний аналіз аналогів

Для вирішення проблеми резервного копіювання даних для мобільних пристроїв існує велика кількість аналогів, які не завжди в повній мірі виконують свої функції.

На основі вимог, що пред’являються до створюваної системи і з урахуванням особливостей предметної області, для проведення аналізу було обрано такі аналоги: FolderSync, DriveSync та Synchronize Ultimate.

FolderSync [7] – платний додаток для синхронізації файлів смартфона за хмарним сховищем, розроблений компанією Tactic Dynamics (рис. 1.1). Цей додаток підтримує синхронізацію з декількома хмарними сервісами, має вбудований файловий менеджер та надає користувачеві можливість вибору частоти резервування.

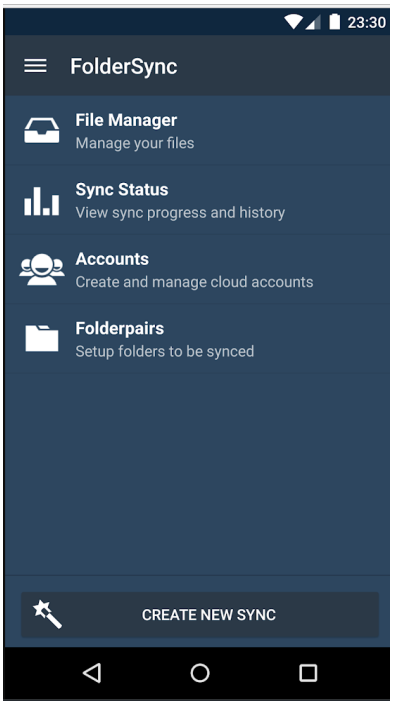


Рисунок 1.1 – Інтерфейс додатку FolderSync

DriveSync [8] – платний (хоча й існує обмежена безкоштовна версія) додаток для перенесення файлів з хмарного середовища на мобільний пристрій (рис. 1.2). Цей додаток має дуже простий але зручний функціонал.

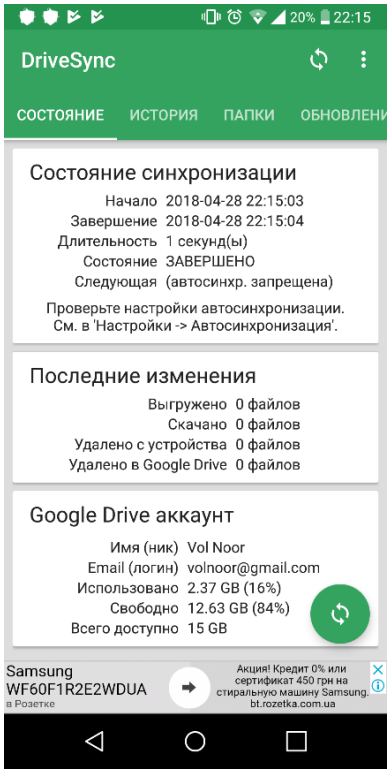


Рисунок 1.2 – Інтерфейс додатку DriveSync

Synchronize Ultimate [9] – безкоштовний додаток, спосіб синхронізації якого базується не на API хмарних сховищ, а на HTTP запитах (рис. 1.3). Такий спосіб синхронізації не має можливості використовувати мережеві ефективно.

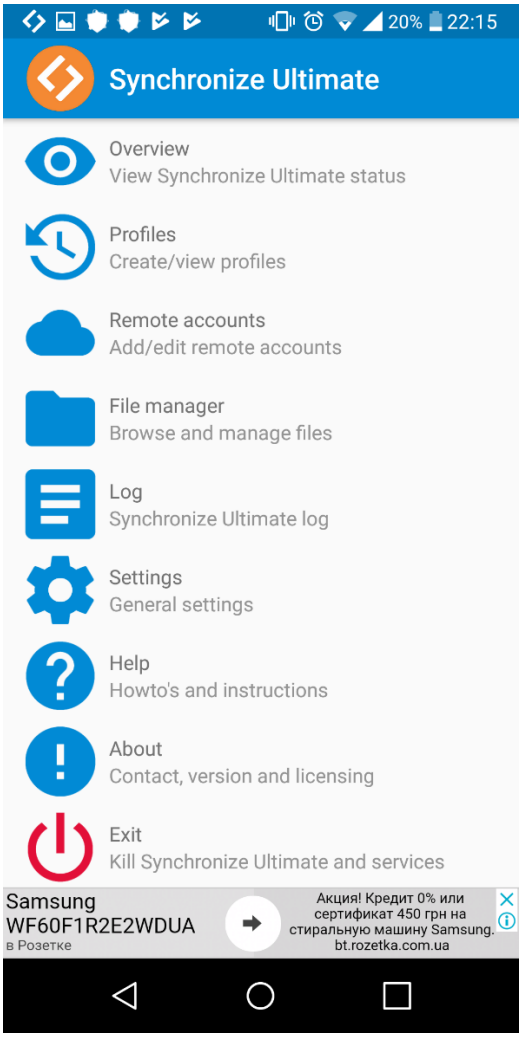


Рисунок 1.3 – Інтерфейс додатку Synchronize Ultimate

Для проведення порівняння виділимо такі ознаки: синхронізація фалів вручну, вибір постійного облікового запису, автоматичне резервування файлів, робота у фоновому режимі, підтримка Google Drive, підтримка Dropbox та вибір частоти резервування.

Проаналізувавши всі аналоги, зведемо результати аналізу до таблиці 1.2.

Таблиця 1.2 – Функціональні характеристики аналогів

Ознака порівняння	FolderSync	DriveSync	Synchronize Ultimate	Simple Backup
Вибір постійного облікового запису	1	1	0	1
Автоматичне резервування даних	1	0	1	1
Робота у фоновому режимі	0	1	1	1
Підтримка Dropbox	0	0	1	1
Вибір частоти резервування	1	0	0	1
Сумарний коефіцієнт	3	2	3	5

Згідно з порівняльним аналізом, показаним у таблиці 1.2, доведено актуальність розробки, оскільки сумарний коефіцієнт розроблюваного додатку перевищує сумарні коефіцієнти аналогів.

1.3 Висновки

В цьому розділі було проаналізовано предметну область та проведено порівняльний аналіз аналогів. В результаті було підтверджено доцільність розробки, оскільки існуючі реалізації мать суттєві недоліки.

2 АНАЛІЗ ЗАСОБІВ РОЗРОБКИ СИСТЕМИ

2.1 Аналіз технології розробки системи

Розробка мобільних додатків – це процес, при якому програмні додатки розроблюються для невеликих портативних пристроїв, таких, як смартфони, планшети або кишенькові персональні комп'ютери (КПК). Ці програмні мобільні додатки можуть бути попередньо встановленими на пристрої під час виробництва, завантаженими користувачем за допомогою різноманітних платформ для розповсюдження програмного забезпечення або являти собою веб-додатки з обробкою інформації на стороні сервера або клієнта (наприклад, JavaScript) для забезпечення можливості їх перегляду використовуючи веб-браузер [10].

Розробка мобільного інтерфейсу користувача, як частина процесу розробки мобільного програмного забезпечення, також має дуже важливе значення при створенні мобільних додатків. Інтерфейс користувача мобільних додатків має суттєві обмеження в розмірах та способі вводу і виводу інформації. Обмеження дизайну пов'язані з особливостями форм-факторів, таких, як розмір екрану пристрою та розмір пристрою. Загалом, мета дизайну для мобільних пристроїв в основному полягає в зрозумілому та зручному для користувача інтерфейсі, тобто інтерфейсі, який враховує обмежену увагу користувача та є орієнтованим на завдання з мінімальним набором функцій.

Критерії вибору платформи розробки зазвичай включають в себе цільові мобільні платформи, існуючу інфраструктуру та навички розробки. При розробці системи, сумісною з більш ніж однією платформою, важливо також зауважити вплив інструменту на досвід користувача. Продуктивність є ще одним важливим критерієм, оскільки дослідження мобільних додатків свідчить про сильне співвідношення між продуктивністю додатку та задоволеністю користувачів. Поряд з продуктивністю та іншими критеріями доступність технології та вимоги проекту можуть вибір між нативними та кросплатформеними середовищами. Як правило, кросплатформені середовища можуть бути використані для розробки під

декілька платформ використовуючи власний контейнер, наприклад HTML, CSS та JavaScript для користувальницького інтерфейсу.

Для розробки мобільних додатків можна виділити такі підходи [11]:

1. Нативні. Нативні додатки є найбільш звичним типом додатків. Вони розробляються під конкретну платформу з використанням мов програмування, прийнятих для цих платформ (наприклад, Swift та Objective-C для операційної системи IOS, Java та Kotlin для операційної системи Android). Ці додатки розробляються з використанням спеціальних інтегрованих середовищ розробки (IDE) для конкретної операційної системи. Обидві компанії Apple та Google надають розробникам власні інструменти для розробки, включаючи елементи інтерфейсу та набір засобів розробки (SDK). В порівнянні з іншими підходами до розробки, цей підхід має найбільше переваг, а саме:

− швидкість роботи додатку та чутливість інтерфейсу;

− найбільша продуктивність;

− можливість розповсюдження додатків в магазинах додатків;

− найкраща інтерактивність, інтуїтивність та плавність роботи при введені та виведенні інформації;

− доступ до усіх можливостей апаратного та програмного забезпечення платформи (камери, мікрофона, акселерометру, телефонної книги і т.п.) та можливість оптимізації продуктивності;

− не обов’язкова наявність Інтернет-з’єднання, хоча це може залежати від функціоналу додатку;

− краща взаємодія з користувачем, оскільки нативні додатки дотримуються стандартів інтерфейсу кожної платформи.

Серед недоліків нативної розробки можна виділити такі:

− складність мов програмування, які використовуються для розробки, що може означати потребу у досвідчених розробниках;

− більша вартість розробки;

− не є найкращим варіантом для простих додатків.

2. Веб-додатки. Веб-додатки являють собою веб-сайти зі зменшеною кількістю інформації та адаптовані для мобільних платформ для покращення функціональності. Веб-додатки, на відміну від звичайних додатків, не вимагають встановлення: для їх відображення використовуються веб-браузери, наприклад Chrome, Safari, або Firefox. Через це вони на займають місця на пристрої користувача. Як правило, ці додатки розроблюються з використанням JavaScript, CSS та HTML5. Для розробників не існує комплекту для розробки, однак є шаблони, з якими розробники можуть працювати. Веб-додатки є дуже простим та швидким для розробки. Проте вони часто є надто простими і не мають функцій, які надають нативні мобільні додатки.

Переваги:

− прості та швидкі для розробки;

− легка підтримка;

− дешеві для розробки;

− розробляються одночасно для всіх платформ, оскільки переглядаються через браузер.

Недоліки:

− необхідність використання браузеру для роботи, що зумовлює незручність використання;

− низька продуктивність;

− нижча інтерактивність та інтуїтивність в порівнянні з нативними додатками;

− відсутність доступу до апаратного забезпечення пристрою.

3. Гібридні додатки. Гібридні додатки працюють на різних платформах та є схожими на нативні додатки. Вони, по суті, являють собою комбінацію нативних та веб-додатків. Користувачі можуть встановлювати їх на своїх пристроях, як нативний додаток, але насправді це веб-додаток. Ці типи додатків побудовані з використанням Javascript, HTML та CSS. Гібридні додатки можуть використовувати функції нативних додатків та мають такі переваги:

− простий процес розробки;

− нижча ціна розробки, ніж у нативних;

− один додаток для всіх платформ;

− відсутня необхідність браузер, на відміну від веб-додатків;

− доступ до апаратного забезпечення пристрою (пам'яті, камери тощо);

− швидка розробка, оскільки існує тільки є одна кодова база для всіх платформ.

Серед недоліків можна виділити такі:

− повільніші, ніж нативні додатки;

− нижча швидкість розробки, ніж у веб-додатків, оскільки існує необхідність у створення обгортки для конкретної платформи;

− менший рівень інтерактивності ніж у нативних додатків.

2.2 Постановка задачі розробки

Після аналізу питання використання технології резервування на мобільних пристроях та аналізу технологій розробки мобільних додатків було визначено завдання, які необхідно виконати згідно з метою роботи.

Метою роботи є дослідження і розробка інтерактивної системи для резервування даних, тому для досягнення поставленої мети роботи необхідно вирішити така задачі:

− проаналізувати особливості розробки системи;

− обґрунтувати вибір мови програмування для реалізації системи;

− обґрунтувати вибір системи керування версіями;

− обґрунтувати вибір інтегрованого середовища розробки;

− розробити інтерфейс системи;

− розробити дизайн розробленого інтерфейсу;

− розробити алгоритм функціонування розроблюваної системи;

− виконати тестування розробленої системи;

− виконати економічне обґрунтування доцільності розробки.

2.3 Вибір засобів реалізації програмного продукту

2.3.1 Вибір мови програмування

На сучасному етапі розвитку комп’ютерних технологій існує велика кількість мов програмування, які використовуються для розробки мобільних додатків. Офіційно, для розробки додатків, сумісних з операційною системою Android з використанням Android SDK можна використовувати такі мови програмування, як Java, C/C++ та Kotlin [12].

Java – об’єктно-орієнтована мова програмування загального призначення, яка є на основі класів [13]. Розроблена компанією Sun Microsystems у 1996 році, хоча розробка почалася в 1990 році. Додатки Java зазвичай транслюються в спеціальний байт-код, тому вини можуть працювати на будь-якій архітектурі, з допомогою віртуальної Java-машини. Достоїнством такого способу виконання є повна незалежність байт-коду від операційної системи та обладнання, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина. Перевагами мови Java є строга типізація, мультиплатформеність та підтримка об’єктно-орієнтованого програмування. Недоліками є велике використання ресурсів комп’ютера та відсутність множинного спадкування.

Мова програмування Java була та залишається основною та найпопулярнішою мовою для операційної системи Android. Вона є незалежною від платформи та має велику кількість інструментів та бібліотек, які полегшують розробку з її використанням.

Kotlin – статично-типізована мова програмування, яка працює поверх віртуальної машини Java (JVM) [14]. Мова Kotlin була у розробці з 2010 року та став доступним спільноті з 2011 року. Автори цієї мови програмування ставили перед собою створення більш лаконічної мови, ніж Java. Kotlin позиціонується розробниками як об’єктно-орієнтована мова промислового рівня а також, мова, яка може змінити мову Java. При цьому мова є повністю сумісною з Java, що дозволяє розробникам поступово перейти з Java на Kotlin.

Kotlin був нещодавно представлений як друга офіційна мова програмування для операційної системи Android, що призвело до швидкого росту популярності цієї мови. Ця мова є повністю сумісною з мовою Java. Різниця між ними полягає лише в тому, що Kotlin вимагає написання меншої кількості коду та є більш зрозумілим для читання.

Мова програмування C++ – мова, яка поєднує властивості як високорівневих, так і низькорівневих мов. Ця мова виникла на початку 1980-х років в результаті створення ряду удосконалень до мови Сі для своїх потреб працівником фірми Bell Labs Б'єрном Страуструпом [15]. C++ підтримує такі парадигми програмування як процедурне програмування, об’єктно-орієнтоване програмування та узагальнене програмування. C++ широко використовується для розробки програмного забезпечення, є одним з найпопулярніших мов програмування.

Перевагами мови C++ є висока сумісність з мовою Сі, підтримка об’єктно-орієнтованого програмування та висока продуктивність. Недоліками є громіздкість коду, наявність небезпечних можливостей, що значно знижають якість програм та погано продуманий синтаксис, що звужує спектр його застосування.

Бібліотеки, написані на C/C++, можуть бути використані в розробці за допомогою Android Native Development Kit (NDK). Ці бібліотеки можуть визиватися з Java коду з використанням стандартних засобів. Однак, згідно з документацією, цей підхід не завжди слід використовувати, оскільки він збільшує складність розробки та не має чітких переваг. NDK слід використовувати в таких випадках:

– потрібно отримати дуже високу продуктивність, або потрібне виконання додатків, які вимагають високої обчислювальної потужності такі, як ігри або симулятори фізики;

– використання вже існуючих C/C++ бібліотек [16].

Функціональні характеристики обраних мов програмування приведені до таблиці 2.1.

Таблиця 2.1 – Функціональні характеристики мов програмування

Ознака порівняння	Java	Kotlin	C++
Автоматичне збирання сміття	1	1	0
Перевантаження операторів	0	1	1
Перевірені виключення	1	0	0
Відсутність необхідності встановлення додаткових залежностей	1	0	0
Статичні змінні	1	0	1
Сумарний коефіцієнт	4	2	2

З таблиці 2.1 видно, що мова програмування Java має найбільше переваг над іншими мовами, тому саме її обрано для реалізації системи.

2.3.2 Вибір системи керування версіями

Система керування версіями (англ. Version Control System, VCS) – програмне забезпечення для полегшення роботи з інформацією, що змінюється [17]. Система керування версіями дозволяє зберігати декілька версій одного і того ж документа, визначати хто і коли зробив ці зміни, та надає можливість повернення до попередніх версій цього документу. Такі системи найбільш широко використовуються при розробці програмного забезпечення для зберігання вихідних кодів розроблюваних програм. Але сфера використання систем керування версіями не обмежується лише розробкою програмного забезпечення: вони можуть з успіхом застосовуватися і в інших областях, в яких ведеться робота з великою кількістю електронних документів, які змінюються.

В інженерії програмного забезпечення керування версіями – це будь-яка практика, яка відстежує та забезпечує контроль над змінами вихідного коду. Розробники програмного забезпечення іноді використовують програмне забезпечення для керування версіями для ведення документації та файлів конфігурації, а також вихідного коду. Під час дизайну, розробки та впровадження програмного забезпечення загальноприйнятним є те, що декілька версій того самого програмного забезпечення можуть бути розгорнуті, а той самий час розробники програмного забезпечення повинні працювати з оновленнями.

Помилки або особливості програмного забезпечення часто присутні лише в певних версіях програмного продукту (через виправлення деяких проблем та створення інших в процесі розробки програми). Тому для пошуку та виправлення помилок дуже важливо мати можливість отримання та запуску різних версій програмного забезпечення, щоб визначити, в якій версії виникає проблема. Можливо також знадобитися одночасно розробити дві версії програмного забезпечення: наприклад, де одна версія має виправлені помилки, але не має нових функцій, тоді як інша – там, де працюють нові функції.

На найпростішому рівні розробники можуть просто зберігати кілька копій різних версій програми та належним чином позначати їх. Цей простий підхід використовувався у багатьох великих програмних проектах. Хоча цей метод може працювати, але він є неефективним тому, що це вимагає відслідковування великої кількості майже однакових копій програми, що вимагає значної кількості самодисципліни з боку розробників і часто призводить до помилок. Для вирішення цієї проблеми були розроблені системи автоматизації керування версіями [18].

При розробці програмного продукту нерідко буває те, що на що над одним проектом одночасно працюють кілька людей. Якщо два розробника змінюють один і той же файл, то один з них може випадково скасувати зміни, внесені іншим. Системи керування версіями можуть відстежувати такі конфлікти і пропонувати їх вирішення. Найпростіше рішення, яке можуть запропонувати більшість систем – автоматично об'єднати (злити) зміни, зроблені різними розробниками. Однак таке рішення, зазвичай, можливо тільки для змінених текстових файлів і за умови того, що змінювались різні (непересічні) частини цього файлу. Таке обмеження пов'язане з тим, що більшість систем керування версіями орієнтовані на підтримку процесу розробки програмного забезпечення, а вихідні коди програм зберігаються в текстових файлах. Якщо автоматичне об'єднання виконати не вдалося, система може запропонувати вирішити проблему вручну.

Сучасні системи керування версіями надають ряд інших можливостей, зокрема:

– контроль прав доступу користувачів – надання або заборона доступу на читання або зміну файлів репозиторію конкретним особам;

– введення журналу змін, в якому користувачі можуть записувати пояснення про внесені ними зміни файлів;

– можливість дізнатись хто і коли додав або змінив конкретний набір рядків у файлі;

– можливість створення різних варіантів одного документа (створення гілки) із загальною історією змін до точки розгалуження і з різними – після неї.

За способом зберігання інформації репозиторію системи керування версіями поділяються на централізовані та розподілені [19].

Централізовані системи керування версіями базуються на ідеї того, що існує центральна копія проекту (наприклад, на сервері, який і виконує більшу частину функцій з керування версіями), і користувачі вносять зміни в цю центральну копію. Більшість сучасних систем керування версіями працюють з набором змін одного або декількох файлів. Централізовані системи керування версіями вирішують проблему зберігання багатьох копій файлів вручну, оскільки система може отримати з центрального репозиторію будь-яку версію файлу за потребою. До розподілених систем керування версіями відносять CVS та Subversion.

За останні декілька років набули популярності розподілені системи керування версіями. Ці системи не залежать від центрального сервера при зберіганні усіх версія усіх файлів репозиторію. Замість цього, кожен користувач має копію репозиторію і має доступ до усієї історії репозиторію у своїй копії. Даний метод може здаватись витратним, але на практиці це не є проблемою. Більшість проектів розробки програмного забезпечення містять у собі лише текстові файли і дисковий простір є таким дешевим, що зберігання великої кількості копій файлів не є дуже обтяжливим. Сучасні системи також стискають файли для того, щоб вони займали менше місця. До розподілених систем керування версіями належать Git та Mercurial.

На сьогоднішній день існує велика кількість систем керування версіями, але для порівняння слід виділити такі: CVS, Subversion (SVN), Git, Mercurial.

CVS є найпопулярнішою і найбільш широко прийнятою системою керування версіями. Після її випуску в 1986 році вона швидко стала де-факто стандартом для систем керування версіями. Вона завоювала популярність в основному через низьку планку входження та її просту систему, що дозволяє зберігати файли та оновлювати їх В даний час активна розробка системи припинена (остання версія була випущена в травні 2008 року), в вихідний код вносяться лише невеликі виправлення. На даний момент CVS є застарілою системою тому, що вона має ряд недоліків та є в наявності більш молоді альтернативні системи керування версіями (наприклад Subversion, Git, Mercurial), в яких немає більшості недоліків CVS [20].

Переваги:

− це перевірена часом і зріла система, яка використовувалася протягом більше трьох десятиліть;

− існує багато інтегрованих середовищ, які використовують CVS.

Недоліки:

− переміщення і перейменування файлів не включено в оновлення версії;

− пропонування символічних посилань на файли передбачає певні ризики для безпеки;

− дуже повільне маркування та розгалуження;

− мала підтримки бінарних файлів.

Subversion (SVN) – ще одна система керування версіями, яка широко використовується. Більшість проектів з відкритим кодом та великі платформи використовують. Через її величезну популярність існує безліч версій та інтегрованих середовищ розробки, які її підтримують. Subversion є вільною централізованою системою керування версіям, офіційно випущеною в 2004 році компанією CollabNet, а з 2010 року є одним із проектів Apache Software Foundation і офіційно називається Apache Subversion. На початку її розробки мала ціль замінити поширену на той момент систему CVS, яка на даний момент є морально застарілою [21].

Переваги:

− нова та значно вдосконалена система на основі CVS;

− процес розгалуження є простішим та швидшим;

− підтримка великою кількістю інтегрованих середовищ розробки.

Недоліки:

– має помилки при перейменування або переміщенні каталогів;

– недостатня кількість команд керування;

– повільніша в порівнянні з іншими системами.

Git швидко набирає популярність за останні роки. Вона швидко завоювала увагу програмістів завдяки різним типам елементів керування та високо-розподіленої системи. Завдяки розподіленій формі контролю без будь-якої основної копії програмного забезпечення, перевагу Git віддають багато проектів з відкритим кодом і системні адміністратори. Проект Git був створений Лінусом Торвальдсом для керування розробкою ядра операційної системи Linux, перша версія була випущена 7 квітня 2005 року. Так, як і більшість інших розподілених систем керування версіями і на відміну від більшості клієнт-серверних систем, кожна директорія Git на кожному комп'ютері є повноцінним репозиторієм з усією історією змін та засобами відстеження змін та є незалежною від Інтернет-з'єднання або центрального сервера [22].

Переваги:

− виняткова швидкість роботи;

− легка та швидка операція розгалуження;

− доступ до дерева історії в автономному режимі;

− має модель однорангової мережі.

Недоліки:

− складна у вивченні;

− погано підходить для розробників, які працюють самостійно;

− обмежена підтримка Windows у порівнянні з Linux.

Mercurial є системою з відкритим кодом, яка вважається ефективною для великих проектів, в яких беруть участь багато розробників та дизайнерів. Mercurial – це високопродуктивна система, що забезпечує оптимальну швидкість. Вона також відома неперевершеною простотою та великою документацією. Розробка

Mercurial була розпочата 19 квітня 2005 року, декілька днів після початку розробки системи Git, та мала з нею однакові наміри. Хоча і Mercurial не був обраний для керування вихіддним кодом Linux, він був використаний великою кількістю компаній, оскільки сервер Mercurial був розроблений спеціально для підтримки великих репозиторіїв, які містять велику кількість проектів [23].

Переваги:

- проста у вивченні (у порівнянні з Git);
- чудова документація;
- високо-розподілена модель;
- високопродуктивна система з великою швидкістю.

Недоліки:

- відсутня можливість об’єднання двох основних гілок;
- недостатньо універсальна для складних маневрів.

Результати порівняння систем керування версіями зведені до таблиці 2.2

Таблиця 2.2 – Функціональні характеристики систем керування версіями

Критерій	CVS	Subversion	Git	Mercurial
Розподіленість	0	0	1	1
Можливість роботи без доступу до мережі Інтернет	0	1	1	1
Кросплатформність	1	1	1	1
Знаходиться у активній розробці	0	1	1	1
Наявність вбудованого графічного інтерфейсу користувача	0	0	1	0
Сумарний коефіцієнт	1	3	5	4

За результатами таблиці 2.2 Git було обрано в якості системи керування версіями для розробки додатку.

2.2.3 Вибір середовища розробки

Інтегроване середовище розробки (IDE) – це програмний продукт, що надає розробникам програмного забезпечення повний набір можливостей для розробки програмного забезпечення [24]. Зазвичай, інтегроване середовище розробки

складається з редактору коду, компілятора, який створює вихідний файл продукту, відлагоджувача, та файлової системи, яка керує файлами проекту. Більшість сучасних IDE мать інтелектуальне завершення коду. Іноді інтегровані середовища розробки мають вбудовані інструменти такі, як системи керування версіями або конструктори графічного інтерфейсу користувача.

Інтегровані середовища розробки розроблені таким чином, щоб максимізувати продуктивність програміста за допомогою складних компонентів із подібними користувальними інтерфейсами. IDE представляє собою єдину програму, в якій відбувається весь процес розробки. Зазвичай, ця програма надає багато можливостей для створення, модифікації, складання, розгортання та налагодження програмного забезпечення. Одна із цілей IDE полягає у поєднанні різноманітних утіліт в одному модулі, який дозволить абстрагуватись від виконання допоміжних задач, тим самим дозволяючи програмісту зосередитись на рішенні особистої задачі і запобігти втраті часу при виконанні типових технічних дій. Таким чином, зменшення часу на налаштування може збільшити продуктивність розробника в тих випадках, коли вивчення IDE є швидшим, ніж налаштування ручне налаштування інструментів. Більш тісна інтеграція всіх завдань розробки має потенціал для підвищення загальної продуктивності, окрім процесів налаштування. Наприклад, код можна постійно аналізувати під час його редагування, надаючи миттєвий відгук при введенні помилок синтаксису. Це може прискорити вивчення нової мови програмування та пов'язаних з нею бібліотек.

Деякі IDE присвячені конкретній мові програмування, що дозволяє встановити набір функцій, який найбільш точно відповідає парадигмам програмування мови. Але також існує багато IDE, які підтримують багато мов програмування.

Серед інтегрованих середовищ розробки, які використовуються для розробки додатків для операційної системи Android, найпопулярнішими є: Android Studio, Eclipse, IntelliJ IDEA та NetBeans.

Android Studio – офіційне (з 2015 року) середовище розробки для Android додатків, яке базується на середовищі IntelliJ IDEA компанії JetBrains та створене

виключно для розробки додатків для операційної системи Android [25]. Воно є заміною середовища Eclipse як основного середовища для створення додатків під Android. Анонсоване та відкрите для раннього доступу 16 травня 2013 року, середовище отримало стабільну версія 1.0 в грудні 2014 року. Android Studio є кросплатформним: воно доступне для використання його на платформах Windows, macOS та Linux. Оскільки середовище базується на середовищі IntelliJ, воно підтримує усі його мови програмування. Це середовище має усі інструменти, які полегшують розробку додатків для операційної системи Android: візуальний редактор розмітки, аналізатор вихідних файлів, емулятор, розумний редактор коду та інструменти профілювання.

Eclipse є найбільш використовуваним середовищем для мови програмування Java [26]. Окрім мови програмування Java, Eclipse підтримує велику кількість інших мов. Воно має базову робочу область і розширювану систему плагінів для налаштування середовище. Одним із таких плагінів був плагін для розробки додатків для Android, який робив Eclipse до 2015 року офіційним середовищем для розробки під цю операційну систему. Eclipse стало доступним з 2001 року, коли компанія IMB випустила його як платформу з відкритим вихідним кодом. Спроектоване для задоволення потреб розробки великих проектів, воно може виконувати різноманітні завдання, такі, як аналітика і дизайн, менеджмент та реалізація продукту, розробка контенту, тестування та документування.

IntelliJ IDEA – інтегроване середовище розробки для мови програмування Java розроблене компанією JetBrains і доступне у двох версіях: безкоштовній та комерційній [27]. Як і середовище Eclipse, підтримує багато мов програмування та підтримує розробку додатків для операційної системи Android за допомогою спеціального плагіна. Перша версія IntelliJ IDEA була випущена в січні 2001 року, і на той момент була одним з самих перших середовищ для мови Java, які мали прогресивні можливості редагування коду та навігації. Плагіни, які доступні для встановлення, широко розширюють можливості цього середовища, роблячи його найпопулярнішим середовищем серед Java розробників.

NetBeans – ще одне відоме середовище для мови програмування Java, яке також підтримує багато інших мов програмування та має плагін для Android розробки [28]. Середовище було незалежно розроблене в кінці 1990-х років та було розміщене як проект з відкритим вихідним кодом після того, як було придбане компанією Sun в 1999 році.

Результати порівняння описаних вище інтегрованих середовищ розробки зведені до таблиці 2.3.

Таблиця 2.3 – Функціональні характеристики інтегрованих середовищ розробки

Критерій	Android Studio	Eclipse	IntelliJ IDEA	NetBeans
Аналізатор продуктивності	1	0	0	0
Автоматичне налаштування проекту	1	0	0	0
Рефакторинг коду	1	1	1	1
Візуальний редактор графічного інтерфейсу користувача	1	1	1	0
Підтримка систем контролю версій	1	1	1	1
Перетворювач векторних зображень	1	0	0	0
Сумарний коефіцієнт	6	3	3	2

З таблиці 2.3 бачимо, що середовище Android Studio найбільше підходить для реалізації проекту оскільки воно має найбільше переваг над аналогами.

2.4 Висновки

У цьому розділі було проаналізовано технології розробки системи, поставлено задачі розробки та обґрунтовано вибір мови програмування, системи керування версіями та інтегрованого середовища розробки. В результаті порівняння різних засобів було обрано мову програмування Java, систему керування версіями Git та інтегроване середовище розробки Android Studio.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Аналіз принципів розробки системи

Додатки для операційної системи Android створюються за допомогою мов програмування Java, Kotlin і C++. Інструменти Android SDK (Software Development Kit – комплект розробки програмного забезпечення) компілюють написаний код разом з усіма необхідними файлами даних та ресурсів в файл APK – програмний пакет Android, який являє собою файл архіву з розширенням .apk. В файлі APK знаходиться все, що необхідно для роботи Android-додатку, і він дозволяє встановити додаток на будь-який пристрій під управлянням операційної системи Android.

Кожен додаток Android виконується у власному ізольованому програмному середовищі та захищений наступними функціями безпеки Android:

– операційна система Android – це багатокористувацька система Linux, в якій кожен додаток є окремим користувачем;

– за замовчуванням, система призначає кожному додатку унікальний ідентифікатор користувача Linux (ідентифікатор використовується тільки системою і є невідомим для додатку). Система встановлює дозволи для всіх файлів у програмі, щоб лише їм можна було отримати доступ до ідентифікатора користувача, призначеного для цього додатка;

– кожен процес має власну віртуальну машину, тому код програми працює ізольовано від інших додатків;

– за замовчуванням кожна програма запускається у власному процесі Linux. Система Android починає процес, коли будь-який компонент додатка повинен бути виконаний, а потім припиняє процес, коли він більше не потрібний, або коли система повинна звільнити пам'ять для інших додатків.

Таким чином, в операційній системі Android реалізовано принцип надання мінімальних прав – кожен додаток за замовчуванням має можливість доступу тільки до тих компонентів, які необхідні для його роботи. Через це додаток немає доступу до недозволених областей системи, що формує безпечне середовище.

Компоненти додатку є необхідними для створення додатку. Кожен компонент представляє собою унікальний елемент структури, який є точною входу для користувача, визначає роботу додатку, або відіграє свою важливу роль. Існує чотири різних типи компонентів [29]:

– операції (Activities);

– служби (Services);

– приймачі широкомовних повідомлень (Broadcast receivers);

– постачальники контенту (Content providers).

Операції представляють собою один екран, який є початковою точкою для взаємодії з користувачем. Кожній операції присвоюється вікно для промальовування відповідного інтерфейсу користувача. Зазвичай, вікно операції відображається на весь екран, однак його розмір може бути меншим, і воно може розміщуватись поверх інших вікон. Android додаток може складатися з декількох операцій. При запуску нової операції, попередня операція зупиняється та зберігається в стеку переходів, а нова також розміщується в стеку і відображається для користувача. Стек переходів відновлює та відображає попередню операцію після того, як поточна була завершена та знищена, що відбувається при закритті поточної операції користувачем. Зазвичай, операції використовуються для виконання різного роду дій, ініційованих користувачем. Операції мають життєвий цикл, який залежить від дій користувача. Протягом життєвого циклу операція може перебувати в одному з трьох станів:

1. Активна та виконується – ця операція знаходиться на передньому плані стеку операцій та є видимою і доступною для взаємодії з користувачем;

2. Призупинена – операція частково або повністю видима, але не доступна для взаємодії з користувачем. Зазвичай, операція призупиняється якщо користувачу представлено діалогове вікно поверх цієї операції.

3. Завершена – операція не видима та недоступна для взаємодії з користувачем. Операції надається статус завершеної в тому випадку, якщо користувачу представлена інша операція. При поверненні до завершеної операції,

вона отримує статус активної (якщо вона не була знищена системою для звільнення ресурсів).

Служби є компонентами, які не мають інтерфейсу користувача та використовуються для виконання тривалих операцій або виконання роботи для інших процесів. Сервіси виконуються у фоновому режимі і є непомітними для користувача, якщо не зв’язані з операціями. Оскільки сервіси виконуються на основному потоці, слід використовувати фоновий потік для виконання задач сервісу для того, щоб не блокувати основний потік, який відповідає за відображення інтерфейсу користувача.

Приймачі широкомовних повідомлень використовуються для прийняття повідомлень від системи по виникнення подій системного. Приймачі можуть отримувати повідомлення від системи навіть тоді, коли додаток не працює. Багато широкомовних повідомлень відправляються системою – наприклад повідомлення, яке свідчить про те, що пристрій має низький заряд батареї або екран пристрою був вимкнений. Додатки також мать можливість відправляти широкомовні повідомлення, наприклад, повідомлення про те, що певна інформація була завантажена на пристрій та доступна для використання її ними. Як і служби, приймачі широкомовних повідомлень не мають інтерфейсу користувача, проте вони можуть створювати візуальні повідомлення, які інформують користувача про виникнення певної події.

Постачальники контенту керують набором даних додатку, які зазвичай зберігаються в базі даних. За допомогою постачальника контенту відбувається, якщо це дозволено, доступ до даних або їх модифікація з інших додатків. Яскравим прикладом постачальника контенту є список контактів користувача: інші додатки можуть робити запити на отримання або редагування змісту контактів. Можна вважати, що постачальники контенту є рівнем абстракції між додатком да базою даних, хоча і мають ширший функціонал.

3.2 Вибір архітектури системи

У більшості випадків, програми для настільних ПК мають одну єдину точку входу з робочого столу чи менеджера запуску програм, в результаті чого відбувається запуск єдиного монолітного процесу. З іншого боку, програми для операційної системи Android мають більш складну структуру.

Типовий додаток для Android містить декілька компонентів (операції, фрагменти, служби, приймачі широкомовних повідомлень, та постачальники контенту), які оголошуються в файлі маніфесту [30]. Потім операційна система використовує цей файл для того, щоб вирішити яким чином інтегрувати додаток у загальний досвід використання пристрою. Зважаючи на те, що правильно написаний додаток для Android містить декілька компонентів і те, що користувачі часто взаємодіють із декількома додатками за короткий проміжок часу, додатки повинні адаптуватися до різних видів робочих процесів та завдань, керованих користувачем.

У будь-який момент використання додатку може бути перерваним телефонним дзвінком або сповіщенням. Після цього переривання користувач розраховує, що зможе повернутися до цього перерваного додатку і відновити роботу з ним. Така поведінка є поширеною на мобільних пристроях, тому додаток повинен правильно організовувати поведінку. Зважаючи на такі умови середовища, компоненти додатку можуть бути запущені у будь-який час та невпорядковано і операційна система або користувач може знищити їх у будь-який час. Через те, що ці подіє є контрольованими, додаток не повинен зберігати інформацію або стан в будь-якому з компонентів і компоненти додатку не повинні залежати один від одного.

Через особливості, описані вище, важливим є питання вибору архітектури системи. В даному випадку, архітектурою можна назвати набір технік та шаблонів проектування, які використовуються для розробки системи.

Шаблони проектування – це загальні рішення для поширених проблем дизайну систем [31]. Шаблони проектування не є кінцевим дизайном, який трансформується напряму у вихідний або машинний код, вони є описом або

заготовкою для того, яким чином вирішувати проблему. Зазвичай, шаблони проектування є формалізованими найкращими практиками та можуть розглядатися як структурний підхід при розробці системи. Існує досить багато шаблонів архітектури додатків, найвідомішими серед яких є три класичних представники:

- MVC (Model-View-Controller);
- MVP (Model-View-Presenter);
- MVVM (Model-View-ViewModel).

Усі ці шаблони мають однакову ідею – структурування коду проекту таким чином, щоб він був розділений на різні загальні шари. Кожен шар несе свою відповідальність, і саме через це проект стає модульним, розділені частини коду легше підлягають тестуванню і додаток стає досить гнучким для постійних змін.

Model-View-Controller (Модель-Вигляд-Контролер) – самий перший шаблон, який почав використовуватись для розробки додатків для операційної системи Android. Він пропонує розділення коду на три шари:

- Model (Модель) – шар даних. Відповідає за обробку бізнес-логіки та зв’язок із рівнями мережі та бази даних;
- View (Вигляд) – шар користувацького інтерфейсу. Цей шар є візуалізацією даних із шару Моделі;
- Controller (Контролер) – логічний шар, отримує повідомлення про поведінку користувача і оновлює Модель (за потребою).

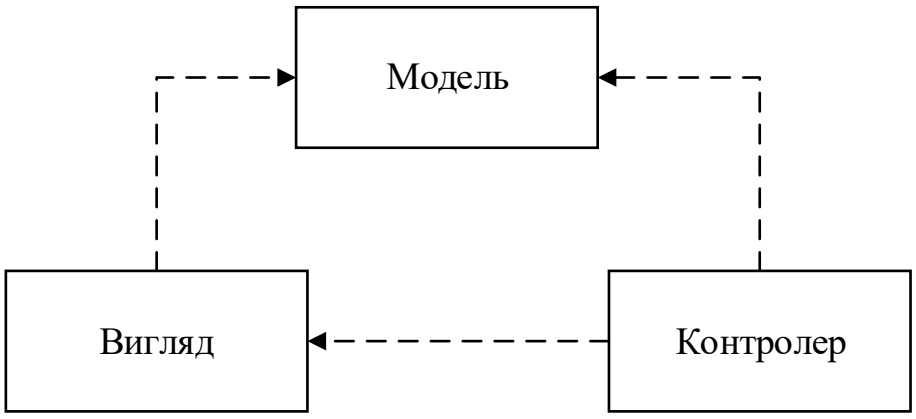


Рисунок 3.2 – Схема шаблону Модель-Вигляд-Контролер

Зі схеми шаблону, зображеної на рисунку 3.2, бачимо, що і Контролер, і Вигляд залежать від Моделі: Контролер через оновлення даних, Перегляд через отримання даних. Модель є відокремленою і її можна тестувати незалежно від інтерфейсу користувача.

Model-View-Presenter (Модель-Вигляд-Представник) – один з найпопулярніших архітектурних шаблонів, який набув широкого поширення і досі є рекомендованим для використання. Має три таких шари:

- Model (Модель) – шар даних. Відповідає за обробку бізнес-логіки та зв'язок із рівнями мережі та бази даних;
- View (Вигляд) – шар користувацького інтерфейсу. Відображає дані та повідомляє Представника про дії користувача;
- Presenter (Представник) – отримує дані з Моделі, керує логікою інтерфейсу користувача, керує станом Вигляду, вирішує які дані показувати і реагує на дії користувача, отримані від Вигляду.

Рисунок 3.3 – Схема шаблону Модель-Вигляд-Представник

Схема, зображена на рисунку 3.3, показує, що Вигляд і Представник тісно пов'язані між собою. Цей шаблон має значний, але контрольований недолік – Представник може бути розширеним до великих розмірів якщо не дотримуватись принципу однієї відповідальності.

Model-View-ViewModel (Модель-Вигляд-Модель Вигляду) – шаблон, рекомендований командою Android. На відміну від шаблону MVP, Модель Вигляду не має посилання на Вигляд (рис. 3.4). Також маж три шари:

- Model (Модель) – абстрагує джерело даних. Модель Вигляду працює з Моделлю для отримання і збереження даних;
- View (Вигляд) – шар, який інформує Модель Вигляду про дії користувача;

– ViewModel (Модель Вигляду) – містить потоки даних для Вигляду.

Рисунок 3.4 – Схема шаблону Модель-Вигляд-Модель Вигляду

Зведемо результати порівняння цих трьох шаблонів проектування до таблиці 3.1

Таблиця 3.1 – Характеристики шаблонів проектування

Шаблон	Залежність від Android API	Складність XML	Здатність до тестування	Модульність
MVC	Висока	Низька	Низька	Ні
MVP	Низька	Низька	Висока	Так
MVVM	Низька або відсутня	Середня або висока	Висока	Так

Отже, в результаті порівняння було обрано MVP шаблон, оскільки він краще підходить для масштабування та підтримки додатків, дозволяє розділяння системи на модулі та має чітке розмежування між шарами.

3.3 Розробка графічного інтерфейсу користувача

Розробка дизайну і інтерфейсу мобільного додатку є важливим етапом створення продукту: від цього залежить зручність його використання та сприйняття додатку користувачем.

Інтерфейс користувача – це зовнішня оболонка додатку, яка надає користувача можливості для керування схованими від нього елементами. Головна мета інтерфейсу – забезпечення зручності, простоти та ефективності роботи з інформацією: документами, базами даних, графікою.

Через те, що мобільні пристрої мають невеликий за розмірами екран, при проектуванні інтерфейсу для них не можна користуватись тими ж правилами, що і

для інтерфейсів додатків для персональних комп'ютерів [32]. Серед вимог до інтерфейсу мобільних додатків можна виділити наступні:

1. Через те, що управління сучасними телефонами з сенсорним екраном здійснюється за допомогою дотику, площа дотику пальця значно перевищує розміри курсора миші, тому інтерфейс не повинен містити дрібних елементів. Також, дрібні елементи погано помітні на дрібному екрані, та можуть стати причиною натискання користувача не на той елемент, що призведе до зниження зручності використання програми та виникнення небажаних наслідків;

2. Найбільш часто використовувані та важливі елементи повинні мати достатній розмір для того, щоб виділятись серед інших та їх слід розміщувати в центрі екрану;

3. Не слід перевантажувати невеликий простір дисплею пристрою великою кількістю елементів. Слід намагатися створювати інтерфейс максимально функціональний та лаконічний та дружній для користувача інтерфейс;

4. Розмір елементів інтерфейсу повинен бути таким, щоб користувач міг його розпізнати з відстані не менше 30 см;

5. На відміну від персональних комп’ютерів, на мобільних телефонах вікна займають весь простір екрану. Це означає, що необхідно оптимізувати весь інтерфейс, прибрати всі зайві елементи, згрупувати схожі за функціоналом, та розмістити їх у додаткових вікнах.

Для побудови інтерфейсу в операційній системі Android користувача використовуються макети. Макет визначає візуальну структуру інтерфейсу користувача, який використовується в операції [33]. Існує два способи визначити макет:

– визначення елементів користувацького інтерфейсу в XML;

– створення екземплярів елементів під час виконання.

Платформа Android надає можливість гнучкого використання будь-якого з цих способів для оголошення користувальницького інтерфейсу програми та його управління. Наприклад, спочатку можна використовувати макети в XML, які

містять елементи екрану, які будуть відображатися, а потім можна додати в додаток код, який змінює стан об'єктів на екрані під час виконання.

Перевага оголошення користувальницького інтерфейсу у файлі XML полягає в тому, що таким чином можна більш ефективно відокремити представлення додатку від коду, який керує його поведінкою. Опис користувальницького інтерфейсу знаходяться поза межами коду вашої програми. Це означає, що можна змінити або адаптувати інтерфейс без необхідності вносити зміни в вихідний код і повторно його компілювати. Наприклад, можна створювати різні файли XML-макету для екранів різних розмірів і різних орієнтацій екрана, а також для різних мов. Крім того, оголошення макета в XML спрощує візуалізацію структур користувальницького інтерфейсу, завдяки чому відлагодження проблем також стає простішою.

Ієрархічна модель інтерфейсу розроблюваного додатку містить три фрагменти (рис. 3.5). Перший – відображає список директорій для яких відбувається резервування. Другий – додавання директорії для резервування. Третій – фрагмент, який використовується при редагуванні опцій резервування вже створеної директорії.

Рисунок 3.5 – Ієрархічна структура інтерфейсу додатку

Для кожного фрагмента створено окремі макети в XML, оскільки цей підхід спрощує процес розробки інтерфейсу.

Інтерфейс фрагменту списку директорій зображено на рис. 3.6.

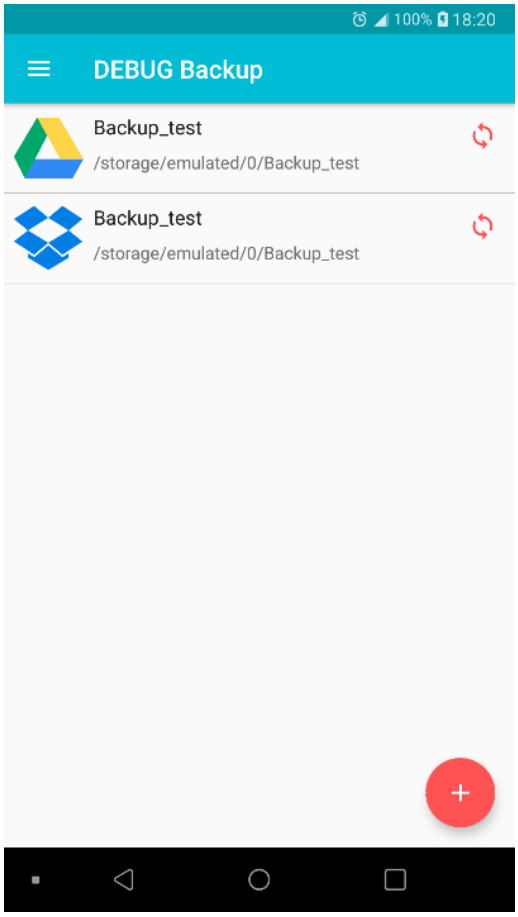


Рисунок 3.6 – Інтерфейс фрагменту списку директорій

Графічна схема інтерфейсу фрагменту списку директорій зображена на рис. 3.7. Інтерфейс фрагменту списку директорій містить такі елементи:

1. Панель дій;
2. Список директорій для резервування, графічна схема яких зображена на рис. 3.8, містять:
    a. Логотип хмарного сервісу;
    b. Назва директорії;
    c. Шлях до директорії;
    d. Кнопка «Резервувати».
3. Кнопка «Додати директорію»;
4. Панель навігації.

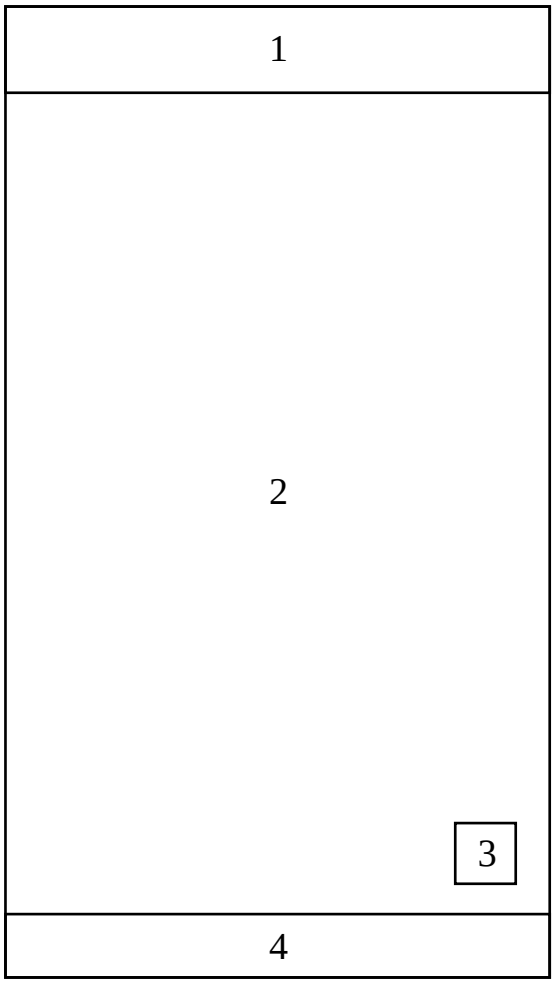


Рисунок 3.7 – Графічна схема Інтерфейсу фрагменту списку директорій

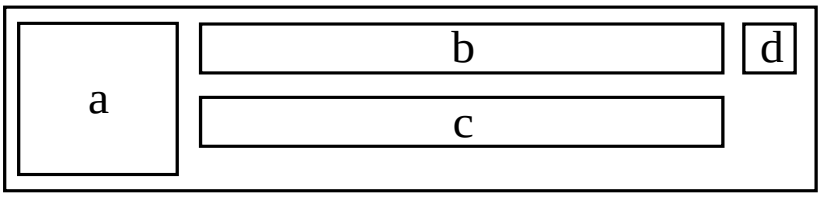


Рисунок 3.8 – Графічна схема елементу списку директорій

Інтерфейс фрагменту додавання директорії зображено на рис. 3.9. Вибір хмарного сервісу та частоти резервування було реалізовано за допомогою вертикального випадного списку, який дозволяє вибрати один з попередньо визначених параметрів.

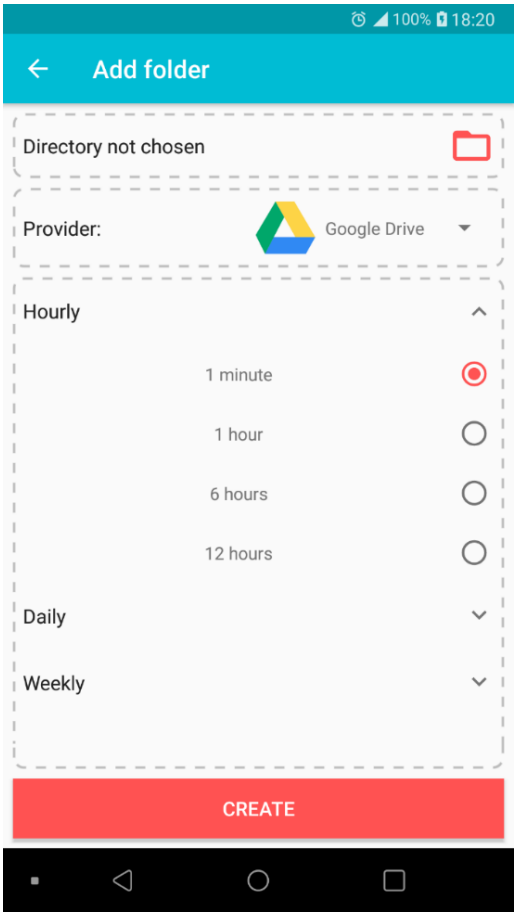


Рисунок 3.9 – Інтерфейс фрагменту додавання директорії

Графічна схема інтерфейсу фрагменту додавання директорії зображена на рис. 3.10. Інтерфейс фрагменту додавання директорії містить такі елементи:

1. Кнопка «Назад»;
2. Панель дій;
3. Назва директорії;
4. Шлях до директорії;
5. Кнопка вибору директорії;
6. Випадний список за вибором хмарного сервісу;
7. Випадний список за вибором частоти резервування;
8. Кнопка «Створити»;
9. Панель навігації.

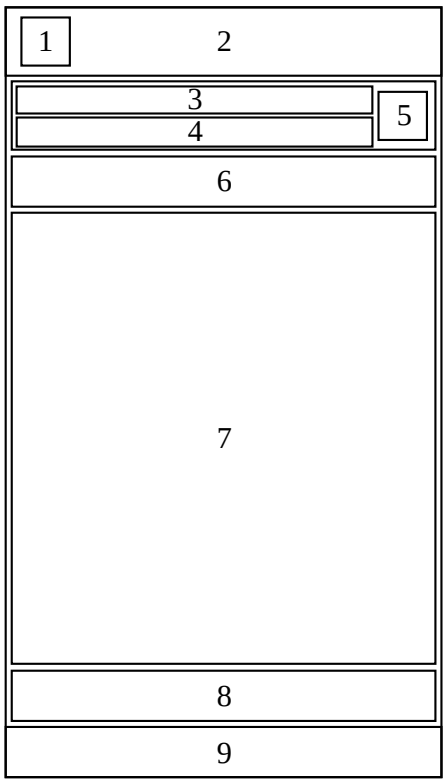


Рисунок 3.10 – Графічна схема інтерфейсу фрагменту додавання директорії

Інтерфейс фрагменту редагування зображено на рис. 3.11.

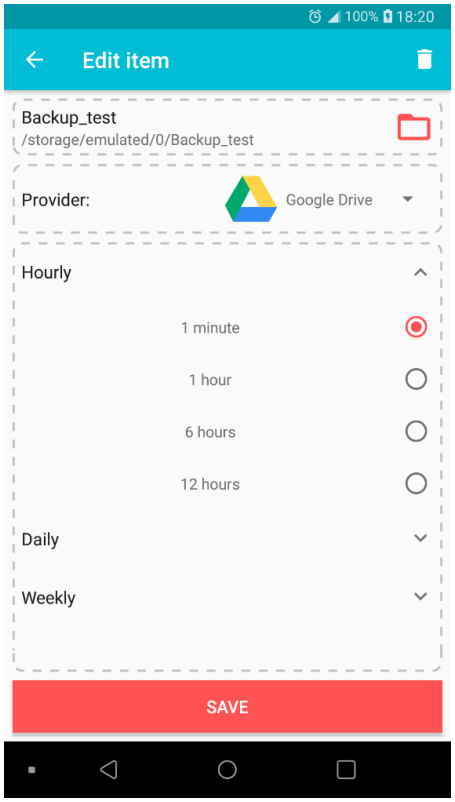


Рисунок 3.11 – Інтерфейс фрагменту редагування

Графічна схема інтерфейсу фрагменту редагування директорії зображена на рис. 3.12. Інтерфейс фрагменту редагування директорії містить такі елементи:

1. Панель дій;
2. Кнопка «Видалити»;
3. Назва директорії;
4. Шлях до директорії;
5. Кнопка вибору директорії;
6. Випадний список за вибором хмарного сервісу;
7. Випадний список за вибором частоти резервування;
8. Кнопка «Зберегти»;
9. Панель навігації.

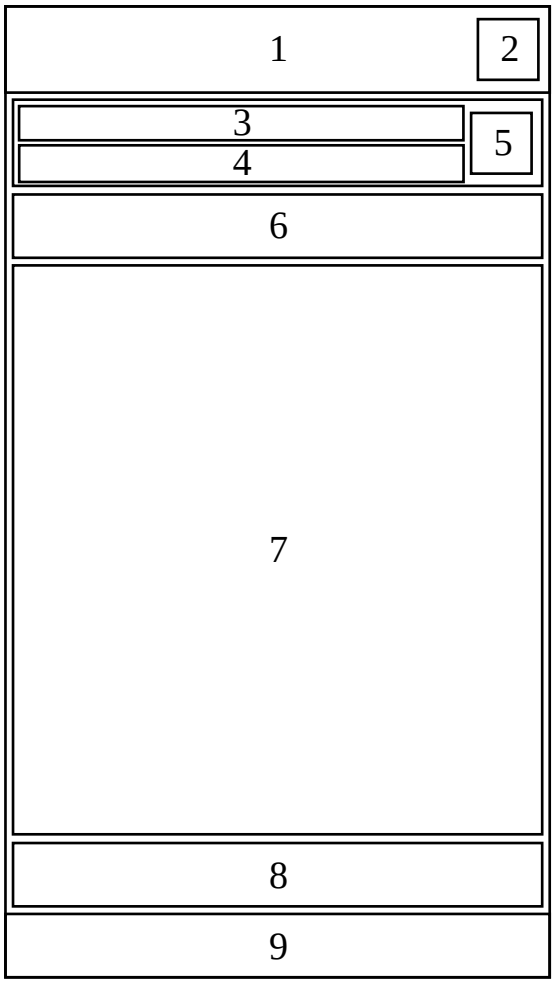


Рисунок 3.12 – Графічна схема інтерфейсу фрагменту редагування

3.4 Розробка алгоритму роботи системи

Алгоритм – це набір інструкцій, які описують порядок виконання дій для досягнення результату вирішення задачі за визначену кількість дій.

Для створення алгоритму потрібно мати такі вхідні дані:

− початковий стан об’єкта (повний набір вхідних даних завдання);

− кінцевий стан об’єкта (мета, задля досягнення якої створюється алгоритм;

− набір дій, необхідних для виконання для досягнення мети, які розуміє виконавець і може їх виконати.

Для розробки складних алгоритмів існує два основних метода:

1. Метод послідовної деталізації завдання («зверху-вниз»). Цей метод полягає в тому, що складна вхідна задача розбивається на підзадачі, кожна з яких розглядається, аналізується та вирішується окремо. Якщо які-небуть з підзадач є складними, вони також розбиваються на підзадачі. Процес розбиття на підзадачі триває до тих пір, поки складність підзадач не зведеться до елементарної. Після вирішення окремих підзадач, результати їх вирішення збираються в єдиний алгоритм, який і є розв’язком вхідної задачі;

2. Складальний метод («знизу-вверх»). Цей метод полягає у створенні великої кількості програмних модулів, які реалізують рішення типових задач. Спочатку вирішуються конкретні задачі, а потім їх результати об’єднуються в більше загальне рішення.

При розробці алгоритму, слід дотримуватись таких вимог [34]:

1) детермінованість (визначеність) означає, що у кожен момент часу кожен наступний крок роботи алгоритму повинен однозначно визначатись станом системи. Це означає, що алгоритм повинен видавати один і той самий результат (відповідь) для одних і тих самих початкових даних;

2) зрозумілість – алгоритм повинен включати в себе тільки ті команди, які доступні для розуміння виконавця і входять в його систему команд;

3) результативність – завершеність алгоритму з певними результатами;

4) завершеність – алгоритм повинен завершувати роботу і видавати результат за кінцеве число кроків при заданих коректно початкових даних;

5) дискретність – алгоритм повинен представляти процес вирішення завдання у вигляді послідовного виконання деяких простих кроків. При цьому для виконання кожного кроку алгоритму потрібен кінцевий відрізок часу, тобто перетворення вихідних даних у результат здійснюється в часі дискретно;

6) масовість (універсальність) – алгоритм повинен бути застосований до різних наборів початкових даних;

7) алгоритм не містить помилок, якщо він дає правильні результати для будь-яких допустимих початкових даних;

8) алгоритм не містить помилок, якщо від видає правильні результати для будь-яких допустимих значень.

В залежності від початкових умов завдання, мети та шляхів її вирішення, визначення дій виконавця поділяються наступним чином [35]:

− лінійний алгоритм – алгоритм, набір команд (вказівок) якого, виконуються послідовно в часі один за одним;

− розгалужений алгоритм – алгоритм, який містить хоча б одну умову, в результаті перевірки якої може здійснюватися поділ на кілька паралельних гілок алгоритму;

− циклічний алгоритм – алгоритм, що передбачає багаторазове повторення однієї і тієї ж дії (одних і тих же операцій) над новими вихідними даними. Цикл програми – послідовність команд, яка може виконуватися багато разів (для нових початкових даних) до задоволення деякої умови;

− гнучкі алгоритми, наприклад стохастичні, тобто імовірнісні та евристичні;

− імовірнісний алгоритм дає програму рішення задачі кількома шляхами або способами, що призводять до ймовірного досягненню результату;

− структурна блок-схема, граф-схема алгоритму – графічне зображення алгоритму у вигляді схеми пов'язаних між собою за допомогою стрілок (ліній переходу) блоків – графічних символів, кожен з яких відповідає одному кроку алгоритму;

− евристичний алгоритм (від грецького слова «еврика») – алгоритм, що використовує різні розумні міркування без строгих обґрунтувань.

Загальний алгоритм роботи додатку є розгалуженим, оскільки містить декілька умов, в результаті перевірки яких здійснюється поділ на кілька паралельних гілок алгоритму. Загальний алгоритм роботи програми зображений на рисунку 3.13.

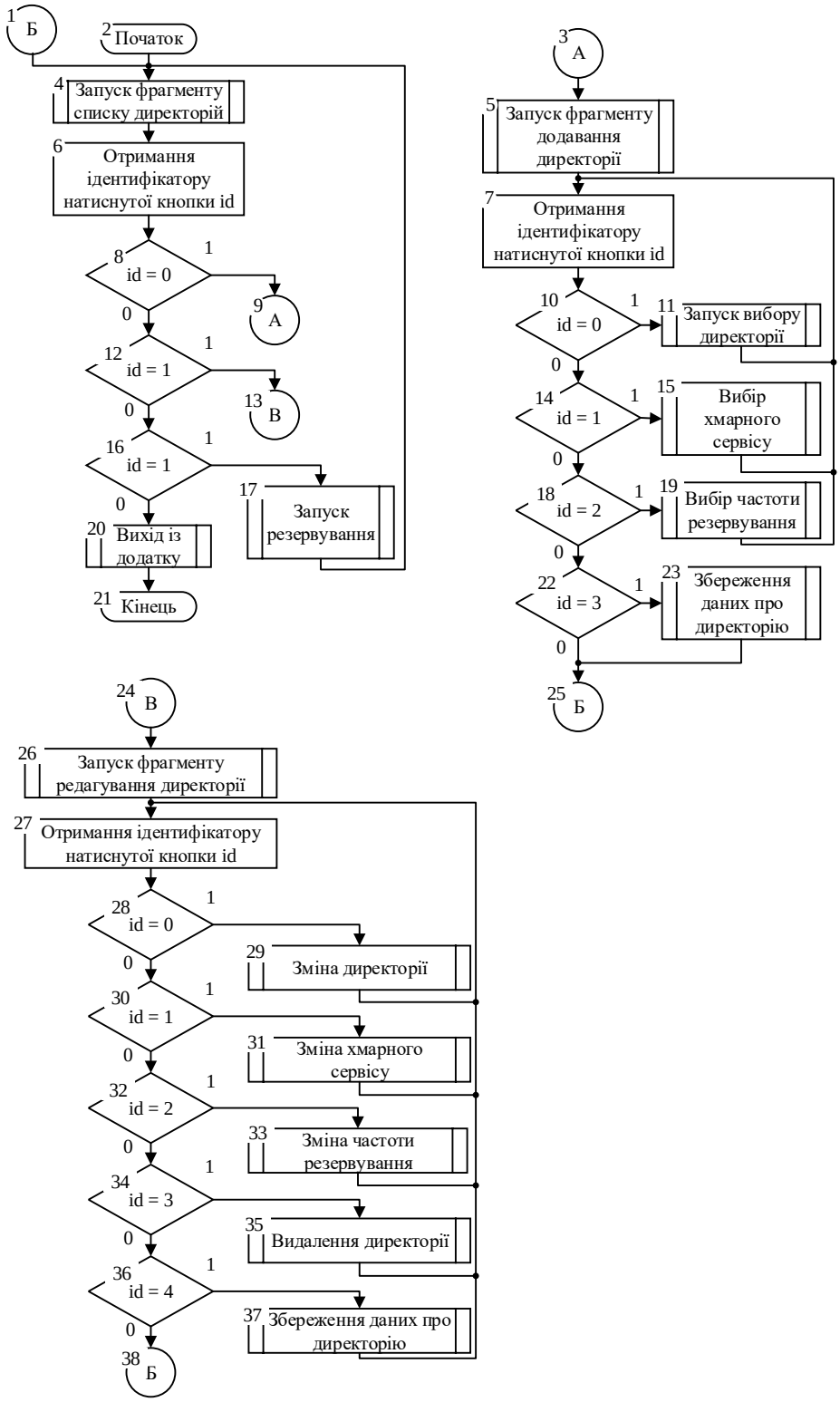


Рисунок 3.13 – Блок-схема загального алгоритму роботи програми

Алгоритм резервування даних, блок-схема якого зображена на рисунку 3.14, використовує рекурсивний обхід папок для резервування.

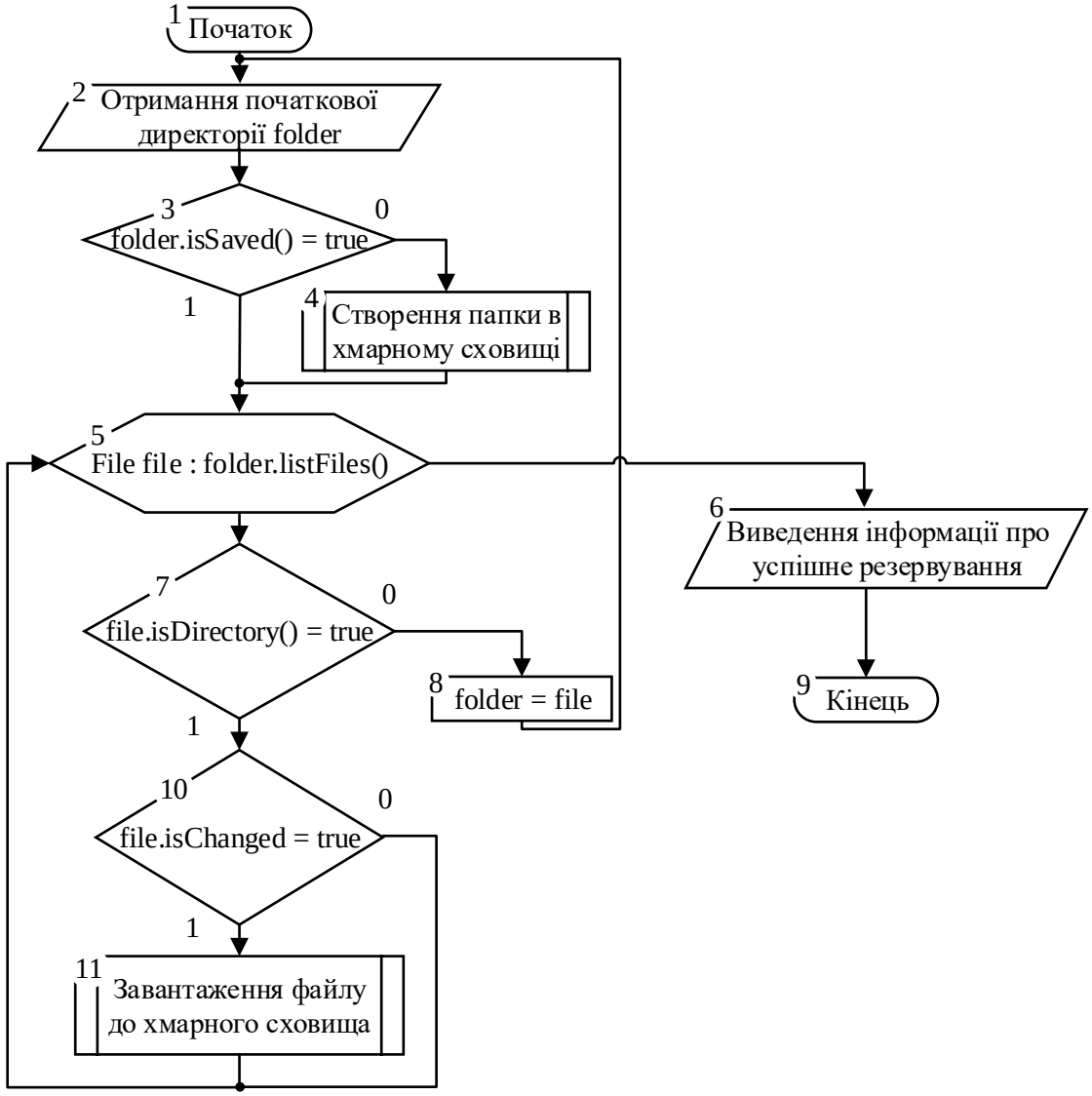


Рисунок 3.14 – Блок-схема алгоритму резервування даних

3.5 Висновки

В цьому розділі було проаналізовано принципи розробки системи, описано особливості створення інтерфейсу користувача для мобільних пристроїв, розроблено структуру інтерфейсу розроблюваного додатку та основні алгоритми для його роботи. В результаті було виділено три фрагменти, які відповідатимуть за функціонування усього програмного засобу та розроблено такий інтерфейс, який буде простим та зручним для використання.

4 ТЕСТУВАННЯ

Тестування програмного забезпечення – це процес технічного дослідження, який виконується на вимогу замовника і призначений для отримання інформації про якість програмного продукту відносно контексту, в якому він має використовуватись; процес аналізу або експлуатації програмного забезпечення з метою виявлення дефектів [36].

Тестування передбачає "аналіз" та "експлуатацію" програмного продукту. Діяльність тестування, яка пов'язана з аналізом результатів розробки програмного забезпечення, називається статичним тестуванням. Вона передбачає перевірку програмних кодів, контроль та перевірку програми без запуску на комп'ютері. Діяльність тестування, що передбачає експлуатацію програмного продукту, називається динамічним тестуванням. Динамічне та статичне тестування доповнюють одне одного.

Тестування програмного коду – це процес виконання програмного коду, спрямований на виявлення існуючих в ньому дефектів. Під дефектом тут розуміється ділянка програмного коду, виконання якої за певних умов призводить до несподіваної поведінки системи (тобто поведінки, не відповідає вимогам). Несподівана поведінка системи може призводити до збоїв у її роботі і відмов, в цьому випадку говорять про суттєві дефекти програмного коду. Деякі дефекти викликають незначні проблеми, що не порушують процес функціонування системи, але трохи ускладнюють роботу з нею. В цьому випадку говорять про середні або малозначні дефекти.

Мета застосування процедури тестування програмного коду – мінімізація кількості дефектів (особливо істотних) в кінцевому продукті. Тестування само по собі не може гарантувати повну відсутність дефектів у програмному коді системи. Однак, у поєднанні з процесами верифікації та валідації, спрямованих на усунення суперечливості і неповноти проектної документації (зокрема вимог на систему), правильно організоване тестування дає гарантію того, що система задовольняє вимогам і веде себе відповідно до них у всіх передбачених ситуаціях.

Програмний продукт є якісним, якщо:

– під час роботи користувача з програмним продуктом виникає невелика кількість відмов;

– програмний продукт надійний, а це означає, що його використання рідко викликало аварійні відмови;

– програмний продукт задовольняє вимогам більшості користувачів.

4.1 Аналіз методів тестування

В залежності від доступу розробника тестів до коду програми розрізняють тестування «білого ящика» та тестування «чорного ящика» [37].

Основна ідея в тестуванні системи як чорного ящика полягає в тому, що всі матеріали, які доступні тестувальнику (вимоги на систему, що описують її поведінку, і сама система, працювати з якою він може), використовуються лише для дії на їх входи деяких зовнішніх впливів і спостерігання результату на виходах. Всі внутрішні особливості реалізації системи приховані від тестувальника, таким чином, система являє собою "чорний ящик", правильність поведінки якого по відношенню до вимог і належить перевірити.

З точки зору програмного коду чорний ящик може представляти з собою набір класів (або модулів) з відомими зовнішніми інтерфейсами, але недоступними вихідними текстами.

Основне завдання тестувальника для даного методу тестування полягає в послідовній перевірці відповідності поведінки системи вимогам. Крім того, тестувальник повинен перевірити роботу системи в критичних ситуаціях, які відбувається в разі подання невірних вхідних значень. В ідеальній ситуації всі варіанти критичних ситуацій повинні бути описані у вимогах на систему, і тестувальнику залишається лише придумувати конкретні перевірки цих вимог. Проте в реальності в результаті тестування зазвичай виявляється два типи проблем системи: невідповідність поведінки системи вимогам та неадекватна поведінка системи в ситуаціях, не передбачених вимогами.

Звіти по обох типах проблем документуються і передаються розробникам. При цьому проблеми першого типу зазвичай викликають зміну програмного коду, набагато рідше – зміну вимог. Зміна вимог в даному випадку може знадобитися через їх суперечливості (кілька різних вимог описують різні моделі поведінки системи в одній і тій самій ситуації) або некоректності (вимоги не відповідають дійсності).

Проблеми другого типу однозначно вимагають зміни вимог через їх неповноту – у вимогах явно пропущена ситуація, яка веде до неадекватної поведінки системи. При цьому під неадекватною поведінкою може розумітися як повний крах системи, так і взагалі будь-яка поведінка, не описана у вимогах.

Тестування чорного ящика називають також тестуванням за вимогами, тому, що це єдине джерело інформації для побудови тест-плану.

При тестуванні системи як білого ящика тестувальник має доступ не тільки до вимог до системи, її входів і виходів, а й до її внутрішньої структури, тобто бачить її програмний код.

Доступність програмного коду розширює можливості тестувальника тим, що він може бачити відповідність вимог ділянок програмного коду і визначати тим самим, чи на весь програмний код існують вимоги. Програмний код, для якого відсутні вимоги, називають кодом, не покритим вимогами. Такий код є потенційним джерелом неадекватної поведінки системи [38]. Крім того, прозорість системи дозволяє поглибити аналіз її ділянок, що викликають проблеми, часто одна проблема нейтралізує іншу, і вони ніколи не виникають одночасно.

4.2 Особливості тестування системи

Тестування мобільних додатків відбувається в середовищі розробки з використанням емулятора. Після цього додаток тестується на пристрої. Емулятори є простим способом тестувати додаток на мобільному телефоні, не використовуючи його фізично. Нижче представлений список доступних інструментів для тестування додатків серед найпопулярніших мобільних операційних систем:

1) Google Android Emulator. Android Емулятор запускається на Windows як окремий додаток без необхідності повністю завантажувати та встановлювати Android SDK.

2) Офіційний Android SDK Emulator. Включає в себе емулятор мобільного пристрою, який реалізує всі апаратні і програмні особливості типового пристрою.

3) MobiOne. MobiOne Developer це mobile Web IDE для Windows допомагає розробнику програмувати, тестувати, налагоджувати упаковувати і впроваджувати мобільні веб-додатки на пристрої, такі як iPhone, BlackBerry, пристрої на Android і Palm Pre.

4) TestiPhone. Заснований на веб-браузері симулятор для швидкого тестування веб-додатків для iPhone. Працює з використанням Internet Explorer 7, Firefox 2 і Safari 3.

5) iPhoney. Надає точну середу веб-браузера, розроблена Safari. Може бути використана для розробки веб-сайтів для iPhone. Не є емулятором iPhone. iPhoney запускається тільки на Mac OS X 10.4.7 і вище.

6) BlackBerry Simulator. Існує безліч офіційних емуляторів BlackBerry. З будь-яким з них можлива перевірка того, як ПО, екран, клавіатура пристрої будуть працювати з додатком.

Для платформи Android найпопулярнішими емуляторами є: Bluestacks, Android SDK Emulator, Andy, Youwave, Oracle VM VirtualBox, GenyMotion. Нижче наведено їх переваги та недоліки.

Bluestacks [39] на даний момент є одним з найвідоміших емуляторів для ПК завдяки досить-таки простому інтерфейсу і непоганому арсеналу можливостей. З плюсів можна назвати наступні:

– можливість роботи в повноекранному режимі;

– установка необмеженої кількості андроїд-додатків в оболонку;

– отримання root-прав, що дозволить працювати з емулятором як з повністю розблокованим гаджетом;

– синхронізація з пристроєм Android, що дозволить завантажити і встановити ігри та програми з вашого смартфона в програму;

– установка 3d Android ігор на комп'ютер;

– можливість управління android-додатками за допомогою клавіатури;

– сумісність з дротяними контролерами;

– можливість грати в онлайн-ігри.

До явних мінусів емулятора можна віднести:

– некоректна робота в Windows x64;

– може зустрічатися проблема при запуску через конфлікти з антивірусом;

– складність у роботі з онлайн-додатками.

Andy головний конкурент Bluestacks на ринку емуляторів андроїд. І не даремно, оскільки програма має низку переваг:

– згідно порівняльному аналізу, Andy перевершує конкурента завдяки більш стабільній роботі ігор;

– повноцінна підтримка онлайн-додатків на ПК;

– можливість використання андроїд-гаджета, як ігровий контролер;

– швидкодія в роботі навіть на слабких ПК, яка знову ж направлено на ігрову складову програми;

– настройка віртуальної машини Andy: обсяг оперативної пам'яті, необхідний додатком, кількість ядер процесора і т.д.;

– коректна робота з Windows x64.

До мінусів програми можна віднести такі як:

– інтерфейс може здатися досить-таки складним зважаючи на обширний функціонал додатку;

– проблеми з використанням клавіатури ПК в додатках.

Одна з переваг AVD при тестуванні пристроїв –можливість задавати довільні значення для розширення і щільності пікселів екрана. Також Android SDK Emulator має й інші переваги:

– можливість протестувати додаток на різних версіях ОС Android, на пристроях з різними типами дисплея;

– різні налаштування, необхідні для тестування, наприклад, зміна орієнтації екрану;

– емуляція SD-карти.

З недоліків можна виділити такі:

– великий проміжок часу між натисканням кнопки «Run» і запуском програми на емуляторі;

– емулятор працює дуже повільно.

Youwave відмінний емулятор Android для Windows, який не так відомий, як емулятори Andy або Bluestacks, але також має низку переваг:

– малі системні вимоги ПК, що дозволить запустити емулятор навіть на слабкому комп'ютері;

– підтримка навіть старих версій Android, починаючи з 2.3;

– інтуїтивний інтерфейс.

З явних мінусів слід виділити відсутність підтримки української мови.

Oracle VM VirtualBox відрізняється від вищеназваних емуляторів завдяки величезному арсеналу інструментів. Також крім версії для ПК існує додаток для Windows-планшетів. Програма орієнтована більше на розробників додатків. З відмінних якостей можна виділити такі:

– можливість встановити відразу декілька віртуальних систем Android на ПК;

– великий функціонал;

– стабільність при роботі на різних версіях Windows.

До мінусів слід віднести:

– велике споживання ресурсів комп'ютера;

– складність в роботі для звичайних користувачів.

GenyMotion – ще один з відомих емуляторів, який чудово підійде розробникам ігор і програм для Android на ПК. Програму варто виділити завдяки таким якостям:

– велика кількість інструментів для роботи;

– хороша продуктивність;

– підтримка великої кількості ігор, також присутня можливість грати в них онлайн;

– стабільність в роботі на ПК.

Проте ця програма, як і деякі її аналоги, відмовляється працювати в середовищі Windows x64.

Проаналізувавши всі переваги та недоліки емуляторів, вирішено використовувати Android SDK Emulator, так як він найкраще підходить для розробки додатків на ОС Windows x64.

4.3 Тестування системи

Тестування потрібне для того, щоб впевнитись в коректності роботи додатку. Тестування слід виконувати на реальному пристрої, а не емуляторі тому, що емулятори лише імітують роботу операційної системи і не дають можливість перевірити функціональність як кінцевий користувач. Також, тільки на реальних пристроях можна перевірити продуктивність розроблюваного додатку за допомогою інструментів середовища розробки.

Результати тестування зведені у таблицю 4.1.

Таблиця 4.1 – Результати тестування

Дія	Очікуваний результат	Фактичний результат
1. Запуск додатку	Відкриття фрагменту зі списком директорій	Був відкрий фрагмент зі списком директорій
2. Додавання директорії	Запуск фрагменту додавання директорії	Був запущений фрагмент додавання після натискання кнопки «Додати»
3. Додавання директорії для резервування	Після натиснення кнопки «Створити», повинен бути запущений фрагмент зі списком директорій та процес резервування для вибраної директорії	Після натиснення кнопки «Створити», був запущений фрагмент зі списком директорій та процес резервування для вибраної директорії

Продовження таблиці 4.1 – Результати тестування

Дія	Очікуваний результат	Фактичний результат
4. Примусове виконання резервування	Після натиснення кнопки «Запустити», повинен бути запущений процес резервування для вибраної директорії	Після натиснення кнопки «Запустити», був запущений процес резервування для вибраної директорії
5. Редагування створеної директорії	Після натиснення на директорію у списку повинно бути запущено редагування. Після збереження зміни повинні одразу вступити в силу	Після натиснення на директорію у списку було запущено редагування. Після збереження зміни одразу вступили в силу
6. Видалення створеної директорії	Після натиснення кнопки «Видалити» директорія повинна бути видалена	Після натиснення кнопки «Видалити» директорія була видалена
7. Частотне виконання резервування	Резервування для директорії повинно бути виконане згідно обраної частоти	Резервування для директорії виконувалось згідно обраної частоти

Протестуємо запуск додатку. Для цього потрібно відкрити додаток. Після запуску з’являється фрагмент з повідомленням про відсутність даних (рис. 4.1).

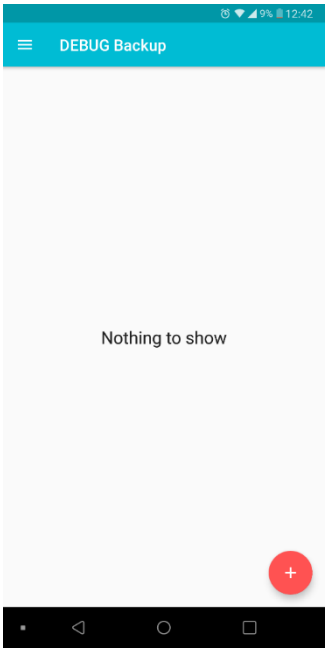


Рисунок 3.1 – Фрагмент з повідомленням про відсутність даних

Після натискання кнопки «Додати» з’являється фрагмент додавання директорії (рис. 4.2).

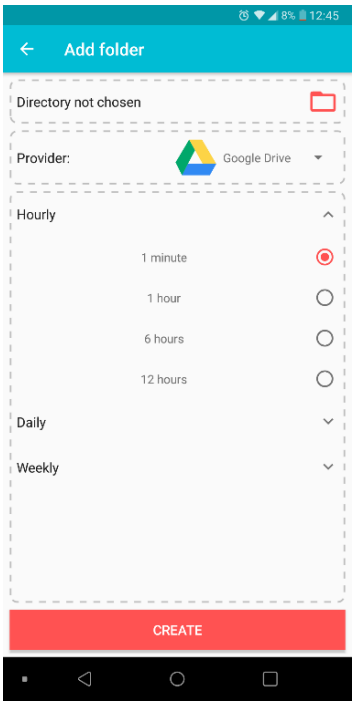


Рисунок 4.2 – Фрагмент додавання директорії

Після вибору директорії та натискання кнопки «Створити» відбувся перехід до фрагменту зі списком директорій (рис. 4.3).

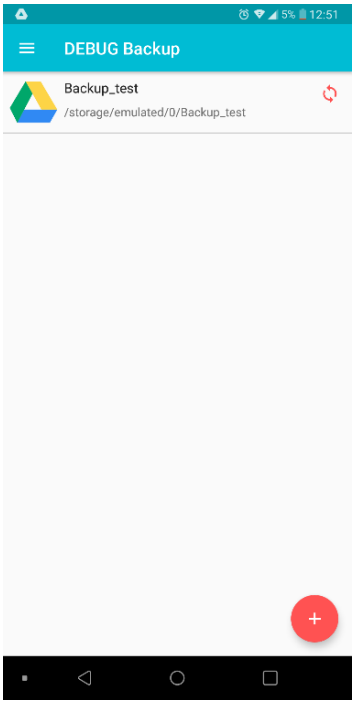


Рисунок 4.3 – Фрагмент зі списком директорій

Також був запущений процес резервування, який виконувався згідно з заданою частотою (рис. 4.4).

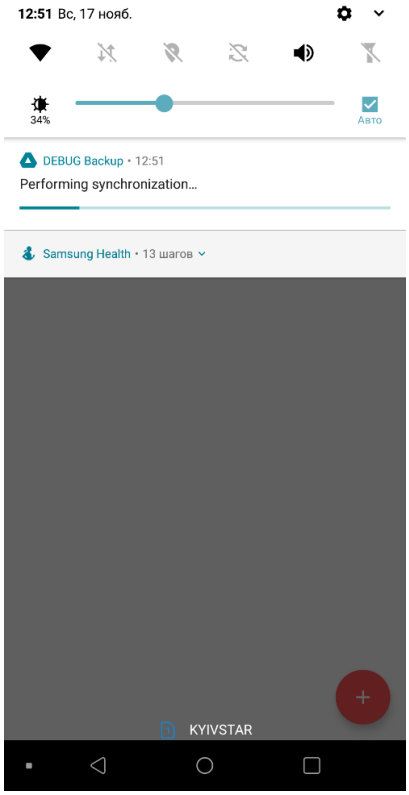


Рисунок 4.4 – Повідомлення про виконання резервування

В результаті натискання кнопки «Запустити» було показано повідомлення про виконання процесу резервування (рис. 4.4), в результаті якого данні директорії було збережено у хмарному сховищі (рис. 4.5).

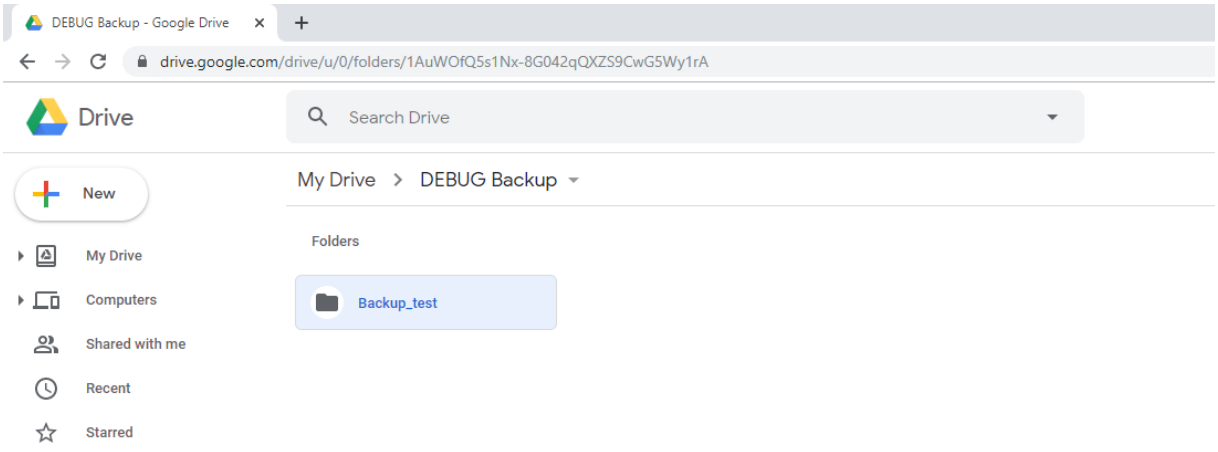


Рисунок 4.4 – Вікно браузера з хмарним сховищем та збереженою директорією

Після натиснення на директорію у списку, вікно редагування було запущено (рис 4.5) Після внесення змін та натискання кнопки «Зберегти» зміни були внесені. При натисканні кнопки «Видалити» директорію було видалено.

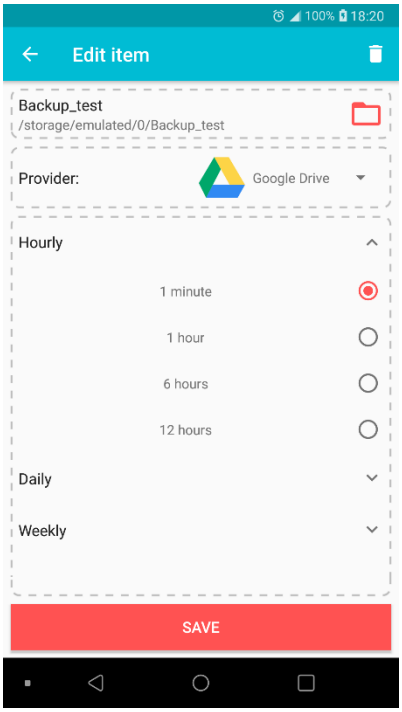


Рисунок 4.5 – Редагування директорії

Отже, розроблений програмний додаток працює вірно.

Тестування програмного продукту показало повну відповідність поставленому технічному завданню.

4.4 Висновки

Одним із важливих етапів розробки додатку є тестування, так як воно є гарантією якості розроблюваного додатку.

В ході створення даного розділу було розглянуто різні стратегії тестування, та визначено, що для даного додатку найоптимальнішим є тестування «чорного ящика».

При проведенні тестування, отримана повна відповідність вхідних даних і вихідних результатів. Помилок при роботі програми не виявлено і тому вона підтверджує нормальний режим роботи додатку.

5 ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки [40]. Для проведення технологічного аудиту було залучено 2-х незалежних експертів. Такими експертами будуть Рейда О. М. та Катєльніков Д. І..

Здійснюємо оцінювання комерційного потенціалу розробки за 12-ма критеріями за 5-ти бальною шкалою.

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.1.

Таблиця 5.1 – Результати оцінювання комерційного потенціалу розробки

Критерії	Експерт	
	Рейда О. М.	Катєльніков Д. І.
	Бали, виставлені експертами:	
1	4	4
2	4	3
3	3	4
4	4	3
5	3	3
6	4	4
7	4	3
8	3	4
9	4	3
10	4	4
11	3	3
12	4	4
Сума балів	$СБ_1 = 44$	$СБ_2 = 42$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 43$	

Отже, з отриманих даних таблиці 5.1 видно, що нова розробка має високий рівень комерційного потенціалу.

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько-технологічної роботи

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою 5.1:

$$З_о = \frac{М}{Т_р} \cdot t, \quad (5.1)$$

де М – місячний посадовий оклад конкретного розробника;

$Т_р$ – кількість робочих днів у місяці, $Т_р = 21$ день;

Т – число днів роботи розробника, t = 45 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 5.2.

Таблиця 5.2 – Розрахунки основної заробітної плати

Працівник	Оклад М, грн.	Оплата за робочий день, грн.	Число днів роботи, t	Витрати на оплату праці, грн.
Науковий керівник	5000	238,1	8	1904,8
Інженер-програміст	3500	166,66	45	7499,9
Всього:				9404,7

Розрахуємо додаткову заробітну плату:

$$З_{дод} = 0,1 \cdot 9404,7 = 940,47 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 37,5…40% від суми їхньої основної та додаткової заробітної плати (формула 5.2):

$$Н_{зп} = (З_о + З_р) \cdot \frac{\beta}{100}, \quad (5.2)$$

$$Н_{зп} = (785,7 + 7857) \cdot \frac{36,3}{100} = 3755,3 \text{ (грн.)}.$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за формулою 5.3:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12}, \quad (5.3)$$

де Ц – балансова вартість обладнання, грн;

$Н_а$ – річна норма амортизаційних відрахувань (для програмного забезпечення 25%);

Т – Термін використання (Т = 3 міс.).

Таблиця 5.3 – Розрахунок амортизаційних відрахувань

Найменування програмного забезпечення	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн
Персональний комп’ютер	9000	25	3	562,5
Всього:				562,5

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою 5.4:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i, \quad (5.4)$$

де n – кількість комплектуючих;

$Н_і$ – кількість комплектуючих і-го виду;

$Ц_і$ – покупна ціна комплектуючих і-го виду, грн;

$К_і$ – коефіцієнт транспортних витрат (приймемо $К_і = 1,1$).

Таблиця 5.4 - Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Флешка	шт.	150	1	150
Пачка паперу	уп.	100	1	100
Ручка	шт.	5	1	5
Всього з урахуванням транспортних витрат				280,5

Витрати на силову електроенергію розраховуються за формулою 5.5:

$$В_е = В \cdot П \cdot Ф \cdot К_п \,, \tag{5.5}$$

де В – вартість 1 кВт-години електроенергії (В = 1,66 грн/кВт);

П – установлена потужність комп'ютера (П = 0,6кВт);

Ф – фактична кількість годин роботи комп'ютера (Ф = 200 год.);

$К_п$ – коефіцієнт використання потужності ($К_п < 1$, $К_п = 0,8$).

$$В_е = 1,66 \cdot 0,6 \cdot 200 \cdot 0,8 = 163,2 \text{ (грн.)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300%) від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто (формула 5.6):

$$В_{ін} = (1..3) \cdot (З_о \cdot З_р). \quad (5.6)$$

Отже, розрахуємо інші витрати:

$$В_{ін} = 1 * (9404,7 + 940,47) = 10345,17 \text{ (грн)}.$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$В = З_о + З_д + Н_{зп} + А + К + В_к + І_в = 9404,7 + 940,47 + 3755,3 + 562,5 + 280,5 + 163,2 + 10345,17 = 25481,54 \text{ (грн.)}.$$

Розрахуємо загальну вартість наукової роботи $В_{заг}$ за формулою 5.7:

$$В_{заг} = \frac{В_{ін}}{\alpha}, \quad (5.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях, α = 1.

$$В_{заг} = \frac{25481,54}{1} = 25481,54$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою 5.8:

$$ЗВ = \frac{В_{заг}}{\beta}, \quad (5.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{25481,54}{0,9} = 28279,82 \text{ (грн.)}.$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. Це забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності.

Оцінка зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою 5.9:

$$\Delta П_i = \sum_{1}^{n} (\Delta П_я \cdot N + П_я \Delta N)_i, \quad (5.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення інформаційної технології зменшаться на 30 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 30 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 200 користувачів, протягом другого року – на 150 користувачів, протягом третього року – 100 користувачів. Реалізація інформаційної технології до впровадження результатів наукової розробки складала 1000 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 400 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_1 = 30 \cdot 1000 + (400 + 30) \cdot 200 = 116000 \text{ грн.}$$

Протягом другого року:

$$\Delta П_2 = 30 \cdot 1000 + (400 + 30) \cdot (200 + 150) = 180500 \text{ грн.}$$

Протягом третього року:

$$\Delta П_3 = 30 \cdot 1000 + (400 + 30) \cdot (200 + 150 + 100) = 223500 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $E_{абс}$ вкладених інвестицій розраховується за формулою 5.10:

$$E_{абс} = (ПП - PV), \quad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки 2, 3,4.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, рисунок 5.1.

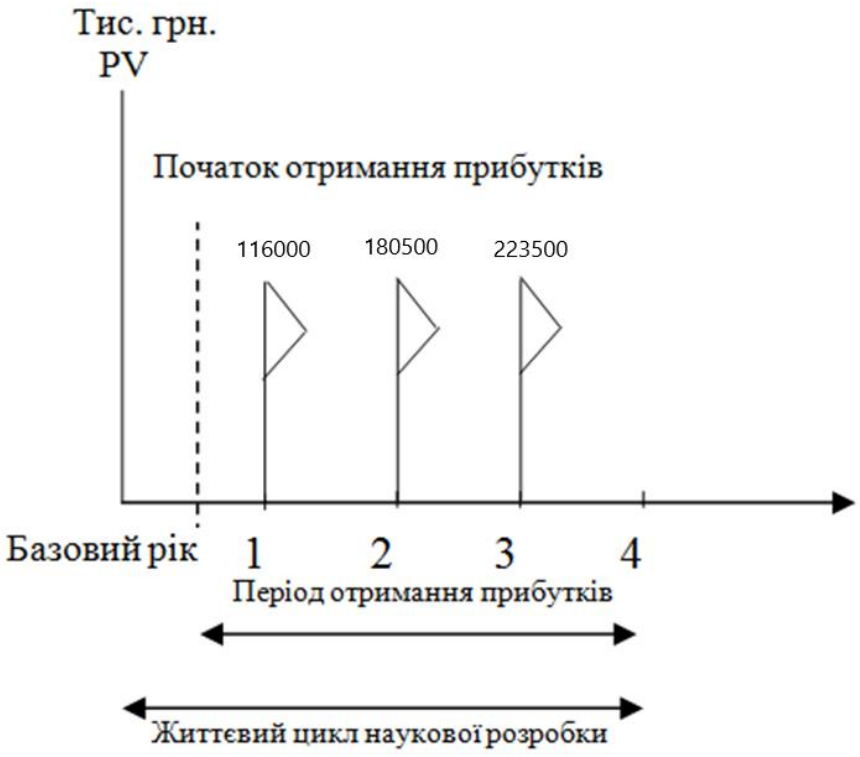


Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою 5.11:

$$ПП = \sum_{1}^{m} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{28279,82}{(1+0,1)^0} + \frac{116000}{(1+0,1)^2} + \frac{180500}{(1+0,1)^3} + \frac{223500}{(1+0,1)^4} = 412413,4 \text{ (грн.)}$$

Тоді розрахуємо $Е_{абс}$:

$$Е_{абс} = 412413,4 - 28279,82 = 384133,58.$$

Оскільки $Е_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою 5.12:

$$Е_в = \sqrt[T]{1+\frac{Е_{абс}}{PV}} - 1 \quad (5.12)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$Е_в = \sqrt[3]{1 + \frac{384133,58}{28279,82}} - 1 = 1,44 \text{ або } 144\ \%.$$

Далі, розраховану величина $Е_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках, в 2019 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0,2 + 0,1 = 0,3.$$

Оскільки $Е_в$ = 144% > $\tau_{мін}$ = 0,3 = 30%, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $Т_{ок}$ розраховується за формулою:

$$Т_{ок} = \frac{1}{Е_в} = \frac{1}{1,44} = 0,69 \text{ років}.$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

5.5 Висновки

Результати проведених розрахунків дають можливість зробити висновок про доцільність розробки та впровадження нашої наукової роботи. Це підтверджують такі показники як:

– абсолютна ефективність вкладених інвестицій, яка дорівнює 412413,4 грн., що є більшим за 0 і вказує на те, що інвестор може бути зацікавленим у нашій розробці;

– відносна ефективність наукової розробки становить 144%, що є вищим за мінімальну ставку дисконтування (30%), тому вкласти кошти у нашу розробку є вигідніше, ніж покласти кошти на депозит;

– термін окупності вкладених у реалізацію наукового проекту інвестицій складе 0,69 року, що є менше 5-ти і вказує на швидку окупність вкладених інвестицій.

Крім того, розраховано, що наукова розробка приноситиме підприємству додатковий прибуток протягом 3-х років за рахунок покращення її якості порівняно з існуючими аналогами.

ВИСНОВКИ

У магістерській кваліфікаційній роботі розроблено інтерактивну систему інкрементного резервування даних як мобільний додаток.

В результаті роботи було досягнуто таких цілей:

1. Обґрунтовано доцільність розробки;
2. Проаналізовано технології розробки системи;
3. Розроблено метод підвищення швидкодії резервування даних;
4. Розроблено програмний продукт;
5. Виконано тестування програмного додатку;
6. Обґрунтовано економічну доцільність розробки.

Для розробки мобільного додатку було проаналізовано предметну область дослідження та порівняно найбільш популярні існуючі аналоги (FolderSync, DriveSync та Synchronize Ultimate), визначено їх переваги та недоліки. Проведено огляд мобільних додатків і визначено клас мобільного додатку, що розроблено. Проведено варіантний аналіз засобів реалізації мобільних додатків і обґрунтовано вибір мови програмування, системи керування версіями та інтегрованого середовища розробки.

Для розробки системи розглянуто базові методи розробки мобільних додатків, розроблено структуру графічного інтерфейсу користувача та алгоритми функціонування системи.

Проведено тестування, доведено ефективність і правильність функціонування та відповідність до технічного завдання.

Розроблений мобільний додаток використовується для резервування даних та має підвищену швидкодію та цілісність резервування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Number of smartphone users worldwide from 2014 to 2020 (in billions). [Електронний ресурс]: Режим доступу до матеріалу: URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide. – Назва з екрану.

2. Mobile cloud storage. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Mobile_cloud_storage. – Назва з екрану.

3. Cloud Storage Comparison: iCloud Drive vs. Dropbox vs. Google Drive vs. OneDrive. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://www.intego.com/mac-security-blog/cloud-storage-comparison-icloud-drive-vs-dropbox-vs-google-drive-vs-onedrive/. – Назва з екрану.

4. The problems with cloud storage services for mobile phones. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://www.networkworld.com/article/2982326/cloud-storage/the-problems-with-cloud-storage-services-for-mobile-phones.html. – Назва з екрану.

5. Основні поняття, визначення і класифікація методів резервованих ТЗ. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://web.posibnyky.vntu.edu.ua/fksa/1vasilevskyj_normuvannya_pokaznykiv_nadijnosti_tehnichnyh_zasobiv/52.htm. – Назва з екрану.

6. Types of backup explained: Full, incremental, differential and mirror. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://searchdatabackup.techtarget.com/feature/Full-incremental-or-differential-How-to-choose-the-correct-backup-type. – Назва з екрану.

7. FolderSync – Apps on Google Play. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://play.google.com/store/apps/details?id=dk.tacit.android.foldersync.lite. – Назва з екрану.

8. Autosync for Google Drive – Apps on Google Play. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://play.google.com/store/apps/details?id=com.ttxapps.drivesync. – Назва з екрану.

9. Synchronize Ultimate. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://play.google.com/store/apps/details?id=com.icecoldapps.synchronizeultimate. – Назва з екрану.

10. Mobile app development. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Mobile_app_development. – Назва з екрану.

11. A Guide to Mobile App Development: Web vs. Native vs. Hybrid. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/. – Назва з екрану.

12. Android software development. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Android_software_development. – Назва з екрану.

13. Java (programming language). [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Java_(programming_language). – Назва з екрану.

14. Kotlin (programming language). [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Kotlin_(programming_language). – Назва з екрану.

15. Страуструп Б. Язык программирования C++. Специальное издание: учебное пособие / Б. Страуструп. – Москва : Бином, 2011. – 1136 с.

16. Getting Started with the NDK. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://developer.android.com/ndk/guides/. – Назва з екрану.

17. Version control. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Version_control. – Назва з екрану.

Why use a Version Control. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://www.git-tower.com/learn/git/ebook/en/desktop-gui/basics/why-use-version-control. – Назва з екрану.

19. What is version control: centralized vs. DVCS. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://www.atlassian.com/blog/software-teams/version-control-centralized-dvcs. – Назва з екрану.

20. Concurrent Versions System. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Concurrent_Versions_System. – Назва з екрану.

21. Apache Subversion. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Apache_Subversion. – Назва з екрану.

22. Git. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Git. – Назва з екрану.

23. Mercurial. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Mercurial. – Назва з екрану.

24. Integrated development environment. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki /Integrated_development_environment. – Назва з екрану.

25. Android Studio. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Android_Studio. – Назва з екрану.

26. Eclipse (software). [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Eclipse_(software) . – Назва з екрану.

27. IntelliJ IDEA. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/IntelliJ_IDEA. – Назва з екрану.

28. NetBeans. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/NetBeans. – Назва з екрану.

29. Application Fundamentals. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://developer.android.com/guide/components/fundamentals ?hl=en. – Назва з екрану.

30. Guide to app architecture | Android Developers. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://developer.android.com/jetpack/docs /guide. – Назва з екрану.

31. Software design pattern. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Software_design_pattern. – Назва з екрану.

32. Методологія проектування та інструментарій для створення мобільних додатків. [Електронний ресурс]. – Режим доступу до матеріалу: URL: http://www.kpi.kharkov.ua/archive/Наукова_періодика/vestnik/новые_решения_в_современных_технологиях/МЕТОДОЛОГІЯ_ПРОЕКТУВАННЯ_ТА_ІНСТРУМЕНТАРІЙ.pdf. – Назва з екрану.

33. Макеты. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://developer.android.com/guide/topics/ui/declaring-layout?hl=ru. – Назва з екрану.

34. Алгоритми. [Електронний ресурс]. – Режим доступу до матеріалу: URL: http://ru.wikipedia.org/wiki/Алгоритм. – Назва з екрану.

35. Виды алгоритмов. Види алгоритмів. [Електронний ресурс]: Режим доступу до матеріалу: URL: https://profmeter.com.ua/communication/learning /course/course19/lesson775/. – Назва з екрану.

36. Software testing. [Електронний ресурс]. – Режим доступу до матеріалу: URL: https://en.wikipedia.org/wiki/Software_testing. – Назва з екрану.

37. Степанченко И.В. Методы тестирования программного обеспечения: учебное пособие / И.В. Степанченко. – Волгоград: ВолгГТУ, 2006. – 74с.

38. Методологія проектування та інструментарій для створення мобільних додатків. [Електронний ресурс]. – Режим доступу до матеріалу: URL: http://www.kpi.kharkov.ua/archive/Наукова_періодика/vestnik/новые_решения_в_современных_технологиях/МЕТОДОЛОГІЯ_ПРОЕКТУВАННЯ_ТА_ ІНСТРУМЕНТАРІЙ.pdf. – Назва з екрану.

39. Тестування. [Електронний ресурс]. – Режим доступу до матеріалу: URL: http://arhiv-statey.pp.ua/index.php?newsid=26947. – Назва з екрану.

40. Грабовецький Б. Є. Економіка підприємства. / Б. Є. Грабовецький, Т.М. Пілявоз –Вінниця: ВНТУ, 2009 –248 с.

41. Big Data processing: methods, models and information technologies: monograph. - edited by Oleg I. Pursky. - Shioba GmbH., Steyr, Austria, 2019. - 234 p.

ДОДАТОК А. Технічне завдання

ВНТУ

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., проф. О.Н. Романюк
“_____” _____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу зі спеціальності 121 – Інженерія програмного забезпечення
студенту групи 1ПІ-18м Горовому Євгенію Вікторовичу

Керівник магістерської кваліфікаційної роботи:
_____к. т. н., доц., О. М. Рейда
"____"_____2019 р.
Виконав:
_____студент гр. 1ПІ-18м Є. В. Горовий
"____"_____2019 р.

Вінниця ВНТУ 2019

1.1 Найменування та галузь застосування

Розробка інтерактивної системи для інкрементного резервування даних. Згідно отриманого завдання кінцевий програмний продукт може використовуватись кінцевими користувачами.

1.2 Підстава для проведення робіт

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № _____ від _____.

1.3 Мета та призначення роботи

Метою роботи є дослідження і розробка інтерактивної системи для резервування даних.

Для досягнення поставленої мети в роботі необхідно вирішити такі завдання:

1. Обґрунтування доцільності розробки;
2. Аналіз технології розробки системи;
3. Програмна реалізація системи резервування даних;
4. Тестування системи;
5. Економічне обґрунтування доцільності розробки.

1.4 Технічні вимоги

- операційна система – Android;
- мова програмування – Java;
- середовище розробки – Android Studio;
- тип резервування – інкрементне.

1.5 Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка;
– лістинги програми.

1.6 Економічні показники

– річний економічний ефект – не більше 120000 грн;
– експлуатаційні витрати – не більше 30000 грн.

1.7 Стадії і етапи розробки

Завдання на проектування видане 7 жовтня 2019 року. Проектування та дослідження повинно бути завершеним до 6 грудня 2019 року.

Обґрунтування доцільності розробки	07.10 – 27.10
Аналіз засобів розробки системи	28.10 – 8.11
Програмна реалізація системи	9.11 – 20.11
Тестування	21.11 – 1.12
Економічне обґрунтування доцільності розробки	1.12 – 6.12

1.7 Порядок контролю і приймання

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

ДОДАТОК Б. Ілюстративний матеріал

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Завідувач кафедри ПЗ, д.т.н., професор _____ О. Н. Романюк

Науковий керівник, к.т.н., доц. кафедри ПЗ _____ О. М. Рейда

Рецензент, к.т.н, доцент кафедри КН _____ О. К. Колесницький

Нормоконтроль, к.т.н., доц. кафедри ПЗ _____ О. М. Рейда

Виконавець, студент групи 1ПІ-18м _____ Є. В. Горовий

Інтерактивна система інкрементного резервування даних

ВИКОНАВ:
ст. гр. 1ПІ-14б Горовий Є.В.

НАУКОВИЙ КЕРІВНИК:
к.т.н. доцент кафедри ПЗ Рейда О.М.

Інтерактивна система інкрементного резервування даних

- **Мета роботи:** підвищення швидкодії і цілісності резервування.
- **Об'єкт дослідження:** процес розробки інтерактивної системи для резервування даних.
- **Предмет дослідження:** методи та засоби розробки інтерактивної системи для резервування даних.

Актуальність обраної теми

- Обмежений обсяг пам’яті для зберігання інформації у мобільних пристроїв.
- Незахищеність пристроїв зберігання інформації від поломок.
- Покращення швидкості, надійності та доступності мережі Інтернет.

Основні задачі дослідження

- Аналіз предметної області.
- Проведення порівняльного аналізу аналогів.
- Аналіз технології розробки системи.
- Розробка методу підвищення швидкодії резервування даних;
- Вибір засобів реалізації програмного продукту.
- Реалізація системи.
- Тестування розробленого програмного продукту.
- Економічне обґрунтування доцільності розробки

Наукова новизна одержаних результатів

Вперше розроблено метод інкрементного резервування даних у хмарному середовищі за таких умов:

- наявність надійного та швидкого інтернет-з'єднання;
- наявність достатнього заряду акумулятора;
- низька завантаженість системи.

Практичне значення одержаних результатів

Розроблено інтерактивну систему інкрементного резервування даних для мобільних пристроїв на платформі Android. Проект призначено для підвищення швидкодії процесу резервування даних мобільних пристроїв з використанням хмарного сховища.

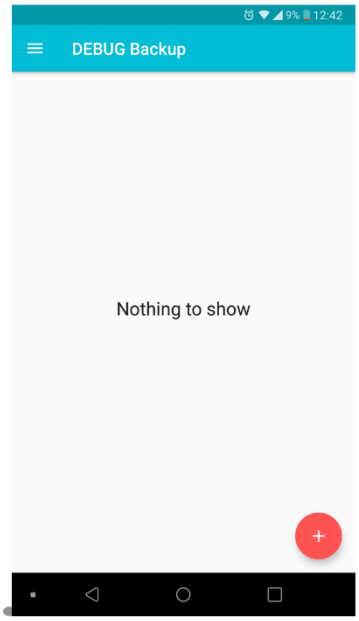
Порівняльний аналіз аналогів

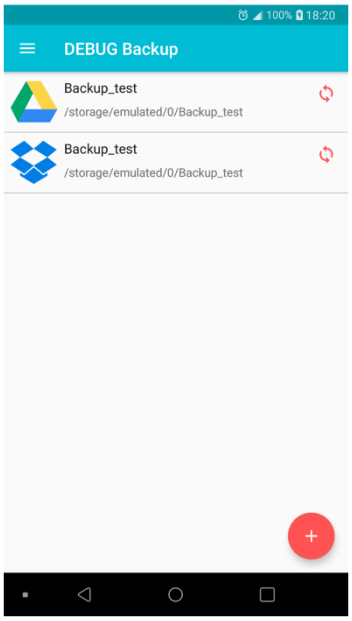
Ознака порівняння	FolderSync	DriveSync	Synchronize Ultimate	Simple Backup
Вибір постійного облікового запису	1	1	0	1
Автоматичне резервування даних	1	0	1	1
Робота у фоновому режимі	0	1	1	1
Підтримка Dropbox	0	0	1	1
Вибір частоти резервування	1	0	0	1
Сумарний коефіцієнт	3	2	3	5

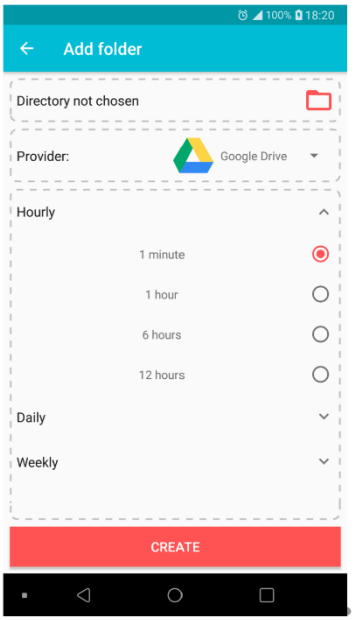
Функціональні відмінності типів резервування

Тип резервування	Данні, що резервуються	Швидкість резервування	Швидкість відновлення	Використання дискового простору
Повне	Всі	Низька	Висока	Високе
Інкрементне	Нові або змінені	Висока	Середня	Низьке
Диференціальне	Нові або змінені з моменту повного резервування	Середня	Висока	Середнє
Синтетичне	Нові або змінені	Висока	Висока	Високе

Графічний інтерфейс







Економічне обґрунтування доцільності розробки

В результаті економічних розрахунків було отримано такі результати:

- абсолютна ефективність вкладених інвестицій дорівнює **412413 грн.**;
- відносна ефективність наукової розробки становить **144%**, що є вищим за мінімальну ставку дисконтування (30%);
- термін окупності вкладених у реалізацію наукового проекту інвестицій складе близько **8 місяців**, що є менше 5-ти років і вказує швидку окупність вкладених інвестицій.

Висновки

В результаті роботи було досягнуто таких цілей:

- Обґрунтовано доцільність розробки;
- Проаналізовано технології розробки системи;
- Розроблено метод підвищення швидкодії резервування даних;
- Розроблено програмний продукт;
- Виконано тестування програмного додатку;
- Обґрунтовано економічну доцільність розробки.

Дякую за увагу!

ДОДАТОК В. Лістинг програми

AppDatabase.java

```
package com.volnoor.backup.core.data;

import android.content.Context;
import android.util.Log;

import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;

import com.volnoor.backup.core.data.item.ItemDao;
import com.volnoor.backup.core.data.item.ItemEntity;

@Database(entities = ItemEntity.class, version = 1)
public abstract class AppDatabase extends RoomDatabase {

    private static final String TAG = "AppDatabase";

    private static final String NAME = "database";

    public static AppDatabase initialize(Context context) {
        Log.d(TAG, "initialize");

        return Room.databaseBuilder(context, AppDatabase.class, NAME)
                .fallbackToDestructiveMigration()
                .build();
    }

    public abstract ItemDao itemDao();
}
```

ItemDao.java

```
package com.volnoor.backup.core.data.item;

import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;

import java.util.List;

import io.reactivex.Completable;
import io.reactivex.Observable;
import io.reactivex.Single;

@Dao
public interface ItemDao {

    @Query("SELECT * FROM item")
    Observable<List<ItemEntity>> getAll();

    @Insert
    Single<Long> addItem(ItemEntity itemEntity);

    @Query("SELECT * FROM item where id = :id")
    Single<ItemEntity> getItem(long id);
```

```
    @Query("DELETE FROM item where id = :id")
    Completable delete(long id);
}
```

ItemEntity.java

```
package com.volnoor.backup.core.data.item;

import androidx.annotation.NonNull;
import androidx.room.ColumnInfo;
import androidx.room.Entity;
import androidx.room.PrimaryKey;

import com.volnoor.backup.core.sync.Providers;

@Entity(tableName = "item")
public class ItemEntity {

    @PrimaryKey(autoGenerate = true)
    private long id;

    @ColumnInfo(name = "name")
    @NonNull
    private String name;

    @ColumnInfo(name = "path")
    @NonNull
    private String path;

    @Providers.Provider
    @ColumnInfo(name = "provider")
    private int provider;

    @ColumnInfo(name = "periodMs")
    private long periodMs;

    public ItemEntity(@NonNull String name, @NonNull String path, int provider,
long periodMs) {
        this.name = name;
        this.path = path;
        this.provider = provider;
        this.periodMs = periodMs;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPath() {
        return path;
```

```
    }

    public void setPath(String path) {
        this.path = path;
    }

    public int getProvider() {
        return provider;
    }

    public void setProvider(int provider) {
        this.provider = provider;
    }

    public long getPeriodMs() {
        return periodMs;
    }

    public void setPeriodMs(long periodMs) {
        this.periodMs = periodMs;
    }
}
```

IProvider.java

```
package com.volnoor.backup.core.sync;

import android.content.Context;
import android.content.Intent;

import androidx.fragment.app.Fragment;

import io.reactivex.Observable;

public interface IProvider {

    boolean isSignedIn();

    void requestSignIn(Fragment fragment, int requestCode);

    Observable<?> handleSignInResult(Intent data);

    Intent getSyncService(Context context, long id);
}
```

GoogleDriveProvider.java

```
package com.volnoor.backup.core.sync.drive;

import android.content.Context;
import android.content.Intent;
import android.util.Log;

import androidx.fragment.app.Fragment;
import androidx.work.Constraints;
import androidx.work.Data;
import androidx.work.NetworkType;
import androidx.work.OneTimeWorkRequest;
import androidx.work.WorkManager;

import com.google.android.gms.auth.api.signin.GoogleSignIn;
```

```
import com.google.android.gms.auth.api.signin.GoogleSignInAccount;
import com.google.android.gms.auth.api.signin.GoogleSignInClient;
import com.google.android.gms.auth.api.signin.GoogleSignInOptions;
import com.google.android.gms.common.api.Scope;
import
com.google.api.client.googleapis.extensions.android.gms.auth.GoogleAccountCredenti
al;
import com.google.api.client.http.javanet.NetHttpTransport;
import com.google.api.client.json.gson.GsonFactory;
import com.google.api.services.drive.Drive;
import com.google.api.services.drive.DriveScopes;
import com.volnoor.backup.BackupApplication;
import com.volnoor.backup.R;
import com.volnoor.backup.core.data.item.ItemEntity;
import com.volnoor.backup.core.sync.IProvider;
import com.volnoor.backup.core.sync.SyncWorker;

import java.io.File;
import java.io.IOException;
import java.util.Collections;
import java.util.concurrent.TimeUnit;

import io.reactivex.Observable;

public class GoogleDriveProvider implements IProvider {

    public static final String TAG = "GoogleDriveProvider";

    private DriveDataSource dataSource;

    public GoogleDriveProvider() {
        Log.d(TAG, "GoogleDriveProvider");

        GoogleSignInAccount account = GoogleSignIn
                .getLastSignedInAccount(BackupApplication.getApplication());

        if (account == null) {
            Log.d(TAG, "GoogleDriveProvider: not signed in");
            silentSignIn();
        } else {
            init(account);
        }
    }

    @Override
    public boolean isSignedIn() {
        Log.d(TAG, "isSignedIn");

        GoogleSignInAccount account = GoogleSignIn
                .getLastSignedInAccount(BackupApplication.getApplication());

        return account != null && !account.isExpired();
    }

    @Override
    public void requestSignIn(Fragment fragment, int requestCode) {
        Log.d(TAG, "requestSignIn");

        GoogleSignInOptions signInOptions =
                new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
                        .requestEmail()
                        .requestScopes(new Scope(DriveScopes.DRIVE_FILE))
```

```
                        .build();
        GoogleSignInClient client = GoogleSignIn.getClient(fragment.getActivity(),
signInOptions);

        fragment.startActivityForResult(client.getSignInIntent(), requestCode);
    }

    @Override
    public Observable<GoogleSignInAccount> handleSignInResult(Intent data) {
        Log.d(TAG, "handleSignInResult");

        return Observable.create(emitter ->
GoogleSignIn.getSignedInAccountFromIntent(data)
                .addOnSuccessListener(googleAccount -> {
                    Log.d(TAG, "handleSignInResult: email " +
googleAccount.getEmail());

                    init(googleAccount);
                    emitter.onNext(googleAccount);
                })
                .addOnFailureListener(emitter::onError));
    }

    private void silentSignIn() {
        Log.d(TAG, "silentSignIn");
        GoogleSignInOptions signInOptions =
                new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
                        .requestEmail()
                        .requestScopes(new Scope(DriveScopes.DRIVE_FILE))
                        .build();
        GoogleSignInClient client =
GoogleSignIn.getClient(BackupApplication.getApplication(),
                signInOptions);
        client.silentSignIn().addOnSuccessListener(account -> {
            Log.d(TAG, "onSuccess: " + account.getEmail());
            init(account);
        }).addOnFailureListener(e -> {
            Log.d(TAG, "onFailure: " + e.getMessage());
            e.printStackTrace();
        });
    }

    private void init(GoogleSignInAccount googleAccount) {
        GoogleAccountCredential credential = GoogleAccountCredential.usingOAuth2(
                BackupApplication.getApplication(),
Collections.singleton(DriveScopes.DRIVE_FILE));

        credential.setSelectedAccount(googleAccount.getAccount());

        Drive googleDriveService = new Drive.Builder(
                new NetHttpTransport(),
                new GsonFactory(),
                credential)

.setApplicationName(BackupApplication.getApplication().getString(R.string.app_name
))
                .build();

        this.dataSource = new V3DriveDataSource(googleDriveService);
    }

    @Override
```

```
    public Intent getSyncService(Context context, long id) {
        return DriveService.getStartIntent(context, id);
    }

    public Observable<String> sync(ItemEntity itemEntity) {
        return this.dataSource.createRootFolder(BackupApplication.getApplication()
                .getString(R.string.app_name))
                .doOnError(throwable -> scheduleSyncAfterError(itemEntity))
                .map(rootId -> {
                    File syncRoot = new File(itemEntity.getPath());
                    GoogleDriveProvider.this.syncFolder(syncRoot, rootId);
                    Log.d(TAG, "sync: finished");
                    scheduleNextSync(itemEntity);
                    return itemEntity.getName();
                });
    }

    private void syncFolder(File directory, String driveParent) throws IOException
{
        Log.d(TAG, "syncFolder: " + directory);

        driveParent = this.dataSource.createOrReturnFolder(directory.getName(),
driveParent);

        File[] files = directory.listFiles();
        if (files != null) {
            for (File file : files) {
                Log.d(TAG, "syncFolder: iterate " + file);

                if (file.isDirectory()) {
                    Log.d(TAG, "syncFolder: directory");
                    syncFolder(file, driveParent);
                } else {
                    Log.d(TAG, "syncFolder: file");
                    this.dataSource.createOrReturnFile(file, driveParent);
                }
            }
        }
    }

    private void scheduleNextSync(ItemEntity itemEntity) {
        Log.d(TAG, "scheduleNextSync: " + itemEntity.getId());

        this.scheduleSync(itemEntity, itemEntity.getPeriodMs());
    }

    private void scheduleSyncAfterError(ItemEntity itemEntity) {
        Log.d(TAG, "scheduleSyncAfterError: " + itemEntity.getId());
        this.scheduleSync(itemEntity, 60_000);
    }

    private void scheduleSync(ItemEntity itemEntity, long delayMs) {
        Log.d(TAG, "scheduleSync: " + delayMs);

        Constraints constraints = new Constraints.Builder()
                .setRequiredNetworkType(NetworkType.CONNECTED)
                .build();

        Data data = SyncWorker.getStartData(itemEntity.getId());

        OneTimeWorkRequest syncWorkRequest = new
OneTimeWorkRequest.Builder(SyncWorker.class)
                .setInitialDelay(delayMs, TimeUnit.MILLISECONDS)
```

```
                .setConstraints(constraints)
                .setInputData(data)
                .build();

        WorkManager.getInstance(BackupApplication.getApplication())
                .enqueue(syncWorkRequest);
    }
}
```

BaseForegroundService.java

```
package com.volnoor.backup.core.sync;

import android.app.Notification;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

import androidx.annotation.DrawableRes;
import androidx.annotation.Nullable;
import androidx.core.app.NotificationCompat;
import androidx.core.app.NotificationManagerCompat;

import com.volnoor.backup.BackupApplication;

import java.util.Random;

public abstract class BaseForegroundService extends Service {

    private static final String TAG = "BaseForegroundService";

    @Override
    public void onCreate() {
        super.onCreate();
        Log.d(TAG, "onCreate");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d(TAG, "onStartCommand");

        startForeground(getId(), getForegroundNotification());

        return START_STICKY;
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy");
    }

    protected void stopForegroundService() {
        Log.d(TAG, "stopForegroundService");

        stopForeground(true);
        stopSelf();
```

```
    }

    protected abstract int getId();

    protected abstract Notification getForegroundNotification();

    protected void showNotification(String title, String text, @DrawableRes int
smallIcon) {
        Notification notification = new NotificationCompat.Builder(this,
                BackupApplication.CHANNEL_NOTIFICATION)
                .setContentTitle(title)
                .setContentText(text)
                .setSmallIcon(smallIcon)
                // TODO .setContentIntent(pendingIntent)
                .build();

        int id = new Random().nextInt(Integer.MAX_VALUE) + 1;
        NotificationManagerCompat.from(this).notify(id, notification);
    }
}
```

DriveService.java

```
package com.volnoor.backup.core.sync.drive;

import android.app.Notification;
import android.content.Context;
import android.content.Intent;
import android.util.Log;

import androidx.core.app.NotificationCompat;

import com.volnoor.backup.BackupApplication;
import com.volnoor.backup.R;
import com.volnoor.backup.core.sync.BaseForegroundService;
import com.volnoor.backup.core.sync.Providers;

import io.reactivex.android.schedulers.AndroidSchedulers;
import io.reactivex.disposables.CompositeDisposable;
import io.reactivex.disposables.Disposable;
import io.reactivex.schedulers.Schedulers;

public class DriveService extends BaseForegroundService {

    private static final String TAG = "DriveService";

    private static final int ID_NOTIFICATION_SYNC = 1;

    private static final String EXTRA_ID = "id";

    private CompositeDisposable disposables = new CompositeDisposable();

    public static Intent getStartIntent(Context context, long id) {
        Log.d(TAG, "getStartIntent: id " + id);

        Intent intent = new Intent(context, DriveService.class);
        intent.putExtra(EXTRA_ID, id);

        return intent;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d(TAG, "onStartCommand");
```

```
        long id = intent.getLongExtra(EXTRA_ID, 0);
        Log.d(TAG, "onStartCommand: " + id);

        startSync(id);
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    protected int getId() {
        return ID_NOTIFICATION_SYNC;
    }

    @Override
    protected Notification getForegroundNotification() {
        return new NotificationCompat.Builder(this,
BackupApplication.CHANNEL_FOREGROUND_SYNC)
                .setContentTitle(getString(R.string.performing_synchronization))
                //.setContentText(text)
                .setProgress(0, 0, true)
                .setSmallIcon(R.drawable.ic_google_drive)
                // TODO .setContentIntent(pendingIntent)
                .build();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        this.disposables.dispose();
    }

    private void startSync(long id) {
        Log.d(TAG, "startSync");

        GoogleDriveProvider provider = (GoogleDriveProvider)
BackupApplication.getApplication() // TODO
                .getProvider(Providers.GOOGLE_DRIVE);

        Disposable disposable =
BackupApplication.getApplication().getDatabase().itemDao().getItem(id).toObservabl
e()
                .flatMap(provider::sync)
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(this::onSyncSuccess, this::onSyncError);

        this.disposables.add(disposable);
    }

    private void onSyncSuccess(String folderName) {
        Log.d(TAG, "onSyncSuccess");

        this.stopForegroundService();
        this.showNotification(getString(R.string.sync_completed), folderName,
R.drawable.ic_google_drive);
    }

    private void onSyncError(Throwable throwable) {
        Log.d(TAG, "onSyncError " + throwable);
        throwable.printStackTrace();

        stopForegroundService();
        showNotification(getString(R.string.error_occurred),
throwable.getMessage(), R.drawable.ic_google_drive);
```

```
    }
}
```

SyncWorker.java

```
package com.volnoor.backup.core.sync;

import android.content.Context;
import android.content.Intent;
import android.os.Build;
import android.util.Log;

import androidx.annotation.NonNull;
import androidx.work.Data;
import androidx.work.RxWorker;
import androidx.work.WorkerParameters;

import com.volnoor.backup.BackupApplication;
import com.volnoor.backup.core.data.item.ItemEntity;

import io.reactivex.Single;
import io.reactivex.disposables.CompositeDisposable;

public class SyncWorker extends RxWorker {

    private static final String TAG = "SyncWorker";

    private static final String EXTRA_ID = "extra_id";

    private CompositeDisposable disposables = new CompositeDisposable();

    public static Data getStartData(long id) {
        return new Data.Builder()
                .putLong(EXTRA_ID, id)
                .build();
    }

    public SyncWorker(@NonNull Context context, @NonNull WorkerParameters
workerParams) {
        super(context, workerParams);
    }

    @NonNull
    @Override
    public Single<Result> createWork() {
        Log.d(TAG, "createWork");
        Data data = getInputData();
        long id = data.getLong(EXTRA_ID, 0);
        Log.d(TAG, "createWork: id " + id);
        return
BackupApplication.getApplication().getDatabase().itemDao().getItem(id)
                .map(this::getSyncIntent)
                .map(intent -> {
                    startForegroundService(intent);
                    return Result.success();
                });
    }

    private Intent getSyncIntent(ItemEntity itemEntity) {
        IProvider provider =
BackupApplication.getApplication().getProvider(itemEntity.getProvider());
        return provider.getSyncService(getApplicationContext(),
itemEntity.getId());
    }
```

```
    private void startForegroundService(Intent intent) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            getApplicationContext().startForegroundService(intent);
        } else {
            getApplicationContext().startService(intent);
        }
    }

    @Override
    public void onStopped() {
        super.onStopped();
        this.disposables.dispose();
    }
}
```

MainActivity.java

```
package com.volnoor.backup.ui.main;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;

import com.volnoor.backup.R;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public void onBackPressed() {
        Fragment fragment =
getSupportFragmentManager().findFragmentById(R.id.fragment_main);

        boolean backPressed = false;
        if (fragment instanceof Navigation) {
            backPressed = ((Navigation) fragment).onBackPressed();
        }

        if (!backPressed) {
            super.onBackPressed();
        }
    }
}
```

MainContract.java

```
package com.volnoor.backup.ui.main;

import com.volnoor.backup.ui.base.BaseView;

public interface MainContract {

    interface View extends BaseView {

        void showListFragment();
    }
```

```
    interface Presenter {

    }
}
```

MainFragment.java

```
package com.volnoor.backup.ui.main;

import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.ActionBar;
import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;

import com.volnoor.backup.R;
import com.volnoor.backup.databinding.FragmentMainBinding;
import com.volnoor.backup.ui.base.BaseFragment;
import com.volnoor.backup.ui.list.ListFragment;

public class MainFragment extends BaseFragment<MainPresenter, FragmentMainBinding>
        implements MainContract.View, Navigation {

    private static final String TAG = "MainFragment";

    private static final String TAG_TOP_FRAGMENT = "top_fragment";

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup
container,
                             @Nullable Bundle savedInstanceState) {
        this.setHasOptionsMenu(true);
        ActionBar actionBar = ((AppCompatActivity)
(getActivity())).getSupportActionBar();
        actionBar.setDisplayShowHomeEnabled(true);
        actionBar.setDisplayHomeAsUpEnabled(true);

        return super.onCreateView(inflater, container, savedInstanceState);
    }

    @Override
    public void onCreateOptionsMenu(@NonNull Menu menu, @NonNull MenuInflater
inflater) {
        ActionBar actionBar = ((AppCompatActivity)
(getActivity())).getSupportActionBar();
        actionBar.setHomeAsUpIndicator(R.drawable.ic_menu);

        super.onCreateOptionsMenu(menu, inflater);
    }

    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item) {
```

```
        switch (item.getItemId()) {
            case android.R.id.home:
                // TODO drawer
                return false;
            default:
                return super.onOptionsItemSelected(item);
        }
    }

    @Override
    protected MainPresenter initializePresenter() {
        return new MainPresenter(this);
    }

    @Override
    protected int getLayoutRes() {
        return R.layout.fragment_main;
    }

    @Override
    public void showListFragment() {
        Log.d(TAG, "showListFragment");
        Fragment fragment = new ListFragment();
        showScreen(fragment);
    }

    @Override
    public void showScreen(Fragment fragment) {
        FragmentManager fragmentManager = getChildFragmentManager();

        Fragment topFragment =
fragmentManager.findFragmentByTag(TAG_TOP_FRAGMENT);
        if (topFragment != null &&
topFragment.getClass().equals(fragment.getClass())) {
            return;
        }

        fragmentManager.popBackStack(null,
FragmentManager.POP_BACK_STACK_INCLUSIVE);
        fragmentManager.beginTransaction()
                .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN)
                .replace(R.id.content, fragment, TAG_TOP_FRAGMENT)
                .commit();
    }

    @Override
    public void addScreen(Fragment fragment) {
        FragmentManager fragmentManager = getChildFragmentManager();
        fragmentManager.beginTransaction()
                .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN)
                .replace(R.id.content, fragment, TAG_TOP_FRAGMENT)
                .addToBackStack(null)
                .commit();
    }

    @Override
    public void setTitle(@NonNull String title) {
        getActivity().setTitle(title);
    }

    @Override
    public boolean onBackPressed() {
        FragmentManager fragmentManager = getChildFragmentManager();
```

```
        if (fragmentManager.getBackStackEntryCount() > 0) {
            fragmentManager.popBackStack();
            return true;
        }
        return false;
    }
}
```

MainPresenger.java

```
package com.volnoor.backup.ui.main;

import com.volnoor.backup.ui.base.BasePresenter;

public class MainPresenter extends BasePresenter<MainContract.View>
        implements MainContract.Presenter {

    public MainPresenter(MainContract.View view) {
        super(view);
    }

    @Override
    protected void onViewCreated() {
        view.showListFragment();
    }
}
```

Nativation.java

```
package com.volnoor.backup.ui.main;

import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;

public interface Navigation {

    void showScreen(@NonNull Fragment fragment);

    void addScreen(@NonNull Fragment fragment);

    void setTitle(@NonNull String title);

    /**
     * @return true if back pressure has been handled
     */
    boolean onBackPressed();
}
```

ListContract.java

```
package com.volnoor.backup.ui.list;

import android.content.Context;
import android.content.Intent;

import androidx.fragment.app.Fragment;

import com.volnoor.backup.ui.base.BaseView;
import com.volnoor.backup.ui.list.model.ItemViewModel;

import java.util.List;

public interface ListContract {
```

```
    interface View extends BaseView {

        Context getContext();

        void showItems(List<ItemViewModel> items);

        ItemViewModel getLastViewModel();

        void showAddItemScreen();

        void showEditItemScreen(ItemViewModel item);

        void showEmptyView();

        void startForegroundService(Intent intent);

        Fragment getFragment();

        int getSignInRequestCode();

        void showError();
    }

    interface Presenter {

        void addButtonClicked();

        void onItemAdded(long id);

        void onItemSyncClicked(ItemViewModel item);

        void onSignInResult(Intent intent);

        void onItemClicked(ItemViewModel item);

        void dispose();
    }
}
```

ListFragment.java

```
package com.volnoor.backup.ui.list;

import android.Manifest;
import android.app.Activity;
import android.content.Intent;
import android.os.Build;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.core.content.ContextCompat;
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.DividerItemDecoration;

import com.volnoor.backup.R;
import com.volnoor.backup.databinding.FragmentListBinding;
import com.volnoor.backup.ui.add.AddItemFragment;
import com.volnoor.backup.ui.base.BaseFragment;
import com.volnoor.backup.ui.edit.EditItemFragment;
```

```
import com.volnoor.backup.ui.list.adapter.ItemAdapter;
import com.volnoor.backup.ui.list.model.ItemViewModel;

import java.util.List;

import pub.devrel.easypermissions.AfterPermissionGranted;
import pub.devrel.easypermissions.EasyPermissions;

public class ListFragment extends BaseFragment<ListPresenter, FragmentListBinding>
        implements ListContract.View {

    public static final String TAG = "ListFragment";

    private static final int RC_ADD_ITEM = 1000;
    private static final int RC_SIGN_IN = 1001;
    private static final int RC_SYNC = 1002;

    private ItemViewModel syncItem; // To hold item (while requesting permissions)

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        this.dataBinding.fabAdd.setOnClickListener(v ->
presenter.addButtonClicked());
    }

    @Override
    protected ListPresenter initializePresenter() {
        return new ListPresenter(this);
    }

    @Override
    protected int getLayoutRes() {
        return R.layout.fragment_list;
    }

    @Nullable
    @Override
    protected String getTitle() {
        return getString(R.string.app_name);
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, @Nullable Intent
data) {
        Log.d(TAG, "onActivityResult: " + requestCode + ", " + resultCode + ", " +
data);
        if (requestCode == RC_ADD_ITEM) {
            if (resultCode == Activity.RESULT_OK) {
                long id = data.getLongExtra(AddItemFragment.EXTRA_ID, 0);
                this.presenter.onItemAdded(id);
            }
        } else if (requestCode == RC_SIGN_IN) {
            if (resultCode == Activity.RESULT_OK) {
                this.presenter.onSignInResult(data);
            }
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
```

```
permissions,
                                           @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);

        EasyPermissions.onRequestPermissionsResult(requestCode, permissions,
grantResults, this);
    }

    @Override
    public void showItems(List<ItemViewModel> items) {
        ItemAdapter adapter = new ItemAdapter(items,
getItemInteractionListener());
        DividerItemDecoration decoration = new
DividerItemDecoration(getActivity(), LinearLayout.VERTICAL);
        decoration.setDrawable(ContextCompat.getDrawable(getContext(),
R.drawable.divider));
        dataBinding.listItems.setAdapter(adapter);
        dataBinding.listItems.addItemDecoration(decoration);
    }

    @Override
    public ItemViewModel getLastViewModel() {
        return this.syncItem;
    }

    @Override
    @AfterPermissionGranted(RC_ADD_ITEM)
    public void showAddItemScreen() {
        if (ensurePermissions(RC_ADD_ITEM)) {
            Fragment fragment = AddItemFragment.newInstance(this, RC_ADD_ITEM);
            this.navigation.addScreen(fragment);
        }
    }

    @Override
    public void showEditItemScreen(ItemViewModel item) {
        Fragment fragment = EditItemFragment.newInstance(item.getId());
        this.navigation.addScreen(fragment);
    }

    @Override
    public void showEmptyView() {
        this.dataBinding.tvEmpty.setVisibility(View.VISIBLE);
    }

    @Override
    public void startForegroundService(Intent intent) {
        Log.d(TAG, "startForegroundService: " + intent);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            getContext().startForegroundService(intent);
        } else {
            getContext().startService(intent);
        }
    }

    @Override
    public Fragment getFragment() {
        return this;
    }

    @Override
    public int getSignInRequestCode() {
```

```
        return RC_SIGN_IN;
    }

    @Override
    public void showError() {
        Toast.makeText(getActivity(), R.string.error_occurred,
Toast.LENGTH_LONG).show();
    }

    private ItemAdapter.ItemInteractionListener getItemInteractionListener() {
        return new ItemAdapter.ItemInteractionListener() {
            @Override
            public void onSyncClicked(ItemViewModel item) {
                Log.d(TAG, "onSyncClicked: " + item.getId());
                ListFragment.this.syncItem = item;
                ListFragment.this.syncItem();
            }

            @Override
            public void onItemClicked(ItemViewModel item) {
                Log.d(TAG, "onItemClicked: " + item.getId());
                ListFragment.this.presenter.onItemClicked(item);
            }
        };
    }

    @AfterPermissionGranted(RC_SYNC)
    private void syncItem() {
        if (this.ensurePermissions(RC_SYNC)) {
            presenter.onItemSyncClicked(this.syncItem);
        }
    }

    private boolean ensurePermissions(int requestCode) {
        String[] permissions = {Manifest.permission.WRITE_EXTERNAL_STORAGE};
        if (EasyPermissions.hasPermissions(getActivity(), permissions)) {
            return true;
        } else {
            EasyPermissions.requestPermissions(this,
getString(R.string.grant_permissions),
                    requestCode, permissions);
        }
        return false;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        this.presenter.dispose();
    }
}
```

ListPresenter.java

```
package com.volnoor.backup.ui.list;

import android.content.Intent;
import android.util.Log;

import com.volnoor.backup.BackupApplication;
import com.volnoor.backup.core.data.item.ItemEntity;
import com.volnoor.backup.core.sync.IProvider;
import com.volnoor.backup.core.sync.Providers;
import com.volnoor.backup.ui.base.BasePresenter;
```

```
import com.volnoor.backup.ui.list.model.ItemViewModel;

import io.reactivex.Observable;
import io.reactivex.android.schedulers.AndroidSchedulers;
import io.reactivex.disposables.CompositeDisposable;
import io.reactivex.disposables.Disposable;
import io.reactivex.functions.Consumer;
import io.reactivex.schedulers.Schedulers;

public class ListPresenter extends BasePresenter<ListContract.View>
        implements ListContract.Presenter {

    public static final String TAG = "ListPresenter";

    private CompositeDisposable disposables = new CompositeDisposable();

    public ListPresenter(ListContract.View view) {
        super(view);
    }

    @Override
    protected void onViewCreated() {
        Disposable disposable =
BackupApplication.getApplication().getDatabase().itemDao().getAll()
                .flatMap(list -> Observable.fromIterable(list)
                        .map(this::convertToViewModel)
                        .toList()
                        .toObservable())
                .observeOn(Schedulers.computation())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(items -> {
                    if (items.isEmpty()) {
                        view.showEmptyView();
                    } else {
                        view.showItems(items);
                    }
                }, throwable -> {
                    throwable.printStackTrace();
                    view.showEmptyView();
                    view.showError();
                });
        this.disposables.add(disposable);
    }

    @Override
    public void addButtonClicked() {
        view.showAddItemScreen();
    }

    @Override
    public void onItemAdded(long id) {
        Log.d(TAG, "onItemAdded: " + id);
        Disposable disposable =
BackupApplication.getApplication().getDatabase().itemDao().getItem(id)
                .map(this::convertToViewModel)
                .subscribeOn(Schedulers.computation())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(itemEntity -> {
                    Log.d(TAG, "accept");
                    startSync(itemEntity);
                }, throwable -> {
                    throwable.printStackTrace();
                    view.showError();
```

```
                });

        this.disposables.add(disposable);
    }

    @Override
    public void onSignInResult(Intent intent) {
        Log.d(TAG, "onSignInResult");
        ItemViewModel item = view.getLastViewModel();

        IProvider provider = BackupApplication.getApplication()
                .getProvider(item.getProvider());

        Disposable disposable = provider.handleSignInResult(intent).subscribe(
                (Consumer<Object>) o -> {
                    Log.d(TAG, "onSignInResult: sign in successful");
                    this.onItemSyncClicked(item);
                },
                throwable -> {
                    Log.d(TAG, "onSignInResult: failed " + throwable);
                    throwable.printStackTrace();
                    view.showError();
                });

        disposables.add(disposable);
    }

    @Override
    public void onItemClicked(ItemViewModel item) {
        this.view.showEditItemScreen(item);
    }

    private ItemViewModel convertToViewModel(ItemEntity itemEntity) {
        return new ItemViewModel(itemEntity.getId(),
                itemEntity.getName(),
                itemEntity.getPath(),
                itemEntity.getProvider(),
                Providers.getProviderData(itemEntity.getProvider()).getIcon());
    }

    @Override
    public void onItemSyncClicked(ItemViewModel item) {
        Log.d(TAG, "onItemSyncClicked: " + item.getId());

        IProvider provider = BackupApplication.getApplication()
                .getProvider(item.getProvider());
        boolean signedIn = provider.isSignedIn();
        Log.d(TAG, "onItemSyncClicked: provider " + provider + ", signed in: " +
signedIn);

        if (signedIn) {
            this.startSync(item);
        } else {
            provider.requestSignIn(view.getFragment(),
view.getSignInRequestCode());
        }
    }

    private void startSync(ItemViewModel item) {
        Log.d(TAG, "startSync");
        IProvider provider =
BackupApplication.getApplication().getProvider(item.getProvider());
        Intent intent = provider.getSyncService(view.getContext(), item.getId());
```

```
            view.startForegroundService(intent);
        }

        @Override
        public void dispose() {
            this.disposables.dispose();
        }
    }
```

ItemViewModel.java

```
package com.volnoor.backup.ui.list.model;

import androidx.annotation.DrawableRes;

import com.volnoor.backup.core.sync.Providers;

public class ItemViewModel {

    private final long id;
    private final String title;
    private final String subtitle;
    @Providers.Provider
    private final int provider;
    @DrawableRes
    private final int icon;

    public ItemViewModel(long id, String title, String subtitle,
@Providers.Provider int provider,
                         @DrawableRes int icon) {
        this.id = id;
        this.title = title;
        this.subtitle = subtitle;
        this.provider = provider;
        this.icon = icon;
    }

    public long getId() {
        return id;
    }

    public String getTitle() {
        return title;
    }

    public String getSubtitle() {
        return subtitle;
    }

    public int getProvider() {
        return provider;
    }

    public int getIcon() {
        return icon;
    }
}
```

ItemAdapter.java

```
package com.volnoor.backup.ui.list.adapter;

import android.view.LayoutInflater;
import android.view.ViewGroup;
```

```
import androidx.annotation.NonNull;
import androidx.databinding.DataBindingUtil;
import androidx.recyclerview.widget.RecyclerView;

import com.volnoor.backup.R;
import com.volnoor.backup.databinding.ItemItemBinding;
import com.volnoor.backup.ui.list.model.ItemViewModel;

import java.util.List;

public class ItemAdapter extends RecyclerView.Adapter<ItemAdapter.ItemViewHolder>
{ // TODO check if adapter can be simplified
    private List<ItemViewModel> items;
    private ItemInteractionListener listener;

    public ItemAdapter(List<ItemViewModel> items, ItemInteractionListener
listener) {
        this.items = items;
        this.listener = listener;
    }

    @NonNull
    @Override
    public ItemViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
        ItemItemBinding binding =
DataBindingUtil.inflate(LayoutInflater.from(parent.getContext()),
                R.layout.item_item, parent, false);
        return new ItemViewHolder(binding);
    }

    @Override
    public void onBindViewHolder(@NonNull ItemViewHolder holder, int position) {
        ItemViewModel item = this.items.get(position);
        holder.binding.setItem(item);
        holder.binding.setListener(this.listener);
    }

    @Override
    public int getItemCount() {
        return items.size();
    }

    class ItemViewHolder extends RecyclerView.ViewHolder {
        private ItemItemBinding binding;

        public ItemViewHolder(@NonNull ItemItemBinding binding) {
            super(binding.getRoot());
            this.binding = binding;
        }
    }

    public interface ItemInteractionListener {

        void onSyncClicked(ItemViewModel item);

        void onItemClicked(ItemViewModel item);
    }
}
```