Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Мобільний застосунок на платформі Android для автоматизації бізнес- процесів торгівельного підприємства

Назва теми

Рівень вищої освіти Перший(бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.190133.19.12.ПЗ

Виконав студент IV курсу, група ІПЗ-19-1 _____ Ю. О Леус
Підпис Ініціали, прізвище

Керівник канд. техн. наук, доцент _____ І. В. Гурман
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ Ю. В. Форкун
Науковий ступінь, звання Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

6 06 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
02 01 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Леусу Юрію Олеговичу
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Мобільний застосунок на платформі Android для автоматизації бізнес-процесів торгівельного підприємства

Керівник кваліфікаційної роботи Іван Васильович Гурман, канд. пед. наук, доцен
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2023 р. № 5
2. Строк подання студентом роботи на кафедру 01.06.2023 р.
3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі. Проектування програмного забезпечення. Програмно реалізація.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди)
Графічна частина (3 діаграми формату А3)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Форкун Юрій Вікторович, доцент	05.06.23	05.06.23
Антиплагіат	Гурман Іван Васильович, доцент	02.06.2023	02.06.23

7. Дата видачі завдання « 02 » січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Збір матеріалу за темою кваліфікаційної роботи (КвР); дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2023	
2 Проектування програмного забезпечення	01.02 – 28.02.2023	
3 Програмна реалізація з використанням відповідних засобів розробки	01.03 – 10.04.2023	
4 Тестування програмного забезпечення	11.04 – 30.04.2023	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2023	
8 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2023	
9 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2023	

Студент _____ Ю. О. Леус
Підпис Ініціали, прізвище

Керівник роботи _____ І. В. Гурман
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Мобільний застосунок на платформі Android для автоматизації бізнес-процесів торговельного підприємства».

Автор роботи: Леус Юрій Олегович.

Керівник роботи: Гурман Іван Васильович.

Пояснювальна записка: 69с., 5 рис., 1 табл., 5 дод., 41 джерел.

Графічна частина: 12 презентаційних слайдів.

ПІДТВЕРДЖЕННЯ ЗАМОВЛЕНЬ, МОБІЛЬНИЙ ЗАСТОСУНОК, СТВОРЕННЯ ТРАНСПОРТНИХ НАКЛАДНИХ, НОВА ПОШТА.

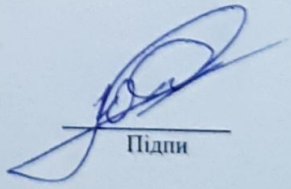
Метою кваліфікаційної роботи є розробка мобільного застосунку на платформі Android, для автоматизації бізнес-процесів торгівельних підприємств. Основною метою програми буде автоматизація підтвердження замовлень і створення транспортних накладних. Мета полягає в тому, щоб забезпечити зручне практичне рішення, яке підвищує ефективність, знижує витрати та покращує задоволеність клієнтів.

У кваліфікаційній роботі проведено аналіз предметної області та її інформаційного забезпечення, визначені вимоги до мобільного застосунку, розроблена загальна архітектура застосунку, спроектована структура бази даних та структура застосунку.

Для розробки застосунку використано мову програмування Kotlin та Java, бази даних SQLite.

У результаті проектування створено мобільний застосунок для автоматизації бізнес-процесів торгових підприємств.

30.05.2023
Дата


Підпи

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.190133.19.12.ПЗ	Пояснювальна записка	69		
2	А4		Завдання на	1		
			кваліфікаційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	12		
5	А3	КвРІПЗ.190133.19.12.Е8	Діаграма деком-	1		
			позиції наших			
6	А3	КвРІПЗ.190133.19.12.Е8	ER – діаграма бази даних	1		
7	А3	КвРІПЗ.190133.19.12.Е8	ER – діаграма класів	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.190133.19.12.ВД		
Розроб.		Леус Ю. О.		04.06	Мобільний застосунок для автоматизації бізнес-процесів торгівельного підприємства Відомість документів	Літ.	Арк. 1 / Акрушів 1
Перевір.		Гурмна І. В.		1.06			
Н. Контр.		Форкун Ю. В.		05.06		ХНУ, ІПЗ-19-1	
Затверд.		Бедратюк Л. П.		6.06			

ЗМІСТ

ВСТУП

У сучасному бізнес-середовищі, яке швидко розвивається, організаціям необхідно впроваджувати інноваційні технології, щоб залишатися конкурентоспроможними та оптимізувати свою діяльність. Мобільні програми стали важливим інструментом для бізнесу, надаючи їм можливість оптимізувати процеси та підвищити ефективність.

Тема кваліфікаційної роботи є вкрай актуальною та необхідною в сучасному бізнес-ландшафті. Тому що існують проблеми, з якими стикаються торговельні компанії в ефективному та результативному управлінні своїми операціями. Ручне підтвердження замовлень і створення транспортних накладних може зайняти багато часу та бути схильним до помилок, що призведе до затримки поставок і незадоволених клієнтів. Автоматизація цих процесів не тільки підвищить ефективність і точність, але й забезпечить своєчасну доставку товарів і послуг клієнтам. Це призведе до більшої задоволеності та утримання клієнтів.

Виходячи з викладеної актуальності, метою кваліфікаційної роботи є розробка мобільного застосунку на платформі Android, для автоматизації бізнес-процесів торгівельних підприємств. Основною метою програми буде автоматизація підтвердження замовлень і створення транспортних накладних. Мета полягає в тому, щоб забезпечити зручне практичне рішення, яке підвищує ефективність, знижує витрати та покращує задоволеність клієнтів.

Завдання, які необхідно вирішити для досягнення мети:

– Провести ретельний аналіз бізнес-процесів торговельних підприємств, щоб виявити конкретні больові точки та вузькі місця, які необхідно усунути за допомогою автоматизації. Це завдання передбачає розуміння поточних процесів, виявлення областей неефективності та помилок, а також визначення конкретних вимог до мобільного застосунку.

– Розробити архітектуру мобільного застосунку, яка відповідає визначеним вимогам і використовує можливості платформи Android. Це завдання передбачає розробку загальної архітектури системи, включаючи необхідні апаратні та

програмні компоненти, а також вибір відповідних засобів розробки та інфраструктури.

– Розробити зручний та інтуїтивно зрозумілий інтерфейс для програми, який забезпечує легку навігацію та ефективне використання. Це завдання передбачає розробку макета інтерфейсу, створення потоків користувачів і взаємодії, а також включення елементів дизайну, які покращують зручність використання.

– Впровадити функціональність програми для автоматизації підтвердження замовлень і створення транспортних накладних. Це завдання передбачає написання коду для інтеграції необхідної функціональності в застосунок, включаючи API та інші серверні служби.

– Тестування програми, щоб переконатися в її точності, надійності та продуктивності, і внесення необхідних покращень.

Виконавши ці завдання, буде досягнута мета розробки мобільного застосунку для автоматизації бізнес-процесів підприємств торгівлі.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Тематична галузь, для якої планується розробка мобільного застосунку – автоматизація бізнес-процесів підприємств торгівлі. Торговельні підприємства – це підприємства, які займаються купівлею та продажем товарів або послуг з метою отримання прибутку. Бізнес-процеси торговельних підприємств включають управління запасами, продажі та маркетинг, обробку замовлень і логістику.

Для аналізу предметної області необхідно розглянути структурно-функціональні особливості торгових підприємств. Підприємства торгівлі мають складну структуру, що включає кілька відділів, включаючи збут, маркетинг, закупівлі, логістику та фінанси. Бізнес-процеси, задіяні в кожному відділі, взаємопов’язані, і будь-яка неефективність одного відділу може вплинути на загальну ефективність підприємства.

З точки зору функціональності, торгові підприємства значною мірою покладаються на інформаційні технології для ефективного управління своїми операціями. Використання програмного забезпечення для управління запасами, обробки замовлень і логістики широко поширене в галузі. Однак багато підприємств досі покладаються на ручні процеси для певних завдань, таких як документообіг і введення даних, що може зайняти багато часу та бути схильним до помилок.

Ця модель IDEF0 представляє основний бізнес-процес торгової компанії, який включає кілька підпроцесів. Процес починається з клієнта, який ініціює процес продажу шляхом розміщення замовлення. Процес продажу включає такі дії, як управління клієнтами, вибір продукту та ціноутворення. Після обробки замовлення він переходить до підпроцесу обробки та підтвердження замовлення, який включає такі дії, як підтвердження замовлення та підготовка відправлення. Процес транспортування та доставки включає такі дії, як планування та

відстеження транспортування. Нарешті, процес виставлення рахунків-фактур і платежів включає такі дії, як створення рахунків-фактур, обробка платежів і звірка рахунків.

Розглянемо один з бізнес-процесів будь якої торгової компанії за допомогою моделі IDEF0. Модель зображена на рисунку 1.1.

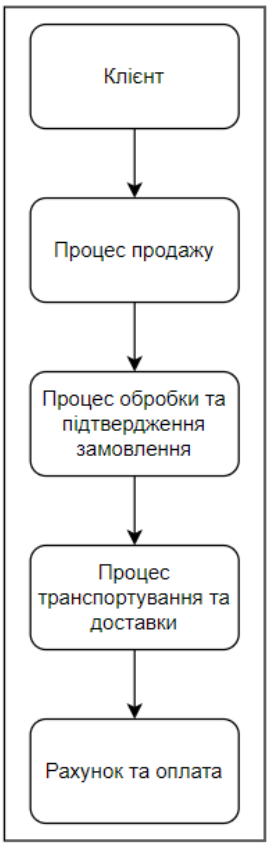


Рисунок 1.1 – Бізнес процес торгової компанії

Мобільний застосунок, який буде розроблений для автоматизації бізнес-процесів торговельних підприємств, в першу чергу буде зосереджений на підпроцесі обробки та підтвердження замовлень, а також на процесі транспортування та доставки, автоматизуючи ці процеси та підвищуючи їх ефективність.

Щоб розробити успішну програмну для торгової компанії, важливо розуміти інформаційні потреби кінцевих користувачів. Кінцевими користувачами програмного забезпечення зазвичай є працівники торгової компанії, які

відповідають за керування та обробку замовлень, створення транспортних накладних та інші пов’язані завдання.

Процес перетворення вхідної інформації у вихідну є критично важливим аспектом при визначені кінцевого користувача. У контексті торгової компанії цей процес включав би введення різних вхідних даних, пов’язаних із замовленнями клієнтів, рівнями запасів, графіками доставки та платіжною інформацією, а також їх обробку таким чином, щоб отримати бажаний результат.

Загалом процес може включати такі кроки:

– Збір даних. Першим кроком у процесі буде збір даних із різних джерел, включаючи замовлення клієнтів, бази даних інвентаризації та системи виставлення рахунків.

– Обробка даних: після того, як дані будуть зібрані, їх потрібно буде обробити, щоб отримати бажаний результат. Це може включати виконання обчислень, сортування та фільтрування даних і створення звітів.

– Генерація вихідних даних: останнім кроком у процесі буде створення бажаних вихідних даних на основі оброблених даних. Це може включати створення транспортних рахунків, оновлення рівня запасів і створення звітів для керівництва.

Програмне забезпечення повинно мати можливість обробляти великі обсяги даних, швидко й точно обробляти їх, а також представляти результати в зрозумілій і зручній для користувача формі.

Загалом, кінцевим користувачам програмного забезпечення потрібен доступ до низки інформації, зокрема:

– Деталі замовлення: це включатиме таку інформацію, як ім’я клієнта, кількість замовлення, деталі продукту та дата доставки.

– Інформація про доставку: це включатиме такі деталі, як адреса доставки, дата доставки та статус доставки.

– Інформація про клієнта: це включатиме такі деталі, як контактна інформація клієнта, історія замовлень і будь-які спеціальні інструкції чи запити.

– З точки зору впровадження інформаційних технологій, автоматизації виробничих процесів, процесів обробки та передачі інформації торговельні підприємства стикаються з рядом проблем і невирішених питань. До них належать:

– Неефективні ручні процеси: багато торговельних підприємств все ще покладаються на ручні процеси для таких завдань, як оформлення документів і введення даних, що може зайняти багато часу та бути схильним до помилок.

– Відсутність видимості в режимі реального часу: торговельні підприємства стикаються з проблемами отримання видимості своїх операцій у режимі реального часу. Це може призвести до затримок у прийнятті рішень і призвести до неефективності загального процесу.

– Обмежена мобільність: багато торговельних підприємств все ще покладаються на настільні програмні застосунки, які обмежують мобільність і ускладнюють роботу співробітників у дорозі.

– Проблеми інтеграції: торговельні підприємства часто використовують кілька програмних застосунків для різних завдань, що призводить до проблем інтеграції та неузгодженості даних.

Розробка мобільного застосунку для автоматизації бізнес-процесів підприємств торгівлі може допомогти вирішити ці проблеми та невирішені питання. Програма може підвищити ефективність, усуваючи неефективні ручні процеси, забезпечуючи видимість операцій у режимі реального часу, покращуючи безпеку даних, забезпечуючи мобільність та полегшуючи інтеграцію між програмними застосунками.

Основне завдання мобільного застосунку – автоматизувати бізнес-процеси підтвердження замовлень та формування транспортних накладних. Це критично важливі процеси в діяльності торгівельного підприємства, оскільки вони включають доставку товарів клієнтам і управління пов’язаною документообіом.

Розробка мобільного застосунку для автоматизації бізнес-процесів торговельних підприємств необхідна для підвищення ефективності, зменшення помилок та оптимізації діяльності. Застосунок дозволить користувачам підтверджувати замовлення та створювати транспортні рахунки на ходу, усуваючи

потребу в ручній роботі з документами та оптимізуючи процес логістики. Це в кінцевому підсумку призведе до швидшого виконання замовлень, підвищення рівня задоволеності клієнтів і підвищення прибутковості підприємства.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області.

Щоб розробити застосунок, який буде підтверджувати замовлення та створювати транспортні накладні через API компанії перевезень, необхідно оцінити існуюче програмне та технічне забезпечення, яке може забезпечити таку функціональність.

Одним з можливих варіантів є використання готових рішень для автоматизації логістики та перевезень, таких як TMS (Transportation Management System - Система управління транспортом) або WMS (Warehouse Management System - Система управління складом). На сьогоднішній день існує декілька популярних сервісів цього типу:

– SAP Extended Warehouse Management (EWM) - це програмне забезпечення для управління складом, яке дозволяє керувати всіма операціями на складі, включаючи отримання замовлень, збір та пакування товарів, відвантаження та перевезення. SAP EWM також інтегрується з різними транспортними компаніями, включаючи "Нову Пошту".

– Oracle Transportation Management - це програмне забезпечення для керування логістикою, яке дозволяє планувати та виконувати операції з доставки товарів, включаючи підтвердження замовлень та створення транспортних накладних через API компаній перевезень.

– Manhattan Associates - це комплексне рішення для керування логістикою та управління складом, яке має інтеграцію з різними транспортними компаніями, включаючи "Нову Пошту". Manhattan Associates дозволяє автоматизувати всі етапи логістичного процесу, починаючи з приймання замовлень і закінчуючи доставкою товарів.

– Існують також інші готові рішення, такі як Magaya, ShipStation, 3PL Central та інші, які також дозволяють підключатися до API перевізних компаній.

Розглянемо деякі компанії більш детально, щоб краще зрозуміти їх переваги та недоліки та зробити відповідні висновки.

SAP Extended Warehouse Management (EWM) - це програмне забезпечення для управління складом, яке дозволяє планувати, контролювати та виконувати операції зберігання та розподілу товарів в складських приміщеннях. Основні функції EWM включають приймання та відвантаження товарів, розміщення та збір товарів, підтримку пакувальних процесів, інвентаризацію складу, кросдокінг та перерозподіл товарів, а також управління транспортними засобами для доставки товарів.

Основна перевага EWM полягає у можливості інтеграції з іншими програмними продуктами SAP, що дозволяє створювати повністю інтегровану систему управління складом, зведення складських запасів до мінімуму та підвищення ефективності операцій. Щодо підтвердження та створення транспортних накладних через API компаній перевезень, це залежить від функціональних можливостей конкретної API компанії. Однак, зазвичай EWM надає можливості для створення документів доставки, включаючи транспортні накладні, які можуть бути створені автоматично на основі даних зі складських операцій.

Деякі з недоліків SAP Extended Warehouse Management включають наступне:

– Висока складність встановлення та налаштування: це може вимагати значних зусиль для успішного встановлення та налаштування системи. Навчання персоналу також може зайняти багато часу та коштів.

– Висока вартість: встановлення та розгортання SAP EWM можуть бути дорогими, особливо для менших підприємств. Також варто враховувати витрати на підтримку та оновлення системи.

– Складна інтеграція з іншими системами: SAP EWM може мати складну інтеграцію з іншими системами, що може призвести до проблем зі збільшенням часу на розробку та налагодження інтеграційних модулів.

– Великий обсяг даних: SAP EWM може генерувати великий обсяг даних, що може створювати проблеми з обробкою та збереженням даних.

– Відсутність гнучкості: SAP EWM може мати обмежену гнучкість, що може ускладнити внесення змін в систему, коли це потрібно.

– Система може бути надто складною для менших складів: SAP EWM може бути надто складною для менших складів, де не всі його функції є необхідними, і може бути дорогим для підприємств з обмеженими бюджетами.

– Вимоги до обладнання та програмного забезпечення: SAP EWM може вимагати певного обладнання та програмного забезпечення для ефективної роботи, що може бути дорогим для підприємств з обмеженими бюджетами.

Побачити інтерфейс SAP Extended Warehouse Management можна на зображені - Рисунок 1.2

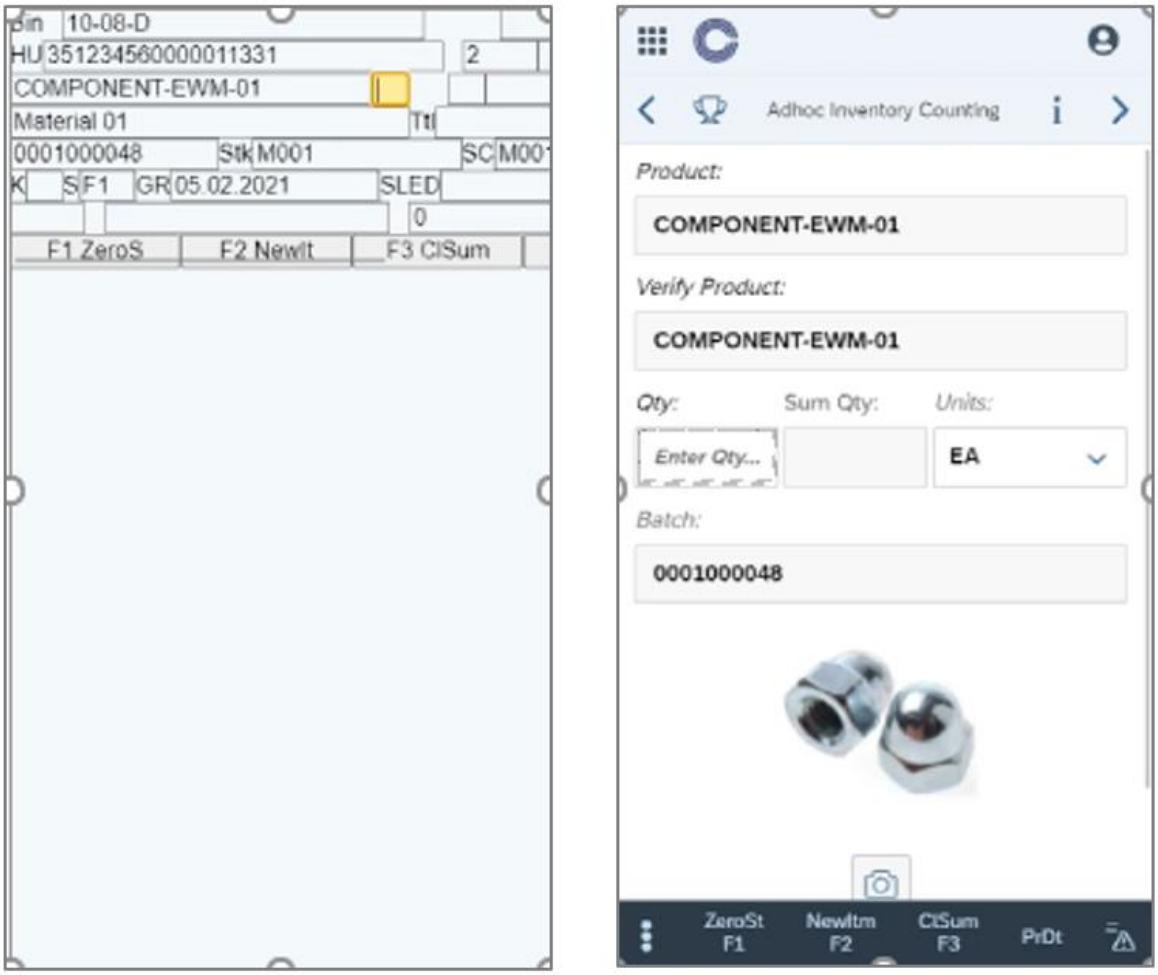


Рисунок 1.2 – Інтерфейс додатку «Extended Warehouse Management»

Далі ми розглянемо наступний застосунок - Manhattan Associates і їхній продукт Manhattan SCALE, програмне забезпечення для управління складом.

Побачити інтерфейс Manhattan SCALE можна на зображені -Рисунок 1.3.

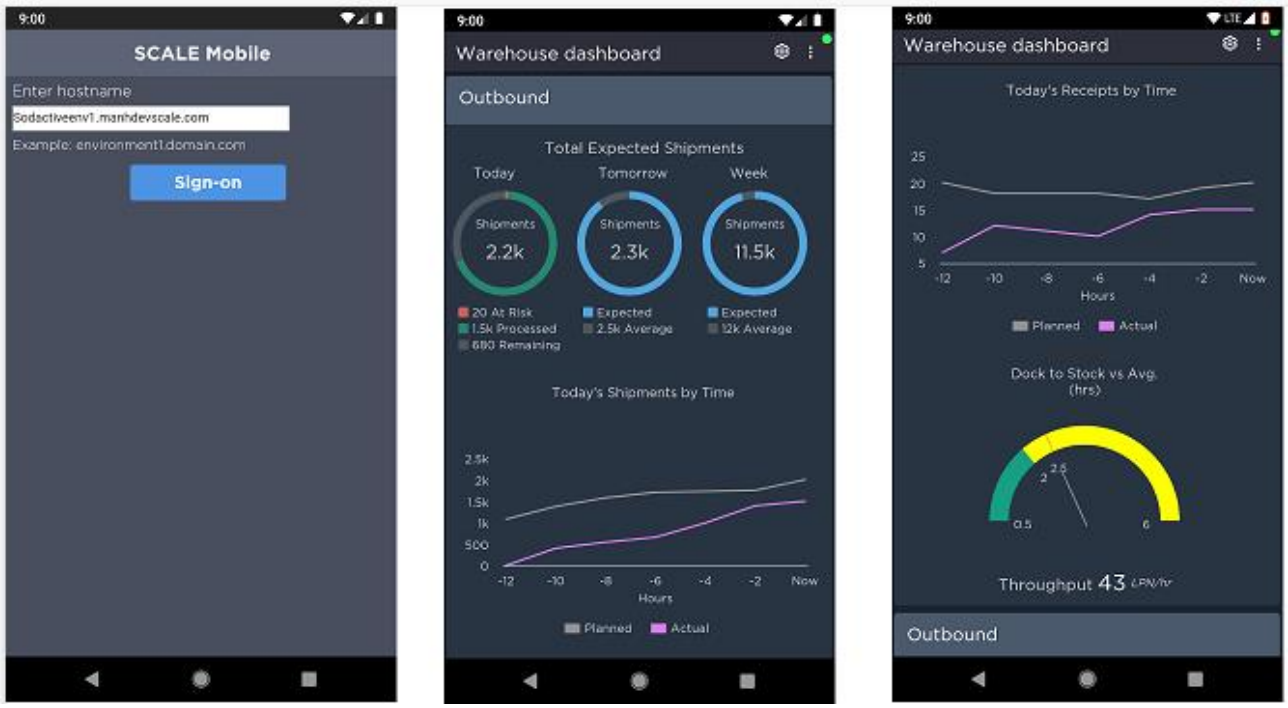


Рисунок 1.3 – Інтерфейс додатку «Manhattan SCALE»

Manhattan SCALE має кілька переваг:

– Масштабованість: Воно працює з різними типами складів та міркувань, включаючи невеликі і великі склади, а також різні ринки та географічні регіони.

– Функціональність: Manhattan SCALE надає широкий спектр функціональних можливостей для управління складом, включаючи вантажні термінали, управління запасами, комплектацію замовлень, оптимізацію маршрутів та багато іншого.

– Налагодження та підтримка: Компанія Manhattan Associates надає якісне налагодження та підтримку своїх продуктів, включаючи швидке вирішення проблем та підтримку 24/7.

– Однак, є кілька недоліків у використанні Manhattan SCALE:

– Вартість: Цей продукт є висококласним і може бути дорогим для багатьох компаній, особливо для менших підприємств з обмеженими бюджетами.

– Складність: Manhattan SCALE має велику кількість функціональностей, що може зробити його вивчення та впровадження складними завданнями.

– Залежність від постачальника: Компанії, які використовують Manhattan SCALE, стають залежними від компанії Manhattan Associates для налагодження та підтримки продукту.

1.3 Аналіз вимог до програмного забезпечення.

Основні вимоги до Android-застосунку для підтвердження замовлень та формування транспортних накладних через API перевізної компанії "Нова Пошта" включають наступне:

– Авторизація та аутентифікація користувачів: Застосунок повинен забезпечувати безпечну авторизацію та аутентифікацію користувачів, використовуючи логін та пароль.

– Інтеграція з API Нової Пошти: Застосунок повинен мати можливість взаємодіяти з API перевізної компанії, зокрема, для підтвердження замовлень та створення транспортних накладних.

– Перегляд статусу підтвердження: Застосунок повинен надавати можливість відображати статус підтвердження замовлення в конкретний момент часу.

– Створення транспортних накладних: Застосунок повинен мати можливість створювати транспортні накладні на основі даних про замовлення.

– Система повідомлень: Застосунок повинен забезпечувати надсилання повідомлень користувачам щодо стану замовлення, доставки та інших подій.

– Зберігання даних: Застосунок повинен мати можливість зберігати та оновлювати дані про замовлення, доставку та інші події.

– Підтримка різних мов: Застосунок повинен підтримувати різні мови для зручності користувачів.

UML-діаграма варіантів використання (Use Case Diagram) використовується для моделювання функціональності системи з точки зору користувача. У даному випадку, UML-діаграма варіантів використання була створена на основі цих вимог для Android-застосунку "Нова Пошта" для підтвердження замовлень та формування транспортних накладних через їх API.

Ця діаграма відображає основні варіанти використання застосунку, які необхідні для його коректної роботи. Вона допомагає уточнити функціональні вимоги до застосунку, описати потреби користувачів, розробити архітектуру застосунку та визначити його ключові компоненти.

За допомогою цієї діаграми можна краще зрозуміти, які можливості повинен мати застосунок для задоволення потреб користувачів, а також які можливі функціональні вимоги можуть виникнути у майбутньому.

Побачити UML діаграму варіантів використання можна на зображені - Рисунку 1.4.

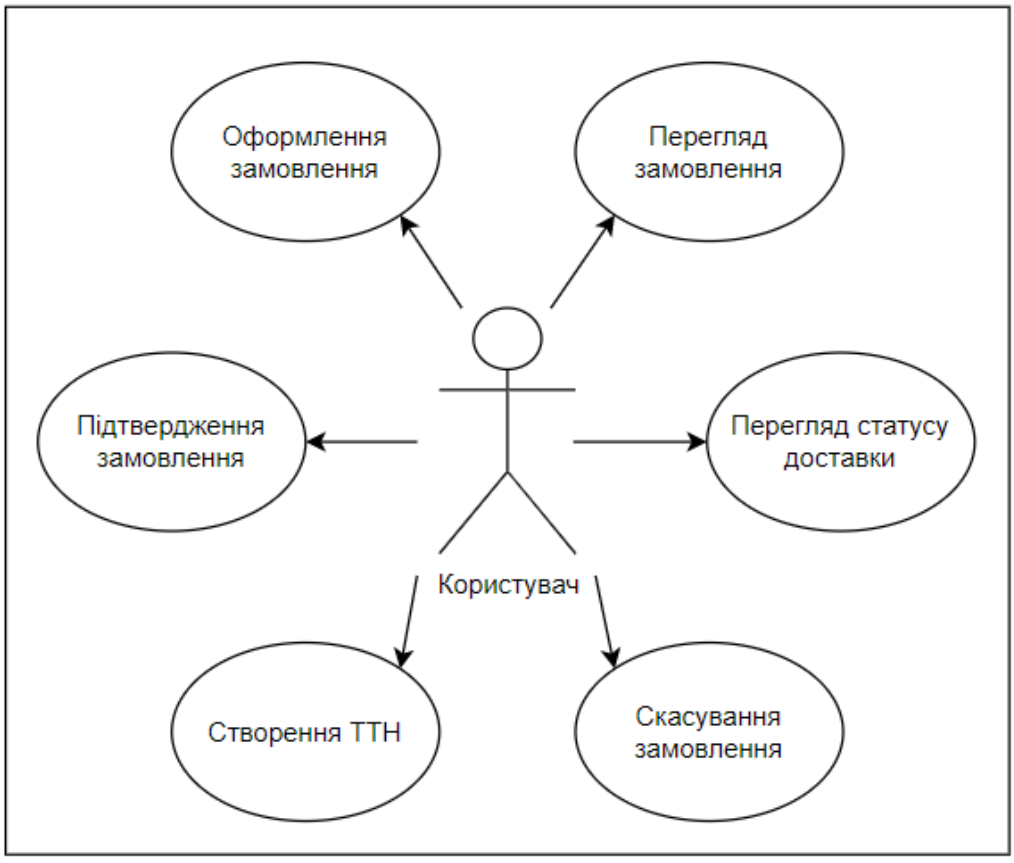


Рисунок 1.4 – Діаграма варіантів використання

Технічне завдання на розробку програмного забезпечення для Android-застосунку, яке забезпечує підтвердження замовлень та формування транспортних накладних через API перевізної компанії "Нова Пошта", містить наступні вимоги:

У цьому технічному завданні описуються вимоги до розробки Android-застосунку для підтвердження замовлень та формування транспортних накладних через API компанії "Нова Пошта".

Технічне завдання:

а) загальні вимоги:

1) застосунок має бути розроблений для платформи Android версії 5.0 і вище;

2) застосунок має підтримувати роботу в офлайн-режимі;

3) застосунок має мати зручний та простий інтерфейс користувача;

4) застосунок має бути сумісним з API компанії "Нова Пошта";

5) застосунок має зберігати історію замовлень та транспортних накладних;

б) функціональні вимоги:

1) авторизація користувача через API компанії "Нова Пошта";

6) перегляд та підтвердження замовлень з використанням API компанії "Нова Пошта";

7) створення транспортних накладних з використанням API компанії «Нова Пошта»;

8) перегляд історії замовлень та транспортних накладних;

9) відстеження статусу відправлень з використанням API компанії "Нова Пошта»;

10) збереження локальної копії інформації про замовлення та транспортні накладні для доступу в офлайн-режимі;

11) введення контактної інформації для кожного замовлення та транспортної накладної;

в) технічні вимоги:

12) розробка застосунку на мові програмування Java або Kotlin;

13) використання стандартних бібліотек Android SDK для реалізації функцій;

14) використання Retrofit для взаємодії з API компанії "Нова Пошта";

15) використання Room для збереження інформації про замовлення та транспортні накладні;

16) адаптивний дизайн для різних розмірів екранів;

г) тестування:

17) функціональне тестування, включаючи тести авторизації, перегляду замовлень, створення транспортних накладних, перегляду історії замовлень та транспортних накладних, а також відстежування статусу відправлень;

18) тестування на різних версіях ОС Android та на різних пристроях;

д) додаткові вимоги:

19) можливість оплати замовлень через інтеграцію з платіжним сервісом;

20) можливість додавання нових адрес доставки.

1.4 Висновки. Постановка задачі.

Зважаючи на вказані вимоги та технічне завдання потрібно створити Андроїд застосунок для підтвердження замовлень та формування транспортних накладних через API перевізної компанії "Нова Пошта". Створити зручний та швидкий інтерфейс для підтвердження замовлень та створення транспортних накладних без необхідності використання веб-сайту компанії.

Потрібно реалізувати підтримку Андроїд версії 5.0 і вище, а також реалізувати, можливість роботи в офлайн-режимі.

Потрібно реалізувати можливості застосунку, такі як авторизація користувача", перегляд та підтвердження замовлень, створення транспортних накладних, перегляд історії замовлень та транспортних накладних, відстеження

статусу відправлень та можливість збереження локальної копії інформації про замовлення та транспортні накладні для доступу в офлайн-режимі.

Під час розробки застосунку також потрібно врахувати вимоги до безпеки даних та користувацької інформації, зокрема захист персональних даних користувачів, шифрування взаємодії з API компанії "Нова Пошта" та забезпечення безпеки даних в офлайн-режимі.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір типу архітектури та шаблонів проектування

Архітектура програми для Android відіграє вирішальну роль у визначенні її якості та успіху. Він забезпечує структурований підхід до проектування та розробки програмного забезпечення, що допомагає гарантувати, що програма є надійною, масштабованою та зручною для обслуговування.

Добре розроблена архітектура сприяє багаторазовому використанню коду, що скорочує час і вартість розробки. Це також робить код більш модульним і легшим для тестування, що призводить до меншої кількості помилок і кращої загальної якості. Крім того, хороша архітектура полегшує додавання нових функцій або зміну існуючих без небажаних побічних ефектів або поломки програми.

Крім того, наявність чіткої та узгодженої архітектури полегшує роботу різних членів команди над проектом і гарантує, що всі працюють на одній сторінці. Це особливо важливо для великих груп розробників, де декілька розробників можуть одночасно працювати над різними частинами програми.

Таким чином, вибір правильної архітектури для програми Android є важливим для забезпечення її успіху та довговічності. Це може заощадити час і гроші в процесі розробки, покращити якість коду та зручність обслуговування, а також спростити спільну роботу в команді.

Тому буде доречним розглянути типи архітектур для Android застосунків:

– Model-View-Controller (MVC): це класичний шаблон архітектури, який розділяє програму на три компоненти: модель, представлення та контролер. Модель представляє дані та бізнес-логіку, представлення відповідає за відображення даних користувачеві, а контролер обробляє введені користувачем дані та спілкується з моделлю та представленням.

– Model-View-Presenter (MVP): цей шаблон архітектури схожий на MVC, але з акцентом на відокремленні логіки презентації від бізнес-логіки. Модель представляє дані, представлення відображає дані для користувача, а презентатор

діє як посередник між ними, обробляючи введені користувачем дані та оновлюючи представлення та модель за потреби.

– Model-View-ViewModel (MVVM): цей шаблон архітектури розроблено для спрощення поділу проблем між інтерфейсом користувача та бізнес-логікою. Модель представляє дані, представлення відображає дані користувачеві, а модель представлення діє як посередник між ними, забезпечуючи зв’язування даних і обробку введених користувачем даних.

– Clean Architecture: цей шаблон архітектури наголошує на розділенні проблем і можливості тестування шляхом поділу програми на рівні на основі рівня абстракції. Рівні включають рівень домену (бізнес-логіка), рівень даних (постійність і мережа) і рівень презентації (UI).

– Flux: цей шаблон архітектури призначений для створення масштабованих і підтримуваних програм. Він розділяє програму на чотири компоненти: дії, магазини, перегляди та диспетчер. Дії – це події, ініційовані користувачем, сховища містять стан програми та бізнес-логіку, представлення відображають дані для користувача, а диспетчер обробляє зв’язок між компонентами.

Виходячи з вимог і проблем, описаних раніше, найуспішнішою архітектурою для Android програми по автоматизації бізнес-споцесів буде архітектура Model-View-ViewModel (MVVM).

Однією з головних переваг архітектури MVVM є її чіткий розподіл проблем. Компонент Model представляє дані та бізнес-логіку програми, компонент View представляє інтерфейс користувача, а компонент ViewModel діє як посередник між компонентами Model і View. Такий розподіл завдань полегшує обслуговування, тестування та модифікацію програми в майбутньому.

Крім того, архітектура MVVM полегшує зв’язування даних, що є потужною функцією розробки Android. Зв’язування даних дозволяє програмі автоматично оновлювати інтерфейс користувача на основі змін даних, не вимагаючи ручного втручання розробника. Це спрощує процес розробки та знижує ризик помилок.

Крім того, архітектура MVVM сприяє багаторазовому використанню, оскільки кожен компонент можна розробляти та тестувати незалежно та

потенційно повторно використовувати в інших проектах. Це може заощадити час і ресурси в довгостроковій перспективі.

Загалом архітектура MVVM добре відповідає вимогам і проблемам, описаним раніше, оскільки вона сприяє поділу проблем, полегшує зв’язування даних і сприяє багаторазовому використанню, що робить її ідеальним вибором для програми для автоматизації бізнес-процесів.

2.2 Опис декомпозиції

Декомпозиція потрібна при розробці програмного забезпечення, зокрема у формі загальної декомпозиції проекту, з кількох причин:

– Розуміння системи: забезпечує чіткий огляд структури та компонентів проекту, допомагаючи краще розуміти систему.

– Управління складністю: декомпозиція розбиває проект на керовані частини, що полегшує роботу зі складністю програмної системи.

– Співпраця та командна робота: забезпечує паралельну розробку та ефективну співпрацю між членами команди, покращуючи продуктивність і координацію.

– Повторне використання та ремонтопридатність: декомпозиція полегшує повторне використання компонентів і спрощує технічне обслуговування.

– Масштабованість і гнучкість: забезпечує легку масштабованість і адаптацію до мінливих вимог.

– Зменшення ризиків: допомагає виділити та вирішити потенційні проблеми в окремих модулях, зменшуючи загальний ризик.

– Архітектура та дизайн системи: декомпозиція підтримує проектування модульної та добре структурованої архітектури системи.

Загальна декомпозиція проекту Android-застосунку для автоматизації бізнес-процесів торгової компанії на архітектурі MVVM представлена в таблиці 2.1.

Таблиця 2.1 – Декомпозиція проекту на архітектурі MVVM

Презентаційний шар	View
	ViewModel
Доменний рівень	Use cases
	Entities
Рівень даних	Repository
	Data sources
	Network layer
Ін'єкція залежності	Dagger2
Утиліта	Utils

Модульна декомпозиція має вирішальне значення в розробці програмного забезпечення з кількох причин. Ось чому потрібна декомпозиція модулів:

– Модульність і багаторазове використання: декомпозиція модулів сприяє модульності, що дозволяє розробляти незалежні та багаторазово використовувані компоненти. Розбиття програмної системи на модулі дозволяє розробникам працювати над різними компонентами окремо, роблячи кодову базу більш керованою, придатною для обслуговування та багаторазового використання в проектах.

– Масштабованість і технічне обслуговування: розбиваючи систему на модулі, стає легше масштабувати та підтримувати програмне забезпечення. Кожен модуль можна розробляти, тестувати та підтримувати незалежно, зменшуючи вплив змін або оновлень на всю систему. Це дозволяє легше вирішувати проблеми, виправляти помилки та додавати нові функції чи функції.

– Співпраця та командна робота: декомпозиція модулів полегшує співпрацю між командами розробників. Різні команди або розробники можуть зосередитися на конкретних модулях, забезпечуючи паралельну розробку та прискорюючи загальний процес розробки. Це також сприяє кращій координації та зменшує конфлікти під час інтеграції різних модулів разом.

– Інкапсуляція та приховування інформації: модулі забезпечують інкапсуляцію та приховування інформації, що покращує читабельність коду та зменшує складність. Завдяки інкапсуляції пов’язаних функцій у модулях стає

легше зрозуміти кодову базу та керувати нею. Модулі можуть приховувати свої внутрішні деталі реалізації, відкриваючи лише необхідні інтерфейси або API для взаємодії з іншими модулями.

– Архітектура та дизайн системи: декомпозиція модулів відіграє вирішальну роль в архітектурі та дизайні системи. Це дозволяє логічно та структурно організувати програмні компоненти, забезпечуючи чітке визначення та розподіл обов'язків між модулями. Це дозволяє створювати модульну, багаторівневу або компонентну архітектуру, підвищуючи загальну структуру та гнучкість системи.

– Тестування та налагодження: модулі роблять тестування та налагодження більш керованими. Кожен модуль можна тестувати незалежно, що дозволяє цілеспрямовано та цілеспрямовано тестувати певні функції. Це спрощує ідентифікацію та ізоляцію помилок або проблем у системі, полегшуючи їх виправлення, не впливаючи на інші модулі.

– Керування залежностями: декомпозиція модулів допомагає керувати залежностями між різними компонентами. Завдяки чіткому визначенню інтерфейсів і залежностей модулів стає легше зрозуміти взаємодію та залежності між модулями та керувати ними. Це сприяє кращому управлінню залежностями, зменшуючи ризик циклічних залежностей або непотрібного зв'язку.

– Розуміння та обслуговування системи: Розкладання системи на модулі покращує загальне розуміння та підтримку програмного забезпечення. Він надає чіткий і структурований огляд функціональності та організації системи. Це спрощує підключення нових членів команди та дозволяє легше майбутні оновлення, технічне обслуговування та вдосконалення.

Загалом, модульна декомпозиція має вирішальне значення для створення модульних, масштабованих, підтримуваних та спільних програмних систем. Це покращує можливість багаторазового використання коду, спрощує роботу з розробки та обслуговування, а також сприяє більш організованому та ефективному процесу розробки програмного забезпечення.

Модуль автентифікації користувача

– Відповідає за процеси входу, реєстрації та автентифікації користувачів.

Модуль управління замовленнями

– Керується створенням, пошуком і керуванням замовленнями.

– Включає такі функції, як створення нових замовлень, відображення деталей замовлення та оновлення статусу замовлення.

Модуль підтвердження

– Керує процесом підтвердження замовлень.

– Перевіряє деталі замовлення, створює сповіщення про підтвердження та оновлює статус замовлення.

Модуль порівняння чеків

– Інтегрується з API Нової Пошти для отримання квитанцій.

– Порівнює квитанції з деталями замовлення, щоб забезпечити точність.

– Оновлює статус квитанції як відповідний або невідповідний.

– Модуль приладової панелі

– Відображає зручну інформаційну панель із відповідною інформацією та швидким доступом до основних функцій.

– Надає огляд статусу замовлення, результати порівняння чеків та інші важливі дані.

Модуль звітності

– Створює звіти про статус замовлення, показники продажів та інші відповідні показники.

– Надає аналітику та розуміння для прийняття рішень і вдосконалення процесів.

Модуль налаштувань

– Дозволяє користувачам налаштовувати параметри програми та переваги.

– Містить параметри налаштувань сповіщень, налаштувань дисплея та інших конфігурацій для користувача.

Модуль обробки та журналювання помилок

– Керує механізмами обробки помилок і журналювання.

– Реєструє помилки, винятки та події програми для налагодження та усунення несправностей.

Модуль зберігання та керування даними

– Займається операціями зберігання та керування даними.

– Керує зберіганням інформації про користувача, деталей замовлення, даних квитанцій та інших відповідних даних.

Модуль інтеграції

– Інтегрується із зовнішніми API та службами, такими як API Nova Poshta для отримання квитанцій.

– Керує зв'язком і обміном даними між застосунком і зовнішніми системами.

Модуль інтерфейсу користувача

– Обробляє компоненти інтерфейсу користувача та взаємодії.

– Включає екрани, макети та обробку введених користувачем даних.

Модуль сповіщень

– Керує сповіщеннями та сповіщеннями для користувачів, наприклад сповіщеннями про підтвердження замовлення.

Ця декомпозиція модулів надає огляд різних компонентів або модулів у програмі. Кожен модуль зосереджується на певному наборі функціональних можливостей і обов’язків, забезпечуючи модульну та організовану структуру для розробки програмного застосунку.

Декомпозиція даних є важливою для розробки програмного забезпечення з кількох причин. Ось чому потрібна декомпозиція даних:

– Організація та структурування: Декомпозиція даних допомагає організувати та структурувати дані в системі. Розбиваючи складні дані на менші, більш керовані одиниці, стає легше зрозуміти, аналізувати та маніпулювати ними.

– Модульність і багаторазове використання: декомпозиція даних сприяє модульності та повторному використанню. Коли дані розділені на окремі компоненти, кожен компонент можна розробляти, тестувати та підтримувати незалежно. Це дозволяє легше використовувати компоненти даних у різних модулях або проектах, зменшуючи надмірність і підвищуючи ефективність.

– Масштабованість і продуктивність: декомпозиція даних допомагає досягти масштабованості та покращити продуктивність системи. Коли дані розбиваються на менші одиниці, вони стають більш масштабованими, оскільки дозволяють розподілено обробляти та зберігати. Він також підвищує продуктивність, забезпечуючи ефективний пошук, обробку та маніпулювання певними підмножинами даних.

– Безпека та контроль доступу: декомпозиція даних допомагає реалізувати заходи безпеки та механізми контролю доступу. Розбиваючи дані на менші одиниці, можна надавати або обмежувати права доступу на основі конкретних компонентів даних. Це гарантує, що чутливі або конфіденційні дані захищені та доступні лише авторизованим особам або системам.

– Цілісність і узгодженість даних: декомпозиція даних забезпечує кращу цілісність і узгодженість даних. Кожен компонент даних може мати власні правила перевірки та обмеження, що гарантує, що дані залишаються точними та узгодженими в системі. Це також спрощує перевірку даних і завдання з обслуговування, полегшуючи виявлення та вирішення проблем із даними.

– Взаємодія та інтеграція: декомпозиція даних сприяє взаємодії та інтеграції із зовнішніми системами. Розбиваючи дані на стандартизовані компоненти, стає легше обмінюватися даними з іншими системами, API або базами даних. Це сприяє бездоганній інтеграції та обміну даними між різними програмами або платформами.

– Гнучкість і адаптивність: декомпозиція даних забезпечує гнучкість і адаптивність до мінливих вимог. У міру розвитку системи або появи нових потреб у даних стає легше модифікувати або розширювати конкретні компоненти даних, не впливаючи на всю систему. Ця гнучкість дозволяє легше адаптуватися до нових вимог бізнесу або технологічного прогресу.

Таким чином, декомпозиція даних має вирішальне значення для організації, керування та оптимізації даних у програмній системі. Він покращує модульність, масштабованість, безпеку та продуктивність, одночасно забезпечуючи цілісність

даних і полегшуючи взаємодію. Він забезпечує надійну основу для створення надійних і ефективних програмних програм.

Декомпозиція даних зображена на Рисунок 2.1.

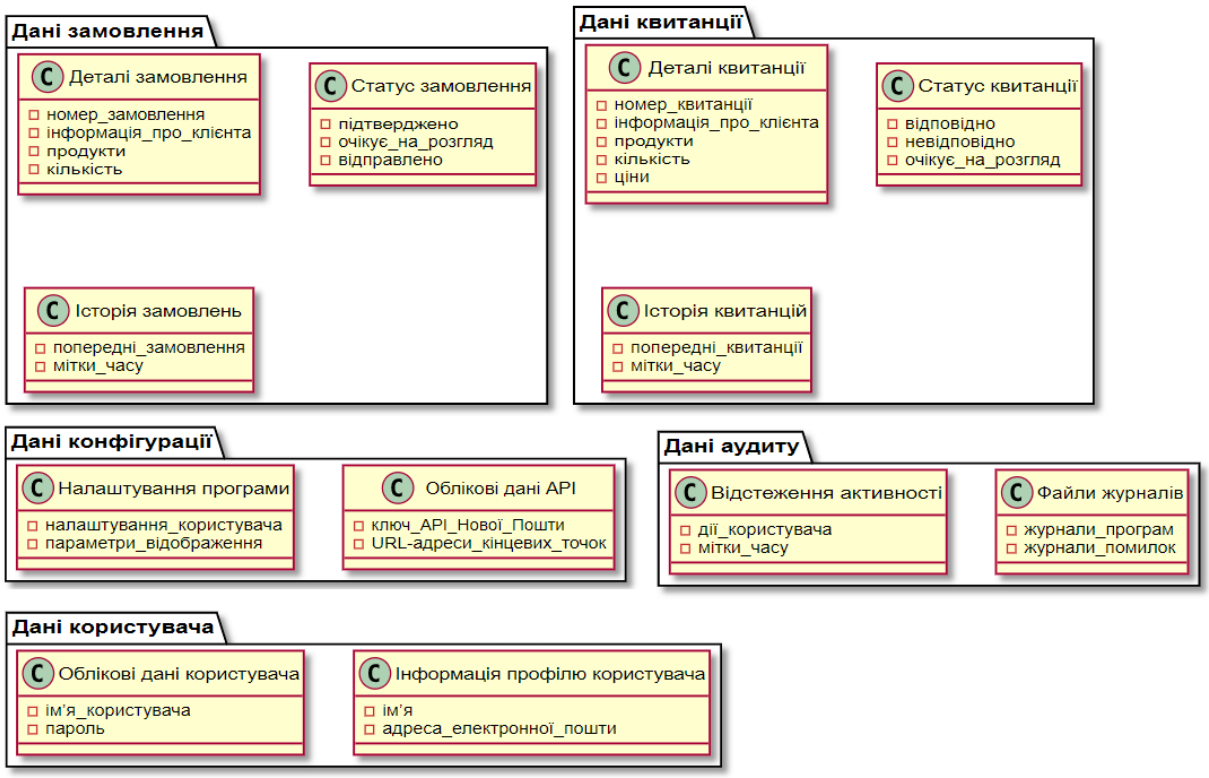


Рисунок 2.1 – Декомпозиція даних

Ця декомпозиція надає огляд різних типів даних, які може обробляти програма. Кожна категорія представляє окремий аспект керування даними програми, що дозволяє ефективно організувати та отримувати відповідну інформацію.

Декомпозиція переходів станів є вирішальним кроком у розробці програмного застосунку. Він надає чітке та структуроване уявлення про те, як різні модулі чи компоненти системи взаємодіють і розвиваються з часом. Ось кілька причин, чому декомпозиція переходів станів є важливою:

– Розуміння поведінки системи: шляхом декомпозиції переходів станів ми отримуємо повне розуміння того, як система поводиться у відповідь на різні події або дії користувача. Це допомагає нам визначити можливі стани, в яких може перебувати система, і переходи, які можуть відбуватися між цими станами.

– Розробка користувацьких інтерфейсів: переходи між станами дають цінну інформацію для розробки користувацьких інтерфейсів. Вони допомагають визначити відповідні екрани або компоненти для відображення на основі поточного стану та направляють користувача через робочий процес програми.

– Управління потоком застосунків: зміни станів дозволяють нам керувати потоком застосунків, визначаючи можливі шляхи, якими користувач може йти в системі. Це допомагає гарантувати, що користувач спрямовується до відповідних екранів або модулів на основі своїх дій.

– Обробка умов помилок: шляхом декомпозиції переходів станів ми можемо ідентифікувати та обробляти умови помилок або виняткові сценарії. Це дозволяє нам визначити, як система має реагувати на несподівані події, такі як помилки мережі, недійсні введення або збої системи.

– Забезпечення правильної поведінки системи: декомпозиція переходів станів допомагає переконатися, що система поводиться правильно та узгоджено. Це дозволяє перевірити поведінку системи на відповідність бажаним вимогам, гарантуючи, що програма функціонує належним чином.

– Полегшення тестування та налагодження: декомпозиція переходів станів забезпечує структуровану структуру для тестування та налагодження програми. Це допомагає визначити конкретні сценарії та умови, які необхідно протестувати, полегшуючи перевірку правильності та надійності системи.

Загалом, декомпозиція переходів станів покращує загальний дизайн, функціональність і надійність програмного забезпечення. Це допомагає гарантувати, що система поводиться належним чином, забезпечує безперебійну роботу користувача та ефективно обробляє різні сценарії та взаємодії користувача.

Декомпозиція переходів станів:

б) модуль входу:

1) початковий стан: екран входу;

2) переходи:

– успішний вхід: перейдіть до модуля «Інформаційна панель»;

– не вдалося ввійти: відобразити повідомлення про помилку;

в) модуль приладової панелі:

1) початковий стан: екран приладової панелі;

2) переходи:

–вибір замовлення: перейдіть до модуля «Деталі замовлення»;

–підтвердження замовлення: запустіть модуль підтвердження замовлення;

–порівняння квитанцій: запустіть модуль порівняння квитанцій;

–доступ до налаштувань: перейдіть до модуля Налаштування;

г) модуль деталей замовлення:

1) початковий стан: екран деталей замовлення;

2) переходи:

–редагування деталей замовлення: оновіть деталі замовлення;

–повернення до інформаційної панелі: поверніться до модуля інформаційної панелі;

д) модуль підтвердження замовлення:

1) початковий стан: екран підтвердження;

2) початковий стан: екран підтвердження;

3) переходи:

–успішне підтвердження: оновіть статус замовлення, поверніться на інформаційну панель;

–помилка підтвердження: відобразити повідомлення про помилку, залишатися на екрані підтвердження;

е) модуль створення ТТН та друк:

1) початковий стан: екран введення або редагування інформації для ТТН;

2) екран друку: екран на якому користувач має змогу надрукувати створену ТТН;

3) переходи:

–успішне створення: оновіть статус квитанції, поверніться на інформаційну панель;

–не вдалося створити: відобразити повідомлення про помилку, залишатися на екрані створення;

ж) модуль налаштувань:

1) початковий стан: екран налаштувань;

2) переходи:

–оновлення налаштувань: збережіть оновлені налаштування, залишайтеся на екрані налаштувань;

–повернення до інформаційної панелі: поверніться до модуля інформаційної панелі;

з) модуль налаштувань торгового обладнання:

1) початковий стан: екран вибору типу обладнання;

2) екран обладнання:

–принтери: тут користувач можу створити або редагувати вже створений принтер;

–сканери: тут користувач можу створити або редагувати вже створений сканер;

–ваги: тут користувач можу створити або редагувати вже створені ваги;

3) переходи:

–оновлення налаштувань: збережіть оновлені налаштування, залишайтеся на екрані налаштувань;

–повернення до інформаційної панелі: поверніться до модуля інформаційної панелі.

Ця декомпозиція ілюструє можливі переходи між різними модулями або екранами програми. Кожен модуль має власний початковий стан і визначає переходи, які можуть відбуватися на основі дій користувача або системних подій. Це допомагає забезпечити чітке розуміння потоку та взаємодії між різними частинами програми.

2.3 Опис залежностей

Опис міжмодульних залежностей важливий з наступних причин:

- Розуміння архітектури системи та взаємодії модулів.
- Визначення каналів зв'язку та механізмів обміну даними.
- Управління залежностями модулів і забезпечення плавної інтеграції.
- Можливість паралельної розробки та модульного проектування.
- Полегшення обслуговування та оновлення системи.
- Підтримка системної інтеграції та взаємодії.

Таким чином, опис міжмодульних залежностей допомагає проектувати, інтегрувати та підтримувати добре структуровану та ефективну систему.

На основі раніше створених декомпозицій наведемо приклад блоку міжмодульних залежностей для проекту Android-застосунку для автоматизації бізнес-процесів торгової компанії з використанням архітектури MVVM:

а) модуль автентифікації користувача:

1) немає;

б) модуль управління замовленнями:

1) модуль автентифікації користувача;
2) модуль підтвердження;

в) модуль підтвердження:

1) модуль автентифікації користувача;
2) модуль керування замовленнями;

г) модуль порівняння чеків:

1) модуль автентифікації користувача;
2) модуль керування замовленнями;

д) модуль приладової панелі:

1) модуль автентифікації користувача;
2) модуль керування замовленнями;
3) модуль підтвердження;
4) модуль порівняння квитанцій;

е) модуль звітності:

1) модуль автентифікації користувача;

2) модуль керування замовленнями;

ж) модуль налаштувань:

1) модуль автентифікації користувача;

з) модуль обробки та журналювання помилок:

1) немає;

и) модуль зберігання та керування даними:

1) модуль автентифікації користувача;

2) модуль керування замовленнями;

3) модуль підтвердження;

4) модуль порівняння квитанцій;

к) модуль інтеграції:

1) модуль автентифікації користувача;

2) модуль керування замовленнями;

3) модуль порівняння квитанцій;

л) модуль інтерфейсу користувача:

1) модуль автентифікації користувача;

2) модуль керування замовленнями;

3) модуль підтвердження;

4) модуль порівняння квитанцій;

м) модуль інтерфейсу користувача:

1) модуль автентифікації користувача;

2) модуль керування замовленнями;

3) модуль підтвердження;

4) модуль порівняння квитанцій;

н) модуль сповіщень:

1) модуль автентифікації користувача;

2) модуль підтвердження.

Цей блок міжмодульних залежностей окреслює зв’язки між різними модулями в проекті. Модулі залежать від інших модулів, коли їм потрібні їхні

функції чи дані. Міжмодульні залежності гарантують, що модулі можуть спілкуватися та ефективно взаємодіяти один з одним для виконання бажаних функцій програми.

Опис залежностей даних важливий з таких причин:

– Розуміння потоку даних і зв’язків.

– Виявлення залежностей між модулями або процесами.

– Управління доступом до даних і забезпечення безпеки.

– Підтримка узгодженості даних у всій програмі.

– Забезпечення модульності та повторного використання.

– Сприяння ефективному тестуванню та налагодженню.

– Сприяння співпраці та спілкуванню між членами команди.

Підсумовуючи, чіткий опис залежностей даних допомагає ефективно керувати та організовувати дані, забезпечуючи безперебійну роботу та надійність програми.

Опис залежностей даних.

Залежність даних користувача:

– Опис: пов’язані з користувачем дані, як-от облікові дані користувача, інформація профілю та маркери авторизації;

– Залежності: немає

Залежність даних замовлення:

– Опис: дані, пов’язані із замовленнями, зокрема деталі замовлення, інформація про клієнта та статус замовлення;

– Залежності: Залежність даних користувача

Залежність даних підтвердження:

– Опис: дані, пов’язані з процесом підтвердження, включаючи статус підтвердження, позначки часу та пов’язану з підтвердженням інформацію;

– Залежності: залежність даних користувача, залежність даних замовлення

Залежність даних квитанції:

– Опис: дані, пов’язані з квитанціями, включаючи деталі квитанції, платіжну інформацію та статус квитанції;

– Залежності: Залежність даних користувача, Залежність даних замовлення, Залежність даних API Нова Пошта

Залежність даних API Нова Пошта:

– Опис: дані, отримані з API «Нова пошта», як-от деталі відправлення, інформація про відстеження та статус доставки;

– Залежності: немає

Залежність даних інформаційної панелі:

– Опис: дані, необхідні для модуля інформаційної панелі, включаючи підсумкову інформацію про замовлення, статистику та ключові показники ефективності;

– Залежності: залежність даних користувача, залежність даних замовлення, залежність даних підтвердження, залежність даних квитанції

Залежність звітних даних:

– Опис: дані, що використовуються для створення звітів і аналітики, включаючи дані про продажі, історію замовлень і показники ефективності;

– Залежності: залежність даних користувача, залежність даних замовлення

Налаштування Залежність даних:

– Опис: дані, пов’язані з параметрами програми та параметрами користувача, наприклад параметри сповіщень і параметри налаштування;

– Залежності: Залежність даних користувача

Залежність від обробки помилок і реєстрації даних:

– Опис: дані, пов’язані з обробкою та веденням журналів помилок, включаючи журнали помилок, деталі винятків та інформацію про відстеження помилок;

– Залежності: немає

Залежність даних інтеграції:

– Опис: дані, необхідні для інтеграції із зовнішніми системами, API або службами, включно з форматами обміну даними та даними, що стосуються інтеграції;

– Залежності: залежність даних користувача, залежність даних замовлення, залежність даних квитанції

Залежність даних сповіщення:

– Опис: дані, які використовуються для надсилання сповіщень і попереджень користувачам, зокрема вміст сповіщень, статус доставки та налаштування користувача;

– Залежності: залежність даних користувача, залежність даних підтвердження

Залежності даних представляють зв’язки та залежності між різними об’єктами даних та їхніми відповідними модулями або процесами в програмі Android. Вони забезпечують доступність необхідних даних для обробки, аналізу та представлення, що дозволяє автоматизувати бізнес-процеси торгової компанії.

2.4 Аналіз та вибір технологій і методів реалізації застосунку

Ось деякі з найпопулярніших бібліотек, які використовуються в розробці застосунків для Android:

– Retrofit: клієнтська бібліотека HTTP із захищеним типом для виконання мережевих запитів у програмах Android. Це спрощує процес взаємодії з RESTful API.

– Glide: бібліотека для завантаження та кешування зображень, яка забезпечує ефективні та плавні можливості завантаження зображень, включаючи кешування, зміну розміру та перетворення.

– Room: бібліотека абстракцій бази даних SQLite, яка спрощує операції з базою даних у програмах Android. Він забезпечує об’єктно-орієнтований інтерфейс і підтримує перевірку запитів під час компіляції.

– Dagger: структура впровадження залежностей для керування залежностями між різними компонентами програми Android. Це допомагає писати модульний і тестований код.

– RxJava: бібліотека реактивного програмування, яка забезпечує асинхронне та кероване подіями програмування. Він забезпечує потужні абстракції для обробки потоків даних і сприяє паралелізму.

– Gson: бібліотека для перетворення об’єктів Java у подання JSON і навпаки. Це спрощує процес розбору даних JSON, отриманих від API, або серіалізації об’єктів Java у JSON.

– OkHttp: потужна клієнтська бібліотека HTTP, яка підтримує HTTP/2 і пул з’єднань. Він надає ефективні та гнучкі мережеві можливості для програм Android.

– Mockito: популярна глузлива структура, яка використовується для модульного тестування в програмах Android. Це дозволяє розробникам створювати макетні об’єкти для тестування залежностей і перевірки поведінки.

– Butter Knife: бібліотека прив’язки представлень, яка спрощує процес прив’язки представлень до відповідних ідентифікаторів у макетах Android XML. Це зменшує шаблонний код для доступу до представлень.

– EventBus: бібліотека шин подій публікації та підписки, яка спрощує зв’язок між різними компонентами програми Android. Це забезпечує слабке з’єднання та роз’єднання компонентів.

– Timber: бібліотека журналювання, яка забезпечує простий у використанні інтерфейс для реєстрації повідомлень про налагодження та помилки в програмах Android. Він пропонує розширені можливості журналювання та дозволяє налаштовувати.

– LiveData: клас власника даних з урахуванням життєвого циклу, наданий компонентами архітектури Android. Це спрощує процес спостереження та оновлення даних реактивним способом.

– Розберемо деякі бібліотеки більш детально тому що вони будуть відігравати важливу роль при написані застосунку.

Для забезпечення зручного та ефективного способу отримання даних ми будемо використовувати бібліотеку Retrofit.

Retrofit - це бібліотека для роботи з RESTful API на платформі Android. Вона дозволяє легко взаємодіяти з веб-службами та обмінюватись даними з сервером.

Бібліотека забезпечує можливість визначати структуру запитів та відповідей у вигляді інтерфейсу, а згодом згенерувати необхідний код для їх виконання.

Крім цього, Retrofit дозволяє автоматично перетворювати отримані дані з формату JSON у об'єкти в програмі та навпаки. Це значно спрощує роботу зі збереженням та передачею даних.

Ще одним важливим аспектом при розробзв ПЗ є правильна та зручна взаємодія з БД. В йьому на допоможе бібліотека Room, вона потрібна для легкого доступу до бази даних та збереження даних.

Для роботи з Room бібліотекою необхідно визначити сутності, які будуть використовуватися в базі даних. Тобто потрібно створити класи, що відображають таблиці бази даних та відповідають за зберігання та отримання даних.

Наступним кроком для роботи з Room є DAO (Data Access Object) - це інтерфейс, який дозволяє виконувати CRUD-операції над таблицями бази даних. Це дає можливість використовувати SQL-запити та Room анотації для виконання запитів до бази даних. Він дозволяє робити операції вставки, оновлення, видалення та вибірки даних з бази даних.

Крім того, Room для здійснення запитів до бази даних у фоновому режимі використовує AsyncTask, який дозволяє виконувати завдання в окремому потоці. Це робить застосунок більш продуктивним та ефективним, оскільки він не блокує головний потік виконання застосунку, коли ми виконуємо запит до бази даних.

Далі не менш йде не менш важлива бібліотека а саме Dagger.

Dagger – це популярна платформа впровадження залежностей для розробки застосунків Android. Він забезпечує надійний і ефективний спосіб керування залежностями між різними компонентами програми. Ось як Dagger може допомогти у написанні програм.

Ін’єкція залежностей: Dagger полегшує практику ін’єкції залежностей (DI) у програмах Android. DI – це шаблон проектування, який сприяє слабкому зв’язку та модульній розробці шляхом відокремлення створення та керування залежностями від залежних класів. За допомогою Dagger ви можете визначати залежності та

зв’язки між ними, а Dagger піклується про створення та надання екземплярів цих залежностей, коли це необхідно.

Модульність коду: за допомогою Dagger ви можете розбити свою програму на менші модульні компоненти. Кожен компонент представляє певну функціональність або функцію вашої програми. Ці компоненти можна легко комбінувати або замінювати, сприяючи модульності коду, повторному використанню та тестуванню.

Безпека під час компіляції: Dagger виконує ін’єкцію залежностей під час компіляції, а не під час виконання. Він генерує код на основі визначених залежностей, а будь-які помилки, пов’язані із залежностями, виловлюються під час процесу компіляції. Це гарантує, що залежності належним чином розв’язані та доступні під час виконання, зменшуючи ймовірність помилок під час виконання та підвищуючи стабільність коду.

Спрощений графік залежностей: Dagger допомагає керувати складними графіками залежностей, надаючи чіткий і чіткий спосіб визначення залежностей. Графік залежностей представляє зв’язки між різними модулями, компонентами та залежностями у вашій програмі. Dagger автоматично вирішує ці залежності на основі графіка, полегшуючи керування та розуміння потоку залежностей у вашій програмі.

Оптимізація продуктивності: підхід Dagger під час компіляції та згенерований код забезпечують оптимізоване та ефективне впровадження залежностей. Це усуває потребу в рамках впровадження залежностей на основі відображення, що призводить до покращення продуктивності програми.

Підтримка тестування: Dagger спрощує модульне тестування, дозволяючи легко імітувати або замінювати залежності. Впроваджуючи фіктивні залежності у ваші класи під час тестування, ви можете ізолювати тестовані компоненти та перевіряти їхню поведінку. Це полегшує написання комплексних модульних тестів для вашої програми.

Інтеграція з компонентами архітектури Android: Dagger добре інтегрується з іншими фреймворками та бібліотеками розробки Android, такими як компоненти

архітектури Android. Він повністю інтегрується з ViewModel, LiveData та іншими компонентами, забезпечуючи послідовний і ефективний спосіб обробки залежностей у вашій програмі.

Підсумовуючи, Dagger – це потужна платформа для ін’єкцій залежностей, яка допомагає писати модульні програми Android, які можна підтримувати та тестувати. Він сприяє модульності коду, безпеці під час компіляції та спрощує керування складними графами залежностей. Використовуючи Dagger, ви можете покращити організацію коду, підвищити продуктивність програми та полегшити модульне тестування.

Загальний висновок полягає в тому, що розглянуті бібліотеки - Retrofit, Room і Dagger - є важливими компонентами при розробці застосунків для платформи Android. Кожна з цих бібліотек виконує свою унікальну функцію і забезпечує певні переваги.

Бібліотека Retrofit дозволяє легко взаємодіяти з веб-службами та обмінюватись даними з сервером, забезпечуючи зручний спосіб отримання даних. Вона автоматично перетворює дані з формату JSON у об'єкти в програмі та навпаки, спрощуючи роботу зі збереженням та передачею даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Детальне проектування модулів

Модуль автентифікації – важливий компонент Android-застосунку, розроблений для автоматизації бізнес-процесів торгової компанії. Його головна мета – полегшити безпечну автентифікацію користувачів і забезпечити доступ до функцій і ресурсів програми лише авторизованим особам.

Модуль автентифікації містить різні функції та компоненти для забезпечення надійного механізму автентифікації. Він відповідає найкращим галузевим практикам і стандартам безпеки для захисту облікових даних користувача та захисту від несанкціонованого доступу. Деякі ключові аспекти модуля автентифікації включають:

– Вхід: зареєстровані користувачі можуть увійти в програму за допомогою своїх облікових даних. Модуль перевіряє надану інформацію та надає доступ авторизованим користувачам, зберігаючи при цьому конфіденційність конфіденційних даних.

– Аутентифікація на основі маркерів: для підвищення безпеки та покращення взаємодії з користувачем модуль використовує автентифікацію на основі маркерів. Після успішного входу генерується унікальний маркер, який пов’язується з сеансом користувача, який використовується для перевірки наступних запитів.

– Керування сеансами: модуль ефективно керує сеансами користувачів, щоб забезпечити безперебійну та безпечну роботу користувача. Він відстежує інформацію про сеанси, включаючи мітки часу входу/виходу та тривалість активного сеансу, щоб відстежувати та контролювати доступ користувачів.

Модуль «Нова Пошта» містить різноманітні функціональні можливості та компоненти для ефективної взаємодії з API та отримання актуальної інформації. Деякі ключові аспекти модуля інтеграції включають:

– Підтвердження замовлення: модуль інтеграції дозволяє застосунку підтверджувати замовлення за допомогою API Нової Пошти. Він надає необхідні функції для надсилання деталей замовлення, як-от інформації про клієнта, специфікації пакета та параметрів доставки, до API для обробки та підтвердження.

– Синхронізація даних: модуль інтеграції забезпечує безперебійну синхронізацію даних між програмою та API Нової Пошти. Він отримує відповідні дані з API, такі як оновлення статусу доставки або інформацію про відстеження, і відповідно оновлює базу даних програми або інтерфейс користувача. Це забезпечує видимість у режимі реального часу та відстеження замовлень у програмі.

– Обробка помилок і звітування: модуль обробляє будь-які помилки або винятки, які можуть виникнути під час процесу інтеграції. Він надає застосунку відповідні повідомлення про помилки та сповіщення, дозволяючи користувачам або адміністраторам виконувати необхідні дії або вирішувати будь-які проблеми, які можуть виникнути.

Завдяки вбудованому модулю інтеграції «Нова Пошта» Android-застосунок безперешкодно взаємодіє з API «Нова Пошта», забезпечуючи ефективне підтвердження замовлення та порівняння квитанцій. Ця інтеграція покращує автоматизацію бізнес-процесів торгової компанії, оптимізує логістичні операції та підвищує загальну ефективність управління замовленнями.

Модуль «Підтвердження замовлення» – важливий компонент Android-застосунку для автоматизації бізнес-процесів торгової компанії. Його основна функція полягає в тому, щоб полегшити безперебійне підтвердження замовлень у програмі, забезпечуючи точну та ефективну обробку замовлень клієнтів.

Модуль підтвердження замовлення містить різноманітні функціональні можливості та функції, які забезпечують плавний процес підтвердження замовлення. Деякі ключові аспекти модуля включають:

– Керування замовленнями: модуль забезпечує інтуїтивно зрозумілий інтерфейс користувача, який дозволяє користувачам переглядати вхідні замовлення та керувати ними. Він відображає важливі деталі замовлення, такі як інформація про клієнта, елементи замовлення, кількість і параметри доставки.

– Перевірка замовлення: модуль перевіряє дані замовлення, щоб забезпечити їх точність і повноту. Він перевіряє, чи надано всю необхідну інформацію, таку як адреса доставки, контактні дані та платіжні дані. Ця перевірка допомагає запобігти помилкам і гарантує, що підтверджені замовлення є дійсними та можуть бути правильно оброблені.

– Історія замовлень і відстеження: модуль підтримує повну історію замовлень, дозволяючи користувачам отримувати доступ і відстежувати статус підтверджених замовлень. Він забезпечує централізований перегляд усіх попередніх замовлень, включаючи їхній поточний статус, деталі доставки та будь-які оновлення або зміни, внесені протягом процесу виконання замовлення.

Завдяки модулю підтвердження замовлення застосунок Android оптимізує та автоматизує процес підтвердження замовлення для торгової компанії. Він підвищує точність, ефективність і задоволеність клієнтів, надаючи бездоганний і зручний інтерфейс для керування та підтвердження замовлень клієнтів.

3.2 Програмна реалізація модулів

Розпочнемо з модуля аутентифікації цей модуль є критично важливим адже він забезпечує безпечну роботу користувача. У нашому модулі відсутня можливість реєстрації, тому що наш застосунок є комерційним на не буде в загально доступному доступі. Тобто будуть клієнти які отримують ліцензію та будуть додані до системи, їм буде виданий логін пароль та ключ ліцензії, та допомогою яких вони будуть входити в застосунок. Метод для входу користувача можна побачити нижче.

```
private void checkLogin(final String email, final String password) {
        if (mSettings.contains(AppConfig.PREF_ADDRESS) &&
mSettings.contains(AppConfig.PREF_API)) {
            String uri = mSettings.getString(AppConfig.PREF_ADDRESS, "");
            String api = mSettings.getString(AppConfig.PREF_API, "");
            ExampleApp.GetRestAPI(uri).getData(AppConfig.API_AUTHORIZATION, "Basic
" + pl, loc, api, "ConfirmDocs/" + BuildConfig.VERSION_NAME).enqueue(new
Callback<JsonElement>() {
                @Override
                public void onResponse(Call<JsonElement> call,
Response<JsonElement> response) {
```

```
                    if (response.isSuccessful()) {
                        Boolean Status =
response.body().getAsJsonObject().get("success").getAsBoolean();
                        if (Status) {
                            String user_token =
response.body().getAsJsonObject().get("token").getAsString();
                            String token =
Base64.encodeToString(user_token.getBytes(), Base64.NO_WRAP);
                            String name =
response.body().getAsJsonObject().get("username").getAsString();
                            session.setLogin(true);

ExampleApp.getDataBaseManager().getUserDAO().insert(new
UserDTO(UUID.randomUUID().toString(), name, email, password, token));
                        }
                    } else {
                        ExampleApp.getDataBaseManager().getUserDAO().deleteAll();
                        if(response.code()==401){
                            Toast.makeText(LoginActivity.this,
getString(R.string.text_error_login), Toast.LENGTH_LONG).show();
                        }
                    }
                }

                @Override
                public void onFailure(Call<JsonElement> call, Throwable t) {
                    session.setLogin(false);
                    ExampleApp.getDataBaseManager().getUserDAO().deleteAll();
                }
            });
        }
    }
}
```

Цей код бере усю введену інформацію та відправляє її на сервер, від сервера приходить відповідь а саме користувач авторизований або ні. В першому випадку також приходить токен який зберігається у застосунку і при наступному входженні відбудеться автологінізація. Також цей токен потрібен для отримання замовлень та будь якої інформації від сервера і у випадку якщо щось сталось з токеном наприклад закінчився його термін користувач буде викинутий з системи.

Приклад методу який викликається для виходу з системи наведений нижче.

```
public void logoutUser(Activity activity) {
    SessionManager.getInstance().setLogin(false);
    getDataBaseManager().getUserDAO().deleteAll();

    Intent intent = new Intent(activity, LoginActivity.class);
    activity.startActivity(intent);
    activity.finish();
}
```

Наступний модуль це модуль взаємодії з Api Нової Пошти. Для нашого застосунку потрібно декілька основних можливостей Api Нової пошти а саме метод

для підтвердження замовлень, отримання інформації про кабінет відправника, коди помилок та різноманітні довідники.

Для початку розглянемо метод для створення накладної на доставку, для створення ТТН потрібно зібрати усю необхідну інформацію, а саме інформацію про товар, розміри тощо, інформацію про відправника та отримувача та інформацію про оплату. Метод для створення ТТН відображений далі.

```
public void createEn(DocOrderItem orderData, boolean useMinPrice, int
backwardPayerType, RequestInterfaceInfo listener) {
    String jsonToCreateEn = getJsonToCreateEn(orderData, useMinPrice,
backwardPayerType);
    String loc = String.valueOf(Resources.getSystem().getConfiguration().locale);
    getRestApi().getData(jsonToCreateEn, loc).enqueue(new Callback<JsonElement>()
{
        @Override
        public void onResponse(@NonNull Call<JsonElement> call, @NonNull
Response<JsonElement> response) {
            try {
                if (response.isSuccessful()) {
                    boolean  success =
response.body().getAsJsonObject().get("success").getAsBoolean();
                    if (success) {
                        JsonArray dataArray =
response.body().getAsJsonObject().getAsJsonArray("data");
                        String EnRef =
dataArray.get(0).getAsJsonObject().get("Ref").getAsString();
                        String EnNumber =
dataArray.get(0).getAsJsonObject().get("IntDocNumber").getAsString();
                        EventBus.getDefault().post(new
BustMessageEvent(BustMessageEvent.BUST_EVENT_NP_CREATED, new String[]{EnRef,
EnNumber, enDate, orderData.getDelivery().getDelivery_cabinet()}));

                    }
                }
            } catch (Exception e) {
                Toast.makeText(ExampleApp.getInstance().getApplicationContext(),
ExampleApp.getInstance().getApplicationContext().getString(R.string.text_error_sen
d), Toast.LENGTH_LONG).show();

StackTraceManager.getInstance().addStackTrace(StackTraceManager.getStackTrace(e));
            }
        }
        @Override
        public void onFailure(@NonNull Call<JsonElement> call, @NonNull Throwable
t) {
            EventBus.getDefault().post(new
BustMessageEvent(BustMessageEvent.BUST_EVENT_NP_API_ERROR_MESSAGE,
t.getLocalizedMessage()));
        }
    });
}
```

Одним з параметрів цього метода є документ в якому зберігається уся інформація про замовлення, далі ми передаємо цей документ у метод getJsonToCreate та отримуємо готовий Json для створення ТТН.

Тепер розглянемо метод який отримує коди помилок Нової Пошти, цей метод є дуже важливим тому що при будь якій помилці Нова Пошта просто віддає код помилки далі ми повинні знайти його в базі та показати користувачу щоб він виконав певні дії. Коди помилок оновлюються регулярно тому застосунок поновлює їх раз на десять днів. Метод для отримання кодів помилок відображено нижче.

```
public void updateErrorsMessages() {
String loc = String.valueOf(Resources.getSystem().getConfiguration().locale);
JsonObject jsonReq = new JsonObject();
CabinetDeliveryDTO cabinetDeliveryDTO
=ExampleApp.getDataBaseManager().getCabinetDeliveryDAO().getDefaultCabinet();
        if (cabinetDeliveryDTO != null) {
    jsonReq.addProperty("apiKey", cabinetDeliveryDTO.getApiKey());
    jsonReq.addProperty("modelName", "CommonGeneral");
    jsonReq.addProperty("calledMethod", "getMessageCodeText");
    jsonReq.add("methodProperties", new JsonObject());
    try {
        Response<JsonElement> response = getRestApi().getData(jsonReq,
loc).execute();
        if (response.isSuccessful()) {
                boolean  success =
response.body().getAsJsonObject().get("success").getAsBoolean();
            if (success) {
                NPErrorDAO npErrorDAO=
ExampleApp.getDataBaseManager().getNPErrorDAO();
                for (int i = 0; i <= messagesArray.size() - 1; i++) {
                    JsonObject oneMessageObject =
messagesArray.get(i).getAsJsonObject();
                    if (i % 20 == 0) publishProgress(i, messagesArray.size());
                }
                EventBus.getDefault().post(new
BustMessageEvent(BustMessageEvent.BUST_EVENT_CLOSE_PROGRESS, true));


NovaPoshtaManager.getManager().setLastDateUpdateErros(System.currentTimeMillis());
            }
        }
    } catch (IOException e) {
        EventBus.getDefault().post(new
BustMessageEvent(BustMessageEvent.BUST_EVENT_CLOSE_PROGRESS, true));
    }
}
```

Цей метод отримує коди помилок та записує їх в базу даних щоб в майбутньому коли нам прийде який код помилки ми могли розшифрувати його.

Далі розглянемо метод для підтвердження замовлення. Для того щоб зарезервувати замовлення за собою його потрібно взяти в роботу тобто перед початком підтвердження потрібно надіслати запит що замовлення в роботі. Також потрібно при виході з замовлення надсилати його актуальний стан. Щоб якщо хтось інший захоче згодом продовжити підтвердження замовлення він мав

найактуальнішу інформацію. Для оновлення стану замовлення використовується нижче наведений код.

```
public void sendEnRef(DocOrderItem orderItem, Context context, Activity activity,
int confirmed, int id) {
    String jsonToSend = getJson(orderItem);
    String token = getToken();
    String loc = String.valueOf(Resources.getSystem().getConfiguration().locale);
    if (preferences[0].contains(AppConfig.PREF_ADDRESS) &&
preferences[0].contains(AppConfig.PREF_API)) {
        String uri = preferences[0].getString(AppConfig.PREF_ADDRESS, "");
        ExampleApp.GetRestAPI(uri).sendEnRef(AppConfig.API_POST_CONFIRM_ORDERS,
"Bearer " + token, loc, jsonToSend, api, "ConfirmDocs/" +
BuildConfig.VERSION_NAME).enqueue(new Callback<JsonObject>() {
            @Override
            public void onResponse(@NonNull Call<JsonObject> call, @NonNull
Response<JsonObject> response) {
                try {
                    if (response.isSuccessful()) {
                        boolean success =
response.body().getAsJsonObject().get("success").getAsBoolean();
                        if (success) {
                            EventBus.getDefault().post(new
BustMessageEvent(BustMessageEvent.BUST_EVENT_SEND_TTN, "4"));
                        } else {
                            EventBus.getDefault().post(new
BustMessageEvent(BustMessageEvent.BUST_EVENT_SEND_TTN, "2"));
                        }
                    }
                } catch (Exception e) {

StackTraceManager.getInstance().addStackTrace(StackTraceManager.getStackTrace(e));
                }
            }

            @Override
            public void onFailure(@NonNull Call<JsonObject> call, @NonNull
Throwable t) {
            }
        });
```

Наведені раніше модулі є найбільш важливими тому що вони виконують головну логіку та суть застосунку.

3.3 Детальне проектування даних

Детальне проектування даних має вирішальне значення для розробки програмного забезпечення з кількох причин

Цілісність даних: детальне проектування даних гарантує, що дані, що зберігаються в базі даних програми, є точними, послідовними та надійними. Він

визначає структуру, зв’язки та обмеження об’єктів даних, запобігаючи невідповідності та пошкодження даних.

Організація даних: детальна інженерія даних допомагає організовувати дані логічно та ефективно. Він визначає відповідні таблиці, атрибути та зв’язки, необхідні для ефективного зберігання та отримання даних.

Доступність даних: добре продумана структура даних забезпечує ефективний пошук і маніпулювання даними. Завдяки визначенню відповідних індексів, ключів і зв’язків можна отримати швидкий доступ до даних, що сприяє підвищенню продуктивності програми.

Аналіз даних і звітність: добре продумана структура даних забезпечує ефективний аналіз даних і звітність. Належним чином структуруючи дані, стає легше отримувати значущі ідеї та створювати звіти для прийняття рішень.

Розглянемо таблицю користувача.

```
@Entity(tableName = "users")
data class UserDTO(
        @PrimaryKey
        @ColumnInfo(name = "guid") var guid: String,
        @ColumnInfo(name = "name") var name: String,
        @ColumnInfo(name = "login") var login: String,
        @ColumnInfo(name = "password") var password: String,
        @ColumnInfo(name = "token") var token: String
)
```

Наданий фрагмент коду представляє клас моделі даних під назвою «UserDTO», який використовується для представлення даних користувача в контексті таблиці бази даних. Давайте розберемо код і опишемо кожну частину:

– guid: атрибут «guid» має тип String і призначений для зберігання унікального ідентифікатора користувача. Він служить первинним ключем для таблиці «користувачі».

– name: Атрибут «name» має тип String і призначений для зберігання імені користувача.

– login: атрибут «login» має тип String і використовується для зберігання облікових даних користувача.

– password: атрибут «password» має тип String і використовується для зберігання пароля, пов'язаного з логіном користувача.

– token: атрибут «token» має тип String і призначений для зберігання маркера, пов'язаного з користувачем. Його можна використовувати для автентифікації або авторизації.

Загалом, клас моделі «UserDTO» представляє структуру та атрибути сутності користувача в базі даних. Він містить поля для унікального ідентифікатора користувача, імені, логіна, пароля та токена. Цю модель можна використовувати для зберігання та отримання даних користувача з відповідної таблиці «користувачі» в базі даних.

Розглянемо таблицю кодів помилок нової пошти.

```
@Entity(tableName = "np_error")
data class NPErrorDTO(
        @PrimaryKey @ColumnInfo(name = "code") var code: String,
                    @ColumnInfo(name = "massage") var massage: String,
                    @ColumnInfo(name = "massage_ru") var massageRU: String,
                    @ColumnInfo(name = "massage_ua") var massageUA: String
)
```

Наданий фрагмент коду представляє клас моделі даних під назвою «NPErrorDTO», який використовується для представлення даних кодів помилок в контексті таблиці бази даних. Давайте розберемо код і опишемо кожну частину:

– code: атрибут «code» має тип String і призначений для зберігання коду помилки, пов’язаного з інтеграцією Нової Пошти. Він служить первинним ключем для таблиці «np_error».

– massage: атрибут «massage» має тип String і використовується для зберігання повідомлення про помилку англійською мовою, пов’язаного з кодом помилки.

– massageRU: атрибут «massageRU» має тип String і використовується для зберігання повідомлення про помилку російською мовою, пов’язаного з кодом помилки.

– massageUA: атрибут «massageUA» має тип String і використовується для зберігання повідомлення про помилку українською мовою, пов'язаного з кодом помилки.

Загалом модель «NPErrorDTO» представляє об’єкт помилки, характерний для інтеграції «Нова пошта» в структуру даних програми. Містить атрибути для

зберігання коду помилки, повідомлень про помилку англійською, російською та українською мовами.

Розглянемо таблицю додаткового обладнання всього їх три види а саме принтери, ваги та сканери. Всі три таблиці дуже схожі тому розглянемо одну з них а саме принтери.

```
@Entity(tableName = "printers")
data class PrinterDTO(
        @PrimaryKey
        @ColumnInfo(name = "guid") var guid: String,
        @ColumnInfo(name = "type_connection") var typeConnection: Int,
        @ColumnInfo(name = "name") var name: String,
        @ColumnInfo(name = "address") var address: String,
        @ColumnInfo(name = "active") var active: Int,
        @ColumnInfo(name = "is_use_scale") var isUseScale: Boolean,
        @ColumnInfo(name = "type_paper") var typePaper: Int,
        @ColumnInfo(name = "type_printer") var typePrinter: Int,
        @ColumnInfo(name = "port") var port: String
)
```

Розглянемо структуру конкретніше.

– guid: атрибут «guid» має тип String і призначений для зберігання унікального ідентифікатора принтера. Він служить первинним ключем для таблиці «принтерів».

– typeConnection: атрибут «typeConnection» має тип Int і використовується для вказівки типу підключення для принтера. Він може мати різні цілі значення, що відповідають різним типам підключення.

– name: атрибут «name» має тип String і використовується для зберігання назви або ідентифікатора принтера.

– адреса: Атрибут «address» має тип String і використовується для збереження адреси або мережевого розташування принтера.

– active: атрибут «active» має тип Int і використовується для вказівки стану принтера (активний чи неактивний). Він може мати різні цілі значення, що представляють активний стан принтера.

– isUseScale: атрибут «isUseScale» має тип Boolean і використовується для визначення того, чи використовує принтер ваги для зважування предметів.

– typePaper: атрибут «typePaper» має тип Int і використовується для визначення типу паперу, який використовується принтером. Він може мати різні цілі значення, що представляють різні типи паперу.

– typePrinter: атрибут «typePrinter» має тип Int і використовується для вказівки типу принтера. Він може мати різні цілі значення, що представляють різні типи принтерів.

– port: атрибут «port» має тип String і використовується для зберігання номера порту або ідентифікатора, пов'язаного з принтером.

Також в застосунку буде зберігатись інформація про кабінети доставки. Ця інформація потрібна щоб користувач кожного разу не вводив свої дані для відправки. Побачити структуру таблиці можна далі.

```
@Entity(tableName = "cabinet_delivery")
data class CabinetDeliveryDTO(
        @PrimaryKey @ColumnInfo(name = "guid") var guid: String,
        @ColumnInfo(name = "api_key") var apiKey: String,
        @ColumnInfo(name = "contact_sender_ref") var contactSenderRef: String,
        @ColumnInfo(name = "contact_sender_name") var contactSenderName: String,
        @ColumnInfo(name = "sender_phone") var senderPhone: String,
        @ColumnInfo(name = "sender_ref") var senderRef: String,
        @ColumnInfo(name = "sender_name") var senderName: String,
        @ColumnInfo(name = "sender_city_ref") var senderCityRef: String,
        @ColumnInfo(name = "sender_department_ref") var senderDepartmentRef:
String,
        @ColumnInfo(name = "delivery_services_id") var deliveryServiceId: String,
        @ColumnInfo(name = "title", defaultValue = "") var title: String
)
```

Розглянемо структуру конкретніше.

– guid: атрибут «guid» має тип String і призначений для зберігання унікального ідентифікатора для кабінету доставки. Він служить первинним ключем для таблиці 'cabinet_delivery'.

– apiKey: атрибут «apiKey» має тип String і використовується для зберігання ключа API, пов’язаного з кабінетом доставки.

– contactSenderRef: атрибут «contactSenderRef» має тип String і використовується для зберігання посилання на відправника контакту.

– contactSenderName: Атрибут «contactSenderName» має тип String і використовується для зберігання імені відправника контакту.

– senderPhone: атрибут «senderPhone» має тип String і використовується для зберігання номера телефону відправника.

– senderRef: атрибут «senderRef» має тип String і використовується для зберігання посилання на відправника.

– senderName: атрибут «senderName» має тип String і використовується для зберігання імені відправника.

– senderCityRef: Атрибут «senderCityRef» має тип String і використовується для зберігання посилання на місто відправника.

– senderDepartmentRef: атрибут «senderDepartmentRef» має тип String і використовується для зберігання посилання на відділ відправника.

– deliveryServiceId: Атрибут «deliveryServiceId» має тип String і використовується для зберігання ідентифікатора служби доставки, пов'язаного з кабінетом доставки.

– title: Атрибут «title» має тип String і використовується для зберігання назви або опису кабінету доставки. Він має значення за замовчуванням порожній рядок ("").

Загалом, модель «CabinetDeliveryDTO» представляє сутність кабінету доставки в структурі даних програми з атрибутами для зберігання унікального ідентифікатора, ключа API, контактного посилання відправника, контактного імені відправника, номера телефону відправника, посилання відправника, імені відправника, посилання на місто відправника , довідка про відділ відправника, код служби доставки та назва кабінету доставки.

3.4 Розробка бази даних

У наступному фрагменті коду ми представляємо реалізацію бази даних за допомогою бібліотеки Room для програми Android. Ця база даних відіграє вирішальну роль в автоматизації бізнес-процесів торгової компанії. Використовуючи можливості Room, ми можемо ефективно керувати різними об'єктами та зберігати їх, забезпечуючи плавний доступ до важливої інформації та підвищуючи загальну продуктивність програми. Давайте зануримося в код і дослідимо, як ця база даних розроблена та використовується в програмі.

```
@Database(entities = [
    CabinetDeliveryDTO::class, LogActionsDTO::class, NPErrorDTO::class,
```

```
ScannerDTO::class, ScalesDTO::class, PrinterDTO::class,
ServicesDeliveryDTO::class,
    TemplatesDeliveryDTO::class, UnsynchronizedOrdersDTO::class,
UnsynchronizedOrdersItemDTO::class, UserDTO::class], version = 2,
    autoMigrations = [AutoMigration (from = 1, to = 2)])
public abstract class DataBase : RoomDatabase() {
    abstract fun cabinetDeliveryDAO(): CabinetDeliveryDAO

    abstract fun logActionsDAO(): LogActionsDAO

    abstract fun npErrorDAO(): NPErrorDAO

    abstract fun printerDAO(): PrinterDAO

    abstract fun scalesDAO(): ScalesDAO

    abstract fun scannerDAO(): ScannerDAO

    abstract fun servicesDeliveryDAO(): ServicesDeliveryDAO

    abstract fun templatesDeliveryDAO(): TemplatesDeliveryDAO

    abstract fun unsynchronizedOrdersDAO(): UnsynchronizedOrdersDAO

    abstract fun unsynchronizedOrdersItemDAO(): UnsynchronizedOrdersItemDAO

    abstract fun userDAO(): UserDAO
}
```

Наданий фрагмент коду представляє конфігурацію та реалізацію бази даних за допомогою бібліотеки Room в Android. Ось опис коду:

@Database: Ця анотація використовується для визначення класу бази даних. Він приймає два параметри:

entities: масив класів сутностей, які представляють таблиці в базі даних. У цьому випадку він включає такі класи , як CabinetDeliveryDTO, NPErrorDTO , та інші.

version: ціле число, що представляє версію бази даних. Це значення використовується для обробки міграцій бази даних у разі зміни схеми.

RoomDatabase: це абстрактний клас, який надається бібліотекою Room і служить основною точкою доступу до основної бази даних SQLite. Він надає методи для створення екземплярів бази даних і керування ними.

autoMigrations: Ця властивість використовується для визначення автоматичних міграцій між різними версіями бази даних. У наданому коді вказано один AutoMigration, який вказує, що коли версія бази даних збільшується з 1 до 2, має відбуватися автоматична міграція.

abstract fun: це абстрактні функції, які визначають об’єкт доступу до даних (DAO) для кожного класу сутності. DAO надає методи для виконання операцій бази даних, таких як вставка, оновлення, видалення та запит.

Функції DAO відповідають різним класам сутностей, згаданим в @Databaseанотації. Наприклад, cabinetDeliveryDAO()повертає екземпляр CabinetDeliveryDAO, logActionsDAO()повертає екземпляр LogActionsDAOтощо.

Розширюючи RoomDatabaseклас і визначаючи функції DAO, цей код налаштовує конфігурацію бази даних і забезпечує доступ до необхідних DAO для взаємодії з базою даних. Він діє як центральний центр для керування операціями бази даних у програмі.

Наступний фрагмент коду представляє інтерфейс DAO для сутності CabinetDeliveryDTO в базі даних програми. Цей інтерфейс DAO надає методи взаємодії з таблицею cabinet_delivery і виконання різних операцій з базою даних. Ці операції включають отримання кабінетів на основі певних критеріїв, вставлення нових кабінетів і доступ до пов’язаної інформації, такої як ключ API NP. Впроваджуючи цей інтерфейс DAO, програма може легко керувати та маніпулювати даними, пов’язаними з доставкою в кабінет. Усі інші DAO реалізовані аналогічно тому розберемо тільки один приклад.

```
@Dao
interface CabinetDeliveryDAO {
    @Query("SELECT * FROM cabinet_delivery WHERE delivery_services_id IN (:guid)
")
    fun getCabinets(guid: String): List<CabinetDeliveryDTO>?

    @Query("SELECT * FROM cabinet_delivery  WHERE guid IN (:guid)")
    fun getCabinet(guid: String): CabinetDeliveryDTO?

    @Query("SELECT * FROM cabinet_delivery")
    fun getDefaultCabinet(): CabinetDeliveryDTO?

    @Query("SELECT api_key FROM cabinet_delivery WHERE guid IN (:guid)")
    fun getNPApiKeyByGuid(guid: String): String?

    @Query("SELECT * FROM cabinet_delivery WHERE api_key IN (:apiKey)")
    fun getCabinetsByNPApi(apiKey: String): CabinetDeliveryDTO?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insert(vararg cabinets: CabinetDeliveryDTO)
}
```

Наданий фрагмент коду представляє інтерфейс об’єкта доступу до даних (DAO) для сутності CabinetDeliveryDTO в базі даних. Цей інтерфейс визначає різні методи, які дозволяють взаємодіяти з таблицею cabinet_delivery.

– Ось розбивка методів, визначених в інтерфейсі CabinetDeliveryDAO:

– getCabinets(guid: String): отримує список об’єктів CabinetDeliveryDTO на основі наданого GUID.

– getCabinet(guid: String): отримує один об’єкт CabinetDeliveryDTO на основі наданого GUID.

– getDefaultCabinet(): отримує стандартний об’єкт CabinetDeliveryDTO.

– getNPApiKeyByGuid(guid: String): отримує ключ API NP, пов’язаний із наданим GUID.

– getCabinetsByNPApi(apiKey: String): отримує об’єкт CabinetDeliveryDTO на основі наданого ключа API NP.

– insert(vararg cabinets: CabinetDeliveryDTO): вставляє один або кілька об’єктів CabinetDeliveryDTO в таблицю cabinet_delivery. У разі конфлікту існуючі дані будуть замінені.

Використовуючи цей інтерфейс DAO, ви можете виконувати різноманітні операції з базою даних, пов’язані з сутністю CabinetDeliveryDTO, як-от отримання шаф на основі певних критеріїв, вставлення нових шаф і доступ до пов’язаної інформації, як-от ключ API NP.

3.5 Вимоги до технічних та програмних засобів

Для забезпечення безперебійної та безперебійної роботи Android-застосунку, призначеного для автоматизації бізнес-процесів торгівельного підприємства, необхідно створити необхідну апаратно-програмну інфраструктуру. Це включає надання необхідних технічних і програмних інструментів, які забезпечують оптимальну роботу програми. На застосунок до основних функцій програми, можуть існувати особливі вимоги до апаратних пристроїв і відповідного

програмного забезпечення/драйверів для підтримки додаткових функцій, таких як ваги, сканер і інтеграція принтера. Ці додаткові вимоги відіграють вирішальну роль у покращенні можливостей програми та забезпеченні ефективних бізнес-операцій. Далі ми окреслимо конкретні технічні та програмні вимоги.

Вимоги до обладнання:

– Пристрій Android: застосунок має бути встановлено та запущено на пристроях Android під керуванням ОС Android 5.0 або новішої версії.

– Процесор: пристрій має мати мінімальну швидкість процесора 1.4 ГГц або вище.

– Пам'ять (RAM): пристрій повинен мати мінімум 2 ГБ оперативної пам'яті.

– Місце для зберігання: пристрій має мати достатньо місця для зберігання програми та її даних мінімальним значенням є 4 ГБ вільного місця.

– Вимоги до програмного забезпечення:

– Підключення до Інтернету: пристрій повинен мати доступ до стабільного підключення до Інтернету для зв’язку з сервером серверу та інтеграції з API «Нова Пошта».

– Дозволи: програма повинна мати необхідні дозволи, надані користувачем для доступу до таких функцій пристрою, як камера, мережа, сховище тощо.

Окрім вищезазначених апаратних і технічних вимог, для програми можуть знадобитися наступні пристрої та відповідне програмне забезпечення/драйвери:

Ваги:

– Апаратне забезпечення: програмі може знадобитися взаємодія із сумісними вагами для зважування та вимірювання. Ваги повинні підтримувати необхідні протоколи зв’язку (наприклад, USB, Bluetooth або Wi-Fi) для з’єднання з пристроєм Android.

– Програмне забезпечення: програма повинна мати доступ до необхідного програмного забезпечення або драйверів, які забезпечують зв’язок і обмін даними між вагами та пристроєм Android. Це програмне забезпечення або драйвери повинні бути сумісні з вагами, що використовуються.

Сканер:

– Апаратне забезпечення: програмі може знадобитися сканер штрих-кодів або QR-кодів для сканування штрих-кодів продуктів або QR-кодів. Сканер має бути сумісний із пристроєм Android і підтримувати необхідні можливості сканування.

– Програмне забезпечення: програма повинна мати доступ до необхідного програмного забезпечення або драйверів, які полегшують зв’язок і отримання даних із пристрою сканера. Це програмне забезпечення або драйвери мають бути сумісні зі сканером, який використовується.

Принтер:

– Апаратне забезпечення: програмі може знадобитися друк документів, квитанцій. Має бути доступний сумісний принтер, який підтримує необхідні можливості друку (наприклад, термодрук, підключення Bluetooth/Wi-Fi тощо).

– Програмне забезпечення: програма повинна мати доступ до необхідного програмного забезпечення або драйверів, які забезпечують зв’язок і можливості друку з вибраним принтером.

3.6 Тестування програмного забезпечення

При тестуванні розробленого програмного забезпечення, важливо вибрати відповідні методи та інструменти тестування, які відповідають характеристикам і вимогам програми. Вибір методів тестування повинен забезпечувати повне охоплення функціональних можливостей програми, виявлення потенційних дефектів або проблем і підтвердження того, що програмне забезпечення відповідає визначеним вимогам. Розглянемо методи та інструменти тестування, які добре підходять для розробленого програмного забезпечення:

Модульне тестування: модульне тестування зосереджується на тестуванні окремих одиниць або компонентів програмного забезпечення окремо. Він ідеально підходить для тестування невеликих незалежних одиниць коду, таких як методи чи функції. Для застосунків Android такі фреймворки, як JUnit або KotlinTest, можна використовувати для написання модульних тестів. Модульне тестування допомагає

перевірити правильність окремих блоків і виявити будь-які дефекти чи помилки на ранній стадії.

Тестування системи: Тестування системи оцінює поведінку всієї системи або програми в реальному чи змодельованому середовищі. Він перевіряє, чи програмне забезпечення відповідає визначеним вимогам і функціонує належним чином. Цей тип тестування може включати такі сценарії, як потоки користувачів, крайові випадки та тестування продуктивності. Такі інструменти, як Appium або UI Automator, можна використовувати для тестування системи на Android. Тестування системи забезпечує цілісне уявлення про функціональність програми та допомагає переконатися в її надійності та зручності використання.

Вибір цих методів та інструментів тестування виправданий їх здатністю охоплювати різні аспекти програми, починаючи від окремих одиниць коду до загальної поведінки системи. Використовуючи комбінацію цих методів тестування, можна досягти повного охоплення тестуванням, виявити дефекти чи проблеми та підтвердити, що розроблене програмне забезпечення відповідає заданим вимогам.

Розпочнемо з тестування методу sendEnRef. Мета цього тестування полягає в тому, щоб переконатися, що метод працює належним чином і обробляє різні сценарії належним чином. Для написання тестів і перевірки функціональності методу використовується KotlinTest.

Метод sendEnRef відповідає за надсилання ТТН замовлення до віддаленого API та обробку відповіді. Він приймає DocOrderItemоб’єкт разом з іншими параметрами, готує необхідні дані та здійснює виклик API. Залежно від відповіді він запускає різні події, використовуючи EventBusдля передачі результату.

Розглянемо написаний тест:

```
class SendEnRefTest : StringSpec({
    val orderItem = mockk<DocOrderItem>()
    val api = mockk<ExampleApp.API>()
    val call = mockk<Call<JsonObject>>()
    beforeTest {
        clearMocks(orderItem, api, call)
    }

    "Should post BustMessageEvent with '4' on successful response" {
        val response = mockk<Response<JsonObject>>()
        every { response.isSuccessful } returns true
        every { response.body() } returns mockk {
            every { getAsJsonObject().get("success").getAsBoolean() } returns true
```

```
        }
        every { call.enqueue(any<Callback<JsonObject>>()) } answers {
            val callback = arg<Callback<JsonObject>>(0)
            callback.onResponse(call, response)
        }
        val eventBus = mockk<EventBus>()
        every { EventBus.getDefault() } returns eventBus

        val target = MyClass()
        target.sendEnRef(orderItem, 1, 2)
        verify(exactly = 1) { eventBus.post(any<BustMessageEvent>()) }
        val postedEvent = slot<BustMessageEvent>()
        verify { eventBus.post(capture(postedEvent)) }
        postedEvent.captured.event shouldBe BustMessageEvent.BUST_EVENT_SEND_TTN
        postedEvent.captured.message shouldBe "4"
    }

    "Should post BustMessageEvent with '2' on unsuccessful response" {
        val response = mockk<Response<JsonObject>>()
        every { response.isSuccessful } returns true
        every { response.body() } returns mockk {
            every { getAsJsonObject().get("success").getAsBoolean() } returns
false
        }
        every { call.enqueue(any<Callback<JsonObject>>()) } answers {
            val callback = arg<Callback<JsonObject>>(0)
            callback.onResponse(call, response)
        }
        val eventBus = mockk<EventBus>()
        every { EventBus.getDefault() } returns eventBus
        val target = MyClass()
        target.sendEnRef(orderItem, 1, 2)
        verify(exactly = 1) { eventBus.post(any<BustMessageEvent>()) }
        val postedEvent = slot<BustMessageEvent>()
        verify { eventBus.post(capture(postedEvent)) }
        postedEvent.captured.event shouldBe BustMessageEvent.BUST_EVENT_SEND_TTN
        postedEvent.captured.message shouldBe "2"
    }
})
```

У цьому тесті ми імітуємо необхідні залежності (такі як DocOrderItem, ExampleApp.API, і Retrofit Calli Responseкласи) за допомогою mockkбібліотеки. Потім ми визначаємо два тести: один для успішної відповіді та інший для невдалої відповіді.

У кожному тестовому випадку ми встановлюємо необхідну макетну поведінку для залежностей і виконуємо тестований метод. Потім ми перевіряємо очікувані взаємодії, такі як публікація в BustMessageEventшині подій.

На основі результатів тестування можна зробити висновок, що sendEnRefметод у наданому коді виконує очікувану функціональність. Тестові приклади охоплюють як успішні, так і невдалі відповіді на виклик API та

перевіряють, що правильна інформація BustMessageEvent розміщена в шині подій на основі відповіді.

Далі розглянемо тетстування DataBase класу, який відповідає за надання доступу до основної бази даних у нашій програмі. Клас DataBase– це абстрактний клас, який розширює RoomDatabase, частину бібліотеки збереження кімнати в Android. Він служить точкою входу для доступу до інтерфейсів DAO (Data Access Object), пов'язаних з різними об'єктами бази даних.

Тестування DataBaseкласу має вирішальне значення, щоб переконатися, що операції з базою даних і функції доступу до даних працюють правильно. Проводячи комплексні тести, ми можемо переконатися, що DataBaseклас правильно налаштований і надає очікувані екземпляри DAO для взаємодії з об'єктами даних.

Розглянемо написаний тест:

```
class DataBaseTest : StringSpec() {
    init {
        val mockCabinetDeliveryDAO = mockk<CabinetDeliveryDAO>()
        val mockLogActionsDAO = mockk<gActionsDAO>()
        val mockNPErrorDAO = mockk<NPErrorDAO>()
        val mockPrinterDAO = mockk<PrinterDAO>()
        val mockScalesDAO = mockk<ScalesDAO>()
        val mockScannerDAO = mockk<ScannerDAO>()
        val mockServicesDeliveryDAO = mockk<ServicesDeliveryDAO>()
        val mockTemplatesDeliveryDAO = mockk<TemplatesDeliveryDAO>()
        val mockUnsynchronizedOrdersDAO = mockk<UnsynchronizedOrdersDAO>()
        val mockUnsynchronizedOrdersItemDAO = mockk<UnsynchronizedOrdersItemDAO>()
        val mockUserDAO = mockk<UserDAO>()

        val testDataBase = object : DataBase() {
            override fun cabinetDeliveryDAO(): CabinetDeliveryDAO =
mockCabinetDeliveryDAO
            override fun logActionsDAO(): LogActionsDAO = mockLogActionsDAO
            override fun npErrorDAO(): NPErrorDAO = mockNPErrorDAO
            override fun printerDAO(): PrinterDAO = mockPrinterDAO
            override fun scalesDAO(): ScalesDAO = mockScalesDAO
            override fun scannerDAO(): ScannerDAO = mockScannerDAO
            override fun servicesDeliveryDAO(): ServicesDeliveryDAO =
mockServicesDeliveryDAO
            override fun templatesDeliveryDAO(): TemplatesDeliveryDAO =
mockTemplatesDeliveryDAO
            override fun unsynchronizedOrdersDAO(): UnsynchronizedOrdersDAO =
mockUnsynchronizedOrdersDAO
            override fun unsynchronizedOrdersItemDAO():
UnsynchronizedOrdersItemDAO = mockUnsynchronizedOrdersItemDAO
            override fun userDAO(): UserDAO = mockUserDAO
        }

        "cabinetDeliveryDAO should return the correct DAO instance" {
            testDataBase.cabinetDeliveryDAO() shouldBe mockCabinetDeliveryDAO
        }
        "logActionsDAO should return the correct DAO instance" {
            testDataBase.logActionsDAO() shouldBe mockLogActionsDAO
```

```
        }
        "npErrorDAO should return the correct DAO instance" {
            testDataBase.npErrorDAO() shouldBe mockNPErrorDAO
        }
        "printerDAO should return the correct DAO instance" {
            testDataBase.printerDAO() shouldBe mockPrinterDAO
        }
        "scalesDAO should return the correct DAO instance" {
            testDataBase.scalesDAO() shouldBe mockScalesDAO
        }
        "scannerDAO should return the correct DAO instance" {
            testDataBase.scannerDAO() shouldBe mockScannerDAO
        }
        "servicesDeliveryDAO should return the correct DAO instance" {
            testDataBase.servicesDeliveryDAO() shouldBe mockServicesDeliveryDAO
        }
        "templatesDeliveryDAO should return the correct DAO instance" {
            testDataBase.templatesDeliveryDAO() shouldBe mockTemplatesDeliveryDAO
        }
        "unsynchronizedOrdersDAO should return the correct DAO instance" {
            testDataBase.unsynchronizedOrdersDAO() shouldBe
mockUnsynchronizedOrdersDAO
        }
        "unsynchronizedOrdersItemDAO should return the correct DAO instance" {
            testDataBase.unsynchronizedOrdersItemDAO() shouldBe
mockUnsynchronizedOrdersItemDAO
        }
        "userDAO should return the correct DAO instance" {
            testDataBase.userDAO() shouldBe mockUserDAO
        }
    }
}
```

Після виконання тесту для DataBase класу можна зробити висновок, що клас функціонує правильно. Тестові приклади перевіряють, що кожен метод DAO повертає очікуваний екземпляр DAO, який вказує на те, що клас DataBase правильно реалізовано та надає відповідні екземпляри DAO для доступу до даних.

Успішно пройшовши тестові випадки, він демонструє, що DataBase клас здатний надавати необхідні об’єкти DAO для доступу до відповідних об’єктів даних у базі даних. Це гарантує, що програма може взаємодіяти з базою даних і виконувати необхідні операції, такі як отримання, оновлення та вставка даних через інтерфейси DAO.

При проведенні серії тестів на різних модулях програми, в тому числі DataBasei, SendEnRefTest, комплексний аналіз результатів тестування дозволяє виявити наступне:

Тести, проведені на об’єктах доступу до даних (DAO), пов’язаних з CabinetDeliveryDTO, LogActionsDTO, NPErrorDTO, PrinterDTO, ScalesDTO, ScannerDTO, ServicesDeliveryDTO, TemplatesDeliveryDTO,

UnsynchronizedOrdersDTO, UnsynchronizedOrdersItemDTO, і UserDTO демонструють їхню успішну функціональність і дотримання визначених специфікацій.

Інтеграційні тести для DataBase модуля перевіряють належну взаємодію між базою даних і DAO, забезпечуючи постійність і отримання даних, як очікувалося.

Тести охоплюють різні операції з базою даних, такі як вставка даних, пошук і виконання запитів, забезпечуючи впевненість у надійності та правильності рівня бази даних.

Тест для sendEnRef методу класу ExampleApp перевіряє функціональні можливості, пов’язані з надсиланням посилання на сутність для замовлення.

Тестові сценарії охоплюють як успішні, так і невдалі випадки, гарантуючи, що метод належним чином обробляє відповіді API та публікує відповідні події.

Тестуючи цю специфічну функціональність, ми отримуємо впевненість, що sendEnRef метод поводиться правильно, гарантуючи правильну передачу посилань на сутності та відповідну публікацію подій.

Загальні результати тестування свідчать про те, що модулі програми успішно реалізовано та надійно виконують поставлені завдання. Прохідні тести демонструють правильне функціонування окремих компонентів та їх хорошу інтеграцію в програму. Цей комплексний підхід до тестування сприяє загальній якості, стабільності та продуктивності програмного забезпечення.

Підсумовуючи, ретельне тестування, проведене на різних модулях, гарантує надійність і цілісність програми. Перевіряючи очікувану поведінку та відповідність вимогам, тести забезпечують високий рівень впевненості у функціональності та надійності програмного забезпечення. Успішні результати тестування свідчать про те, що програма добре оснащена для автоматизації бізнес-процесів і забезпечення бездоганної взаємодії з користувачем, зрештою вирішуючи виявлені проблеми та покращуючи ефективність, економічну ефективність і задоволеність клієнтів.

ВИСНОВКИ

Під час написання кваліфікаційної роботи було проведено глибокий аналіз предметної області, зосереджено увагу на проблемах автоматизації бізнес-процесів торгових підприємств, зокрема на проблемах підтвердження замовлень та формування транспортних накладних. Було проведено дослідження наявних рішень на ринку, спрямоване на автоматизацію цих процесів. В ході дослідження було визначено їх основні переваги та недоліки, і доведено, що розробка нового програмного продукту для автоматизації бізнесу є доцільною та здатною вирішити існуючі проблеми у торговельних компаніях. Вимоги до програмного продукту були детально описані в технічному завданні, зосереджуючись на сильних сторонах конкурентних рішень.

Наступним етапом у процесі створення програмного продукту було проведено аналіз та порівняння переваг і недоліків різних архітектурних підходів для мобільних застосунків. За результатами аналізу було встановлено, що найбільш підходящим та зручним є архітектурний шаблон під назвою MVVM. Декомпозиція даних та модулів допомогла зрозуміти, як конструювати застосунок та реалізувати логіку.

У процесі детального аналізу була проведена оцінка технологій і інструментів, які планується використовувати при розробці програмного продукту. Були виокремлені та описані основні модулі застосунку, а також встановлено, як вони будуть взаємодіяти між собою.

У результаті був розроблений мобільний застосунок для автоматизації бізнес-процесів, зокрема для підтвердження замовлень та створення транспортних накладних. Реалізація цього застосунку була успішно завершена, а використання сучасних технологій та інструментів дало змогу отримати цінні знання та навички, які будуть надзвичайно корисними в моїй подальшій професійній діяльності.

Загальною метою цієї кваліфікаційної роботи було не лише досягнення поставлених завдань, але й розширення розуміння процесів автоматизації бізнесу та здатність застосувати сучасні технології для їх реалізації. Виконання цієї роботи

дало можливість зробити практичний внесок у сферу автоматизації бізнесу і отримати цінний досвід, який буде використовуватися у подальших професійних зусиллях.

ПЕРЕЛІК ПОСИЛАНЬ

1. Сімейство блогів «Tumblr» [Електронний ресурс] – Режим доступу: https://www.tumblr.com/
2. Онлайн портфоліо «Behance» [Електронний ресурс] – Режим доступу: https://www.behance.net
3. Спільнота дизайнерів «Dribbble» [Електронний ресурс] – Режим доступу: https://dribbble.com
4. The Java EE 5 Tutorial [Електронний ресурс] – Режим доступу: https://docs.oracle.com/cd/E19575-01/819-3669/6n5sg7arb/index.html
5. Web-разработка на Python глазами PHP-программиста [Електронний ресурс] – Режим доступу: https://habrahabr.ru/post/243961/
6. PHP – найбільш популярна мова для веб-програмування [Електронний ресурс] – Режим доступу: http://asaweb.com.ua/ua/php5/
7. SQLite vs MySQL vs PostgreSQL: порівняння систем управління базами даних [Електронний ресурс] – Режим доступу: http://devacademy.ua/posts/sqlite-vs-mysql-vs-postgresql/
8. MySQL 5.0. Бібліотека програміста – СПб.: Питер, 2010. – 253 с.: ил.
9. Дейт К. Дж. Введення в систему баз даних. – СПб.: Вильямс, 2005. – 1316 с.: ил.
10. Нельсен Якоб. Веб дизайн. – СПб.: Питер, 2013. – 504 с.: ил.
11. «MVC, MVP and MVVM Design Pattern» [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad
12. «UnitOfWork And Repository Pattern» [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@utterbbq/c-unitofwork-and-repository-pattern305cd8ecfa7a
13. «What is ArcGIS, and where is it available at IU?» [Електронний ресурс] – Режим доступу до ресурсу:

14. OWASP Python Security wiki [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ebranca/owasp-pysec/wiki.
15. Compilers, principles, techniques, and tools / Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, 2007. 2nd ed.
16. P.D. Terry. Compilers and Compiler Generators – 1996. С. 23.
17. Static Code Analysis [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/www-community/controls/Static_Code_Analysis.
18. Ball T. The Concept of Dynamic Analysis / Thomas Ball. – С. 1–3.
19. Heribertus Y. Web Application Vulnerability Detection Using Taint Analysis and Blackbox Testing / Yulianton Heribertus. – 2020. – С. 3–4.
20. Freitas A. A survey of evolutionary algorithms for data mining and knowledge discovery. - 2001.
21. Freund Y. Boosting a weak learning algorithm by majority // COLT: Proceedings of the Workshop on Computational Learning Theory. - Morgan Kaufmann Publishers, 1990.
22. Freund Y. An adaptive version of the boost by majority algorithm // COLT: Proceedings of the Workshop on Computational Learning Theory. - Morgan Kaufmann Publishers, 1999.
23. Freund Y., Schapire RE A decision-theoretic generalization of on-line learning and an application to boosting // European Conference on Computational Learning Theory. - 1995. - Pp. 23-37.
24. Freund Y., Schapire RE Experiments with a new boosting algorithm // International Conference on Machine Learning. - 1996. - Pp. 148-156.
25. Angeline PJ Adaptive and self-adaptive evolutionary computations // Computational Intelligence: A Dynamic Systems Perspective / Ed. by M. Palaniswami, Y. Attikiouzel. - IEEE Press, 1995. - Pp. 152-163.
26. Angeline PJ, Pollack JB Competitive environments evolve better solutions for complex tasks // Proceedings of the 5th International Conference on Genetic Algorithms (GA-93). - 1993. - Pp. 264-270.
27. Back T. Self-adaptation in genetic algorithms.

28. Back T. Optimization by means of genetic algorithms // 36th International Scientific Colloquium / Ed. by E. Kohler. - Technical University of Ilmenau: 1991. - Pp. 163-169.
29. Back T., Hoffmeister F. Global optimization by means of evolutionary algorithms.
30. Dymitr Ruta BG Analysis of the correlation between majority voting error and the diversity.
31. Ficici SG, Melnik O., Pollack JB A game-theoretic investigation of selection methods used in evolutionary algorithms // Proceedings of the 2000 Congress on Evolutionary Computation CEC00. - La Jolla Marriott Hotel La Jolla, California, USA: IEEE Press, 6-9 2000. - P. 880.
32. Ficici SG, Pollack JB A game-theoretic approach to the simple coevolutionary algorithm // Parallel Problem Solving from Nature - PPSN VI 6th International Conference / Ed. by H.-PS Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph.
33. Dawson M. Programmable на Python (програмування Python для Absolute Beginner) /d. «Peter», серія Bestsells O'Reilly, 2016,- 416с.
34. Dawson M. Programmable на Python. - СПб.: Питер, 2014. - 416 р.
35. Основи програмування на мові Python належать Zlatopol D.M. Basics. - М.: ДМК Пресс, 2017. - 284 р.
36. Lutz M. Ми вивчаємо Python, 4-е видання, - Per. Від Енг. - СПб.: СимвоПлюс, 2011. - 1280 с., іл.
37. Lutz M. Programming на Python, том I, 4-е видання. С английскою. - СПб.: Символ-Плюс, 2011. - 992 р.
38. Lutz M. Programming on Python, Volume II, 4-е видання. С английскою. - СПб.: Символ-Плюс, 2011. - 992 р.
39. Маклаков С.В. Моделювання процесів – Київ: Діалог, 2012. – 223 с.
40. Титаренко Г.А., Автоматизовані інформаційні технології – Київ: Символ-Плюс, 2010. – 159 с.
41. Арсаж Ж. Програмування задач на Python – Київ: Пересвіт, 2014. – 521 с.

ДОДАТОК А
(обов'язковий)

Клас бази даних:

```
@Database(entities = [
    CabinetDeliveryDTO::class, LogActionsDTO::class, NPErrorDTO::class,
ScannerDTO::class, ScalesDTO::class, PrinterDTO::class,
ServicesDeliveryDTO::class,
    TemplatesDeliveryDTO::class, UnsynchronizedOrdersDTO::class,
UnsynchronizedOrdersItemDTO::class, UserDTO::class], version = 2,
    autoMigrations = [AutoMigration (from = 1, to = 2)])
public abstract class DataBase : RoomDatabase() {
    abstract fun cabinetDeliveryDAO(): CabinetDeliveryDAO

    abstract fun logActionsDAO(): LogActionsDAO

    abstract fun npErrorDAO(): NPErrorDAO

    abstract fun printerDAO(): PrinterDAO

    abstract fun scalesDAO(): ScalesDAO

    abstract fun scannerDAO(): ScannerDAO

    abstract fun servicesDeliveryDAO(): ServicesDeliveryDAO

    abstract fun templatesDeliveryDAO(): TemplatesDeliveryDAO

    abstract fun unsynchronizedOrdersDAO(): UnsynchronizedOrdersDAO

    abstract fun unsynchronizedOrdersItemDAO(): UnsynchronizedOrdersItemDAO

    abstract fun userDAO(): UserDAO
```

Клас DTO користувача:

```
@Entity(tableName = "users")
data class UserDTO(
        @PrimaryKey
        @ColumnInfo(name = "guid") var guid: String,
        @ColumnInfo(name = "name") var name: String,
        @ColumnInfo(name = "login") var login: String,
        @ColumnInfo(name = "password") var password: String,
        @ColumnInfo(name = "token") var token: String
)
```

Клас DTO кабінетів доставки:

```
@Entity(tableName = "cabinet_delivery")
data class CabinetDeliveryDTO(
        @PrimaryKey @ColumnInfo(name = "guid") var guid: String,
        @ColumnInfo(name = "api_key") var apiKey: String,
        @ColumnInfo(name = "contact_sender_ref") var contactSenderRef: String,
```

```
        @ColumnInfo(name = "contact_sender_name") var contactSenderName: String,
        @ColumnInfo(name = "sender_phone") var senderPhone: String,
        @ColumnInfo(name = "sender_ref") var senderRef: String,
        @ColumnInfo(name = "sender_name") var senderName: String,
        @ColumnInfo(name = "sender_city_ref") var senderCityRef: String,
        @ColumnInfo(name = "sender_department_ref") var senderDepartmentRef:
String,
        @ColumnInfo(name = "delivery_services_id") var deliveryServiceId: String,
        @ColumnInfo(name = "title", defaultValue = "") var title: String
)
```

Клас DTO принтера:

```
@Entity(tableName = "printers")
data class PrinterDTO(
        @PrimaryKey
        @ColumnInfo(name = "guid") var guid: String,
        @ColumnInfo(name = "type_connection") var typeConnection: Int,
        @ColumnInfo(name = "name") var name: String,
        @ColumnInfo(name = "address") var address: String,
        @ColumnInfo(name = "active") var active: Int,
        @ColumnInfo(name = "is_use_scale") var isUseScale: Boolean,
        @ColumnInfo(name = "type_paper") var typePaper: Int,
        @ColumnInfo(name = "type_printer") var typePrinter: Int,
        @ColumnInfo(name = "port") var port: String
)
```

Код activity списку замовлень:

```
class OrderActivity : LocalizationActivity(), OrderListener,
    NavigationView.OnNavigationItemSelectedListener {
    private lateinit var binding: ActivityOrdersBinding
    private val viewModel by viewModels<ViewModelOrder>()
    private lateinit var viewPagerAdapter: ViewPagerAdapter
    var searchView: SearchView? = null
    private lateinit var pDialog: ProgressDialog
    private var progressDialog: ProgressDialogFragment? = null
    private var firstStart = true

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = DataBindingUtil.setContentView(this, R.layout.activity_orders)
        setSupportActionBar(binding.ordersListToolbar)
        ExampleApp.getInstance().getDeliveryServices(this)
        ExampleApp.getInstance().getDeliveryTemplates(null)
        ExampleApp.getDataBaseManager().logActionsDAO.deleteSeniorFor()

        val mLanguageChangedReceiver = object : BroadcastReceiver() {
            override fun onReceive(context: Context, intent: Intent) {
                this@OrderActivity.recreate()
            }
        }
        registerReceiver(
            mLanguageChangedReceiver,
            IntentFilter(AppConfig.LANGUAGE_HAS_BEEN_CHANGED)
        )
```

```
        viewPagerAdapter = ViewPagerAdapter(supportFragmentManager, lifecycle)

        viewModel.listener = this

        viewPagerAdapter.addFragment(OrderFragment.newInstance(false))
        viewPagerAdapter.addFragment(OrderFragment.newInstance(true))

        binding.viewPager.orientation = ViewPager2.ORIENTATION_HORIZONTAL
        binding.viewPager.adapter = viewPagerAdapter
        binding.viewPager.reduceDragSensitivity(4)

        TabLayoutMediator(binding.tabLayout, binding.viewPager) { tab, position ->
          when (position) {
                0 -> getString(R.string.confirm_order)
                1 -> getString(R.string.archive_order)
            }
        }.attach()

        val toggle = ActionBarDrawerToggle(
            this,
            binding.drawerLayout,
            binding.ordersListToolbar,
            R.string.navigation_drawer_open,
            R.string.navigation_drawer_close
        )
        binding.drawerLayout.addDrawerListener(toggle)
        toggle.syncState()
        val navigationView = binding.navView
        navigationView.setNavigationItemSelectedListener(this)
        navigationView.setCheckedItem(R.id.nav_orders)
        val menu = navigationView.menu
        navigationView.headerCount

        val userDTO = viewModel.user
        val userName =
navigationView.getHeaderView(0).findViewById<TextView>(R.id.userName)
        if (userDTO != null && userName != null) {
            val name = userDTO.name
            userName.text = name
        }

        try {
            val pInfo = packageManager.getPackageInfo(packageName, 0)
            val appVersion = pInfo.versionName
            menu.findItem(R.id.nav_version).title = appVersion
        } catch (e: Exception) {
            e.printStackTrace()
        }
        pDialog = ProgressDialog(this)
        pDialog.setCancelable(false)
        showDialog(getString(R.string.Receiving_orders))
        viewModel.refresh()
    }

    private fun ViewPager2.reduceDragSensitivity(value: Int) {
        val recyclerViewField =
ViewPager2::class.java.getDeclaredField("mRecyclerView")
        recyclerViewField.isAccessible = true
        val recyclerView = recyclerViewField.get(this) as RecyclerView

        val touchSlopField =
RecyclerView::class.java.getDeclaredField("mTouchSlop")
```

```
        touchSlopField.isAccessible = true
        val touchSlop = touchSlopField.get(recyclerView) as Int
        touchSlopField.set(recyclerView, touchSlop * value)
    }

    override fun error(textId: Int) {
        Toast.makeText(this, getString(textId), Toast.LENGTH_LONG).show()
    }

    override fun error(textId: String) {
        Toast.makeText(this, textId, Toast.LENGTH_LONG).show()
    }

    override fun update(isArchive: Boolean, isUpdateAll: Boolean) {
        val fragment = if (isArchive) {
            binding.tabLayout.getTabAt(1)?.text =
                getString(R.string.archive_order) + " (" +
viewModel.getTotalCount(isArchive) + ")"
            viewPagerAdapter.getFragments().find { it.archive && it.isInit }
        } else {
            binding.tabLayout.getTabAt(0)?.text =
                getString(R.string.confirm_order) + " (" +
viewModel.getTotalCount(isArchive) + ")"
            viewPagerAdapter.getFragments().find { !it.archive && it.isInit }
        }
        if (isUpdateAll) {
            fragment?.updateAll()
        } else
            fragment?.update()

    }


    override fun hideProgress(isArchive: Boolean) {
        if (isArchive) {
            val fragment = getActiveFragment()
            if (viewPagerAdapter.getFragments().find { it.archive }?.archive ==
fragment.archive)
                fragment.hideProgress()
        } else {
            val fragment = getActiveFragment()
            if (viewPagerAdapter.getFragments().find { !it.archive }?.archive ==
fragment.archive)
                fragment.hideProgress()
        }
        if (pDialog.isShowing) pDialog.dismiss()
    }

    override fun logoutUser() {
        ExampleApp.getInstance().logoutUser(this)
    }

    override fun showProgressForPagination(isArchive: Boolean) {
        viewPagerAdapter.getFragments().find { it.activeFragment
}?.showProgressForPagination()
    }


    private val receiverSunmiScan: BroadcastReceiver = object :
BroadcastReceiver() {
        override fun onReceive(context: Context, intent: Intent) =
            selectionFound(intent.getStringExtra(AppConfig.DATA))
    }
```

```
    private val receiverZebraScan: BroadcastReceiver = object :
BroadcastReceiver() {
        override fun onReceive(context: Context, intent: Intent) =
            selectionFound(intent.getStringExtra(AppConfig.DATA))
    }

    fun selectionFound(code: String?) {
        if (code != null && code.isNotEmpty() &&
lifecycle.currentState.isAtLeast(Lifecycle.State.RESUMED)) {
            val barCode = code.replace("[^A-Za-z0-9]".toRegex(), "")
            searchView?.isIconified = false
            searchView?.setQuery(barCode, true)
        }
    }

    @SuppressLint("NonConstantResourceId")
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.menu_refresh -> {
                val fragment = getActiveFragment();
                fragment.showProgressForRefresh()
                viewModel.refresh()
            }
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.activity_orders_list_menu, menu)
        var searchText = ""

        val myActionMenuItem = menu.findItem(R.id.top_menu_search)
        searchView = myActionMenuItem.actionView as SearchView
        searchView?.inputType = InputType.TYPE_CLASS_NUMBER
        val searchImgId = resources.getIdentifier("android:id/search_button",
null, null)
        val v = searchView?.findViewById<View>(searchImgId) as ImageView
        v.setImageResource(R.drawable.ic_search)
        val handler = Handler()
        val runnable = Runnable {
            viewModel.refresh()
        }
        searchView?.setOnQueryTextListener(object : SearchView.OnQueryTextListener
{
            override fun onQueryTextSubmit(query: String): Boolean {
                searchView?.requestFocus()
                searchView?.clearFocus()
                return false
            }

            override fun onQueryTextChange(s: String): Boolean {
                viewModel.search = s
                handler.removeCallbacks(runnable)
                handler.postDelayed(runnable, 400)
                getActiveFragment().adapter.selectSearchText(s)
                return false
            }
        })
        searchView?.setOnCloseListener {
            viewModel.search = ""
            val fragment = getActiveFragment()
            viewModel.getData(fragment.archive)
```

```
            fragment.adapter.selectSearchText("")
            false
        }
        return true
    }

    private fun showDialog(message: String) {
        if (!pDialog.isShowing) {
            pDialog.setMessage(message)
            pDialog.show()
        }
    }

    fun getActiveFragment(): OrderFragment {
        return supportFragmentManager.fragments.find {
it.tag=="f"+binding.viewPager.currentItem } as OrderFragment
    }

    fun updateErrorsNP() {
        if (firstStart) {
            firstStart = false
            try {
                Thread.sleep(500)
            } catch (e: InterruptedException) {
                e.printStackTrace()
            }
            val dateUpdateErrors =
                NovaPoshtaManager.getManager().lastDateUpdateErros -
System.currentTimeMillis()
            val daysDiff = TimeUnit.MILLISECONDS.toDays(dateUpdateErrors)
            val cabinetDeliveryDTO =

ExampleApp.getDataBaseManager().cabinetDeliveryDAO.getDefaultCabinet()
            if (cabinetDeliveryDTO != null) {
                val api = cabinetDeliveryDTO.apiKey
                if ((daysDiff > 14 || daysDiff < 0) && api.isNotEmpty()) {
                    progressDialog = ProgressDialogFragment.newInstance(
                        1,
                        this.getString(R.string.Update_of_directories),
                        100
                    )

progressDialog?.show(supportFragmentManager.beginTransaction(), "d")
                    NovaPoshtaApi.getInstance().updateErrorsMessages()
                }
            }
        }
    }

    override fun onStart() {
        super.onStart()
        EventBus.getDefault().register(this)
        applicationContext.registerReceiver(
            receiverSunmiScan,
            IntentFilter(AppConfig.ACTION_DATA_SUNMI_CODE_RECEIVED)
        )
        applicationContext.registerReceiver(
            receiverZebraScan,
            IntentFilter(AppConfig.ACTION_DATA_ZEBRA_CODE_RECEIVED)
        )
    }

    override fun onStop() {
```

```
        EventBus.getDefault().unregister(this)
        applicationContext.unregisterReceiver(receiverSunmiScan)
        applicationContext.unregisterReceiver(receiverZebraScan)
        super.onStop()
    }

    @SuppressLint("NotifyDataSetChanged")
    @Subscribe(threadMode = ThreadMode.MAIN)
    fun onMessageEvent(event: BustMessageEvent) {
        when (event.typeEvent) {
            BustMessageEvent.BUST_EVENT_CLOSE_PROGRESS -> if
(progressDialog?.isVisible == true) progressDialog?.dismiss()
            BustMessageEvent.BUST_EVENT_SHOW_PROGRESS ->
progressDialog?.setProgress(
                event.intData,
                event.intData2
            )
            BustMessageEvent.BUST_EVENT_UPDATE_ERROR_NP -> updateErrorsNP()
            BustMessageEvent.BUST_EVENT_SEND_TTN_MAIN_ACTIVITY -> {
                val pos = event.arrData[1].toInt()
                when (event.arrData[0]) {
                    "1" -> getActiveFragment().updateStatusByPosition(2, pos)
                    "3" -> getActiveFragment().updateStatusByPosition(3, pos)
                }
            }
        }
    }

    override fun onNavigationItemSelected(item: MenuItem): Boolean {
        if (item.itemId == R.id.nav_exit) firstStart = true
        return ExampleApp.getInstance().menuClick(this, item,
binding.drawerLayout)
    }

    override fun onBackPressed() {
        if (binding.drawerLayout.isDrawerOpen(GravityCompat.START)) {
            binding.drawerLayout.closeDrawer(GravityCompat.START)
        } else {
            super.onBackPressed()
        }
    }
}
```

Код activity підтвердження замовлення:

```
public class ConfirmOrderActivity extends LocalizationActivity implements
updateOrderInterface {
    private static final int COUNT_PAGER = 2;
    private int confirm_Position_Count, position;
    private DocOrderItem orderData;
    private float textSizeSmall, textSizeMedium, textSizeLarge, textSizeXLarge;
    SharedPreferences preferences;
    Animation animAlpha;
    TabLayout tabLayout;
    ConfirmOrderFragment fragmentConfirmOrder;
    DeliveryOrderFragment fragmentDeliveryOrder;
    private static final String EXTRA_DOC_ORDER_ITEM = "docOrderItem";
    private static final String EXTRA_POSITION = "position";
```

```
    private static final String EXTRA_CONFIRM = "confirm";
    private ViewPager2 viewPager;
    private final List<HistoriItemInf> HistoryItemsList = new ArrayList<>();
    private HistoryAdapter historyItemsAdapter;
    DrawerLayout drawer;
    Button clear_history_list;
    Boolean confirm = false;

    public static Intent createIntent(Context context, DocOrderItem docOrderItem,
int position) {
        Intent intent = new Intent(context, ConfirmOrderActivity.class);
        intent.putExtra(EXTRA_DOC_ORDER_ITEM, docOrderItem);
        intent.putExtra(EXTRA_POSITION, position);
        return intent;
    }

    public static int getPosition(Intent intent) {
        return intent.getIntExtra(EXTRA_POSITION, -1);
    }

    public static DocOrderItem getOrder(Intent intent) {
        return (DocOrderItem) intent.getSerializableExtra(EXTRA_DOC_ORDER_ITEM);
    }

    @Override
    public void onBackPressed() {
        if (fragmentConfirmOrder.save) {
            AlertDialog.Builder builder = new AlertDialog.Builder(this);

builder.setTitle(getResources().getString(R.string.close_order_title)).setIcon(get
Resources().getDrawable(R.drawable.ic_close_order_dialog)).setMessage(this.getStri
ng(R.string.close_order_question)).setNegativeButton(getResources().getString(R.st
ring.app_No), (dialog, id) ->
dialog.cancel()).setPositiveButton(getResources().getString(R.string.app_Yes),
(dialog, id) -> {
                if (orderData.getConfirmed_status() == 1 && !confirm) {
                    cancel_confirm(orderData.getGuid());
                }
                Intent intent = new Intent();
                intent.putExtra(EXTRA_POSITION, position);
                setResult(RESULT_OK, intent);
                finish();
            });
            AlertDialog alert = builder.create();
            alert.show();
        } else {
            Intent intent = new Intent();
            intent.putExtra(EXTRA_POSITION, position);
            if (orderData.getConfirmed_status() == 2)
                intent.putExtra(EXTRA_DOC_ORDER_ITEM, orderData);
            setResult(RESULT_OK, intent);
            finish();
        }
    }

    @SuppressLint({"SimpleDateFormat", "NotifyDataSetChanged"})
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_confirm_order);
        Toolbar toolbar = findViewById(R.id.confirm_order_toolbar);
        toolbar.setLogo(R.mipmap.ic_launcher);
        preferences = getSharedPreferences(AppConfig.PREF_NAME,
```

```
Context.MODE_PRIVATE);
        switch (SettingsManager.getManager().getTextSize()) {
            case AppConfig.TEXT_SIZE_SMALL:
                textSizeSmall =
getResources().getDimension(R.dimen.item_small_text_size_small);
                textSizeMedium =
getResources().getDimension(R.dimen.item_medium_text_size_small);
                textSizeLarge =
getResources().getDimension(R.dimen.item_large_text_size_small);
                textSizeXLarge =
getResources().getDimension(R.dimen.item_x_large_text_size_small);
                break;
            case AppConfig.TEXT_SIZE_MEDIUM:
                textSizeSmall =
getResources().getDimension(R.dimen.item_small_text_size_medium);
                textSizeMedium =
getResources().getDimension(R.dimen.item_medium_text_size_medium);
                textSizeLarge =
getResources().getDimension(R.dimen.item_large_text_size_medium);
                textSizeXLarge =
getResources().getDimension(R.dimen.item_x_large_text_size_medium);
                break;
            case AppConfig.TEXT_SIZE_LARGE:
                textSizeSmall =
getResources().getDimension(R.dimen.item_small_text_size_large);
                textSizeMedium =
getResources().getDimension(R.dimen.item_medium_text_size_large);
                textSizeLarge =
getResources().getDimension(R.dimen.item_large_text_size_large);
                textSizeXLarge =
getResources().getDimension(R.dimen.item_x_large_text_size_large);
                break;
            case AppConfig.TEXT_SIZE_X_LARGE:
                textSizeSmall =
getResources().getDimension(R.dimen.item_small_text_size_x_large);
                textSizeMedium =
getResources().getDimension(R.dimen.item_medium_text_size_x_large);
                textSizeLarge =
getResources().getDimension(R.dimen.item_large_text_size_x_large);
                textSizeXLarge =
getResources().getDimension(R.dimen.item_x_large_text_size_x_large);
                break;

        }

        tabLayout = findViewById(R.id.layout_tabs);
        tabLayout.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
            @Override
            public void onTabSelected(TabLayout.Tab tab) {
                switch (tabLayout.getSelectedTabPosition()) {
                    case 0:
                        viewPager.setCurrentItem(0);
                        break;
                    case 1:
                        viewPager.setCurrentItem(1);
                        break;
                }
            }

            @Override
            public void onTabUnselected(TabLayout.Tab tab) {
            }
```

```
            @Override
            public void onTabReselected(TabLayout.Tab tab) {
            }
        });
        Intent intent = getIntent();
        if (intent != null) {
            if (intent.hasExtra("docOrderItem")) {
                orderData = (DocOrderItem)
intent.getSerializableExtra("docOrderItem");

                position = intent.getIntExtra("position", -1);
                fragmentConfirmOrder =
ConfirmOrderFragment.newInstance(orderData);
                fragmentDeliveryOrder =
DeliveryOrderFragment.newInstance(orderData);

                viewPager = findViewById(R.id.pager);
                viewPager.setUserInputEnabled(false);
                ScreenSlidePagerAdapter stateAdapter = new
ScreenSlidePagerAdapter(this);
                stateAdapter.addFragment(fragmentConfirmOrder);
                stateAdapter.addFragment(fragmentDeliveryOrder);
                viewPager.setAdapter(stateAdapter);
                for (OrderItemData orderItemData : orderData.getItems()) {
                    if (orderItemData.getQuantity() ==
orderItemData.getConfirmed_quantity()) {
                        confirm_Position_Count++;
                        confirm = true;
                    }
                    if (orderItemData.getConfirmed_quantity() > 0) confirm = true;
                }
            } else {
                ConfirmOrderActivity.this.onBackPressed();
                Toast.makeText(this,
getString(R.string.the_order_wa_not_selected), Toast.LENGTH_LONG).show();
            }
        }
        viewPager.registerOnPageChangeCallback(new
ViewPager2.OnPageChangeCallback() {
            @Override
            public void onPageSelected(int position) {
                tabLayout.selectTab(tabLayout.getTabAt(position));
            }
        });
        if (orderData.getConfirmed_status() == 3) {
            if (orderData.getItems().size() == confirm_Position_Count)
                NovaPoshtaApi.getInstance().sendEnRef(orderData, this, this, 1, -
1);
            else NovaPoshtaApi.getInstance().sendEnRef(orderData, this, this, 0, -
1);
        }
        animAlpha = AnimationUtils.loadAnimation(this, R.anim.animalbutton);
        ProgressDialog pDialog = new ProgressDialog(this);
        pDialog.setCancelable(false);
        String clientName;
        if (orderData.getDelivery() == null) {
            clientName = orderData.getPartner_name();
        } else {
            clientName = orderData.getDelivery().getLname() + " " +
orderData.getDelivery().getFname() + " " + orderData.getDelivery().getMname();
        }
        String title = "  " + getString(R.string.order_numbet_text) + " " +
orderData.getNumber() + "  " + getString(R.string.from_text) + " " + new
```

```
SimpleDateFormat("dd.MM.yyyy").format(new Date(orderData.getDate() * 1000)) + "
" + clientName;
        if (orderData.getSum() > 0) {
            DecimalFormatSymbols unusualSymbols = new DecimalFormatSymbols();
            unusualSymbols.setDecimalSeparator(',');
            unusualSymbols.setGroupingSeparator(' ');
            unusualSymbols.setCurrencySymbol("грн.");
            DecimalFormat formatSums = new DecimalFormat("###,###,###,##0.00
\u00A4", unusualSymbols);
            ((TextView)
findViewById(R.id.orderSumma)).setText(formatSums.format(orderData.getSum()));
        } else {
            ((TextView) findViewById(R.id.orderSumma)).setText("");
        }
        ((TextView)
findViewById(R.id.orderSumma)).setTextSize(TypedValue.COMPLEX_UNIT_PX,
textSizeLarge);
        toolbar.setTitle(title);
        for (int i = 0; i < toolbar.getChildCount(); i++) {
            final View child = toolbar.getChildAt(i);
            if (child instanceof TextView) {
                final TextView textView = (TextView) child;
                textView.setTextSize(TypedValue.COMPLEX_UNIT_PX, textSizeLarge);
            }
        }
        setSupportActionBar(toolbar);
        if (getSupportActionBar() != null)
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        getSupportActionBar().setDisplayShowHomeEnabled(true);
        new LinearSmoothScroller(this) {
            @Override
            protected int getVerticalSnapPreference() {
                return LinearSmoothScroller.SNAP_TO_START;
            }
        };
        if (orderData.getConfirmed_status() == 2 && (orderData.getDelivery() !=
null || (orderData.getTtn() != null && orderData.getTtn().getTtn_number() !=
null))) {
            tabLayout.setVisibility(View.VISIBLE);
        }
        RecyclerView.LayoutManager mLayoutManagerH = new LinearLayoutManager(this,
LinearLayoutManager.VERTICAL, false);
        RecyclerView historyRecyclerView =
findViewById(R.id.recycler_view_history_items);
        historyRecyclerView.setLayoutManager(mLayoutManagerH);
        historyItemsAdapter = new HistoryAdapter(this, HistoryItemsList);
        historyRecyclerView.setAdapter(new
AlphaInAnimationAdapter(historyItemsAdapter));
        historyRecyclerView.setItemAnimator(new SlideInUpAnimator());
        drawer = findViewById(R.id.drawer);
        drawer.setDrawerLockMode(DrawerLayout.LOCK_MODE_LOCKED_CLOSED);
        clear_history_list = findViewById(R.id.claer_history_list);
        clear_history_list.setOnClickListener(view -> {
            HistoryItemsList.clear();
            historyItemsAdapter.notifyDataSetChanged();
        });
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if (item.getItemId() == android.R.id.home) {
            onBackPressed();
            return true;
```

```
        }
        return super.onOptionsItemSelected(item);
    }

    @Override
    public boolean onKeyUp(int keyCode, KeyEvent event) {
        if(event.getKeyCode() == KeyEvent.KEYCODE_BACK){
            onBackPressed();
        }
        else {
            fragmentConfirmOrder.sendKey(keyCode, event);
        }
        return true;
    }

    @Override
    public void setOrderData(DocOrderItem orderData) {
        this.orderData = orderData;
        confirm_Position_Count = 0;
        for (OrderItemData orderItemData : orderData.getItems()) {
            if (orderItemData.getQuantity() ==
orderItemData.getConfirmed_quantity())
                confirm_Position_Count++;
        }
        fragmentConfirmOrder.save = true;
        fragmentConfirmOrder.showButton();
        if (confirm_Position_Count == orderData.getItems().size()) {
            String commentLog = "№" + orderData.getNumber() + " ";
            if (orderData.getTtn() != null) {
                if (orderData.getTtn().getDelivery_service() != null) {
                    ServicesDeliveryDTO servicesDeliveryDTO =
ExampleApp.getDataBaseManager().getServicesDeliveryDAO().getService(orderData.getT
tn().getDelivery_service());
                    if (servicesDeliveryDTO != null) {
                        if (servicesDeliveryDTO.getType() == 1) {
                            tabLayout.setVisibility(View.VISIBLE);
                            viewPager.setUserInputEnabled(true);
                            tabLayout.selectTab(tabLayout.getTabAt(1));
                        }
                    }
                }
                if (orderData.getTtn().getTtn_ref() != null) {
                    commentLog += ", " +
getString(R.string.text_comment_there_is_ref);
                } else if (orderData.getTtn().getTtn_number() != null) {
                    commentLog += ", " +
getString(R.string.text_comment_there_is_number);
                }
            } else if (orderData.getDelivery() != null &&
orderData.getDelivery().getDelivery_service() != null) {
                tabLayout.setVisibility(View.VISIBLE);
                viewPager.setUserInputEnabled(true);
                tabLayout.selectTab(tabLayout.getTabAt(1));
                commentLog += ", " +
getString(R.string.text_comment_there_is_info_delivery);
            }
            LogManager.saveLog(orderData.getGuid(), commentLog,
AppConfig.TYPE_LOG_CONFIRMED);
        }
    }

    @Override
    public void addHistory(HistoriItemInf historyItemsList) {
```

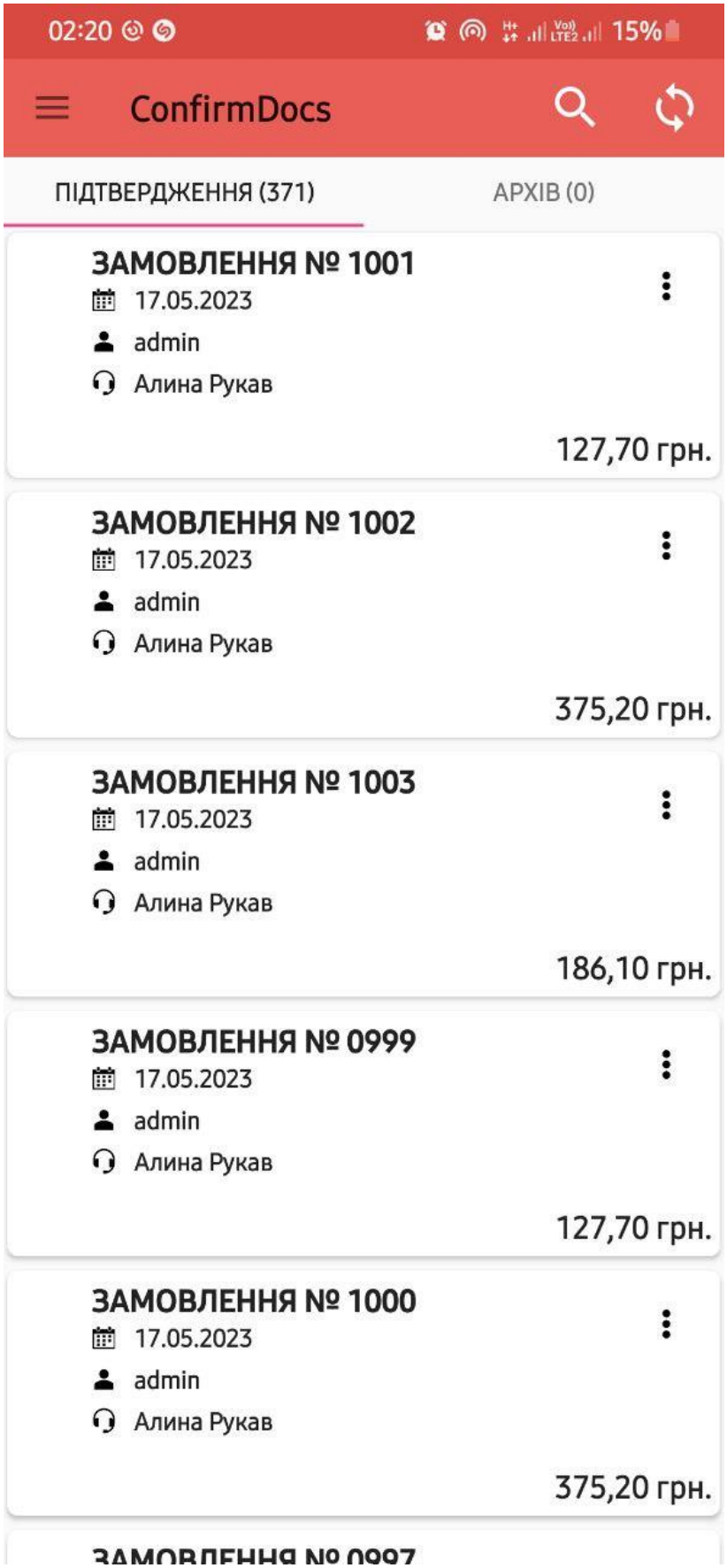
```
        int pos = HistoryItemsList.size();
        HistoryItemsList.add(historyItemsList);
        historyItemsAdapter.notifyItemInserted(pos);
    }

    @SuppressLint("RtlHardcoded")
    @Override
    public void click() {
        drawer.openDrawer(Gravity.RIGHT);
    }

    private void showMessageDialog(String message) {
        Intent intent = new Intent(ConfirmOrderActivity.this,
dialogActivity.class);
        intent.putExtra("message", message);
        startActivity(intent);
    }

    private Boolean SaveOrder() {
        if (fragmentConfirmOrder.save) {
            if (orderData.getConfirmed_status() == 3) {
                for (OrderItemData orderItemData : orderData.getItems()) {
                    UnsynchronizedOrdersItemDTO unsynchronizedOrdersItemDTO = new
UnsynchronizedOrdersItemDTO(orderItemData.getProduct(), orderData.getGuid(),
orderItemData.getConfirmed_quantity());

ExampleApp.getDataBaseManager().getUnsynchronizedOrdersItemDAO().insert(unsynchron
izedOrdersItemDTO);
                }
                String ttn_ref = "";
                if (orderData.getTtn() != null && orderData.getTtn().getTtn_ref()
!= null)
                    ttn_ref = orderData.getTtn().getTtn_ref();
                UnsynchronizedOrdersDTO unsynchronizedOrdersDTO = new
UnsynchronizedOrdersDTO(orderData.getGuid(), ttn_ref);

ExampleApp.getDataBaseManager().getUnsynchronizedOrdersDAO().insert(unsynchronized
OrdersDTO);
            }
            Intent intent = new Intent();
            intent.putExtra(EXTRA_DOC_ORDER_ITEM, orderData);
            intent.putExtra(EXTRA_POSITION, position);
            setResult(RESULT_OK, intent);
            finish();
            return true;
        }
        return false;
    }

    @Override
    public void onStart() {
        super.onStart();
        EventBus.getDefault().register(this);
    }

    @Override
    public void onStop() {
        EventBus.getDefault().unregister(this);
        super.onStop();
    }

    public void cancel_confirm(String guid) {
        final SharedPreferences[] preferences = new SharedPreferences[1];
```

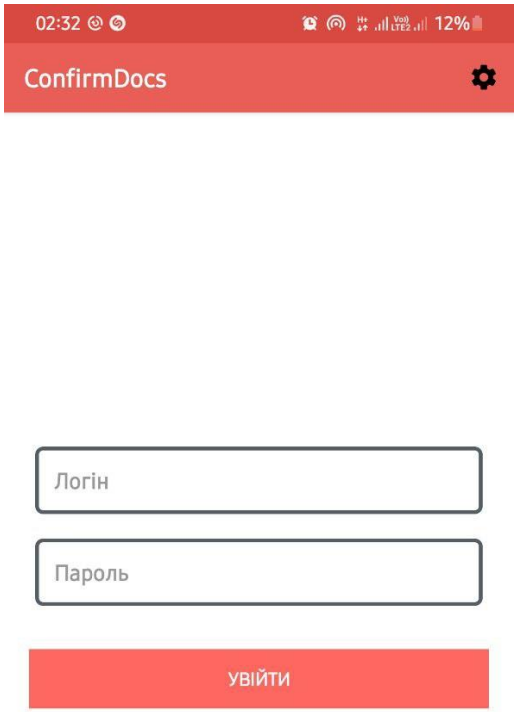
```
        preferences[0] = getSharedPreferences(AppConfig.PREF_NAME,
Context.MODE_PRIVATE);

            UserDTO userDTO =
ExampleApp.getDataBaseManager().getUserDAO().getUser();
            if(userDTO!=null){
            String token = userDTO.getToken();
            JsonObject ttn_ref = new JsonObject();
            ttn_ref.addProperty("guid", guid);
            String loc =
String.valueOf(Resources.getSystem().getConfiguration().locale);
            if (loc.startsWith("uk")) {
                loc = loc.substring(0, 2);
            } else if (loc.startsWith("ru")) {
                loc = loc.substring(0, 2);
            } else loc = "en";
            if (preferences[0].contains(AppConfig.PREF_ADDRESS) &&
preferences[0].contains(AppConfig.PREF_API)) {
                String uri = preferences[0].getString(AppConfig.PREF_ADDRESS, "");
                String api = preferences[0].getString(AppConfig.PREF_API, "");

ExampleApp.GetRestAPI(uri).sendEnRef(AppConfig.API_POST_CANCEL_CONFIRM, "Bearer "
+ token, loc, ttn_ref, api, "ConfirmDocs/" + BuildConfig.VERSION_NAME).enqueue(new
Callback<JsonObject>() {
                    @Override
                    public void onResponse(@NonNull Call<JsonObject> call,
@NonNull Response<JsonObject> response) {
                        if (!response.isSuccessful()) {
                            switch (response.code()) {
                                case 401:

ExampleApp.getInstance().logoutUser(ConfirmOrderActivity.this);
                                    break;
                                case 403:
                                    Toast.makeText(ConfirmOrderActivity.this,
"Error: " + response.code() + "\n" + response.message(),
Toast.LENGTH_LONG).show();

ExampleApp.getInstance().logoutUser(ConfirmOrderActivity.this);
                                    break;
                            }
                        }
                    }

                    @Override
                    public void onFailure(@NonNull Call<JsonObject> call, @NonNull
Throwable t) {

                    }
                });
            }
        }
    }

    @Subscribe(threadMode = ThreadMode.MAIN)
    public void onMessageEvent(BustMessageEvent event) {
        switch (event.getTypeEvent()) {
            case BustMessageEvent.BUST_EVENT_NP_API_ERROR_MESSAGE:
                showMessageDialog(event.getData());
                break;
            case BustMessageEvent.BUST_EVENT_NP_WAREHOUSE_WEIGHT_ERROR:
                orderData.setDepartmentErrorEn(true);
                break;
```

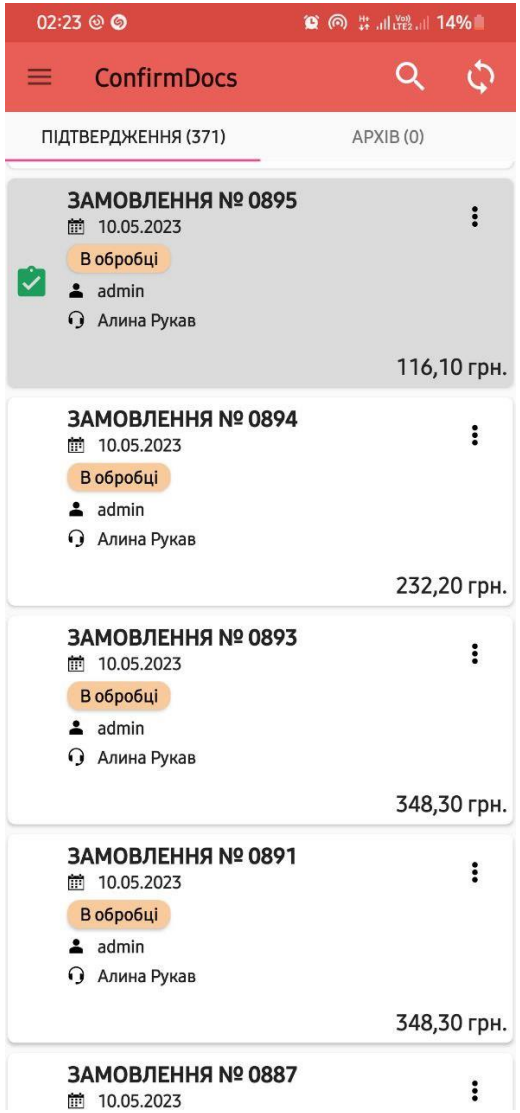
```
            case BustMessageEvent.BUST_EVENT_SEND_TTN:
                switch (event.getData()) {
                    case "1":
                        if (orderData.getItems().size() == confirm_Position_Count)
                            orderData.setConfirmed_status(2);
                        else orderData.setConfirmed_status(1);
                        if (SaveOrder()) finish();
                        fragmentConfirmOrder.save = true;
                        break;
                    case "2":
                        Toast.makeText(this,
R.string.tse_zamovlennya_already_confirmed, Toast.LENGTH_LONG).show();
                        Intent intent = new Intent();
                        intent.putExtra(EXTRA_DOC_ORDER_ITEM, orderData);
                        intent.putExtra(EXTRA_POSITION, position);
                        intent.putExtra(EXTRA_CONFIRM, true);
                        setResult(RESULT_OK, intent);
                        finish();
                        fragmentConfirmOrder.save = true;
                        break;
                    case "3":
                        orderData.setConfirmed_status(3);
                        if (SaveOrder()) finish();
                        fragmentConfirmOrder.save = true;
                        break;
                    case "4":
                        orderData.setConfirmed_status(2);
                        fragmentConfirmOrder.save = false;
                        fragmentConfirmOrder.HideFabSave();
                        break;
                }
                break;
        }
    }

    public double getTextSizeSmall() {
        return textSizeSmall;
    }

    public void setTextSizeSmall(float textSizeSmall) {
        this.textSizeSmall = textSizeSmall;
    }

    public double getTextSizeMedium() {
        return textSizeMedium;
    }

    public void setTextSizeMedium(float textSizeMedium) {
        this.textSizeMedium = textSizeMedium;
    }

    public double getTextSizeXLarge() {
        return textSizeXLarge;
    }

    public void setTextSizeXLarge(float textSizeXLarge) {
        this.textSizeXLarge = textSizeXLarge;
    }

    private static class ScreenSlidePagerAdapter extends FragmentStateAdapter {
        private final ArrayList<Fragment> arrayList = new ArrayList<>();

        public ScreenSlidePagerAdapter(FragmentActivity fa) {
```

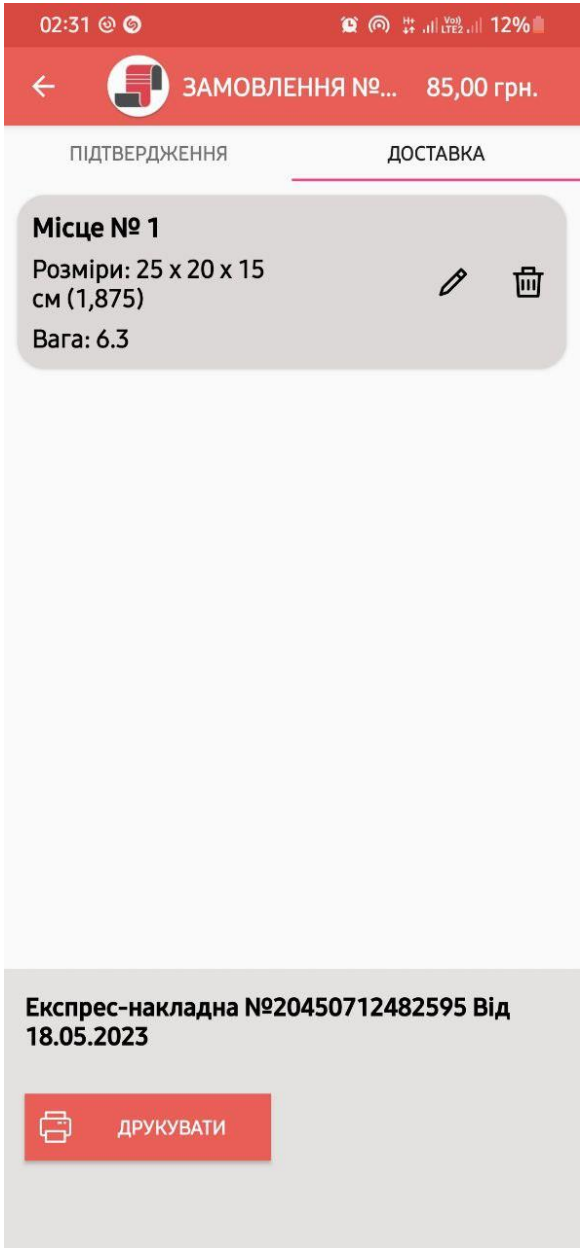
```
            super(fa);
        }

        @NonNull
        @Override
        public Fragment createFragment(int position) {
            return arrayList.get(position);
        }

        public void addFragment(Fragment fragment) {
            arrayList.add(fragment);
        }

        @Override
        public int getItemCount() {
            return COUNT_PAGER;
        }
    }

}
```

ДОДАТОК Б


02:20
15%
ConfirmDocs
ПІДТВЕРДЖЕННЯ (371)
АРХІВ (0)
ЗАМОВЛЕННЯ № 1001
17.05.2023
admin
Алина Рукав
127,70 грн.
ЗАМОВЛЕННЯ № 1002
17.05.2023
admin
Алина Рукав
375,20 грн.
ЗАМОВЛЕННЯ № 1003
17.05.2023
admin
Алина Рукав
186,10 грн.
ЗАМОВЛЕННЯ № 0999
17.05.2023
admin
Алина Рукав
127,70 грн.
ЗАМОВЛЕННЯ № 1000
17.05.2023
admin
Алина Рукав
375,20 грн.


02:32
12%
ConfirmDocs
Логін
Пароль
УВІЙТИ

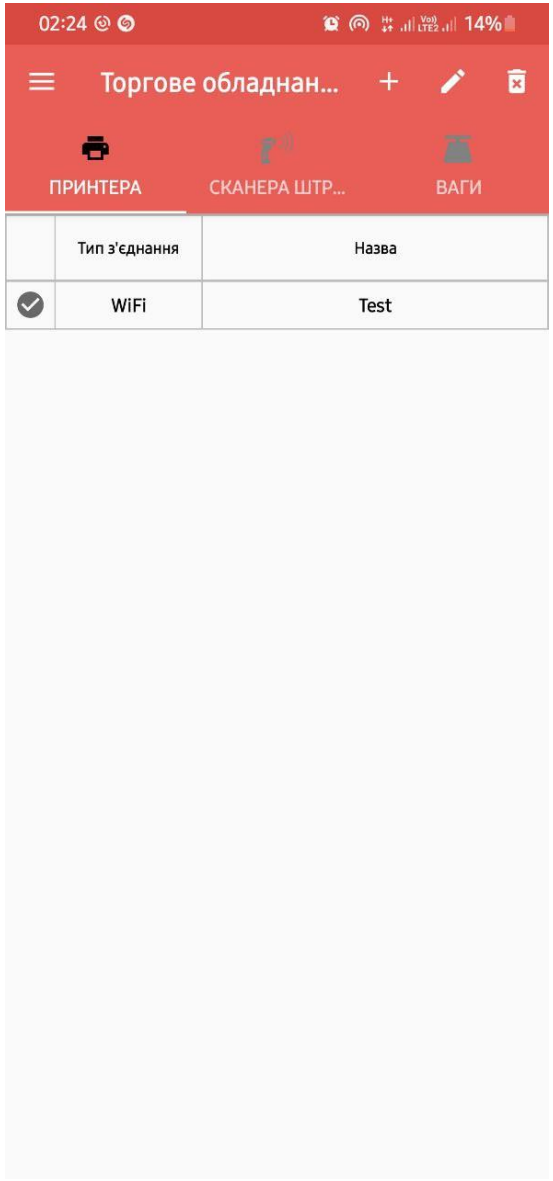

02:23
14%
ConfirmDocs
ПІДТВЕРДЖЕННЯ (371)
АРХІВ (0)
ЗАМОВЛЕННЯ № 0895
10.05.2023
В обробці
admin
Алина Рукав
116,10 грн.
ЗАМОВЛЕННЯ № 0894
10.05.2023
В обробці
admin
Алина Рукав
232,20 грн.
ЗАМОВЛЕННЯ № 0893
10.05.2023
В обробці
admin
Алина Рукав
348,30 грн.
ЗАМОВЛЕННЯ № 0891
10.05.2023
В обробці
admin
Алина Рукав
348,30 грн.
ЗАМОВЛЕННЯ № 0887
10.05.2023

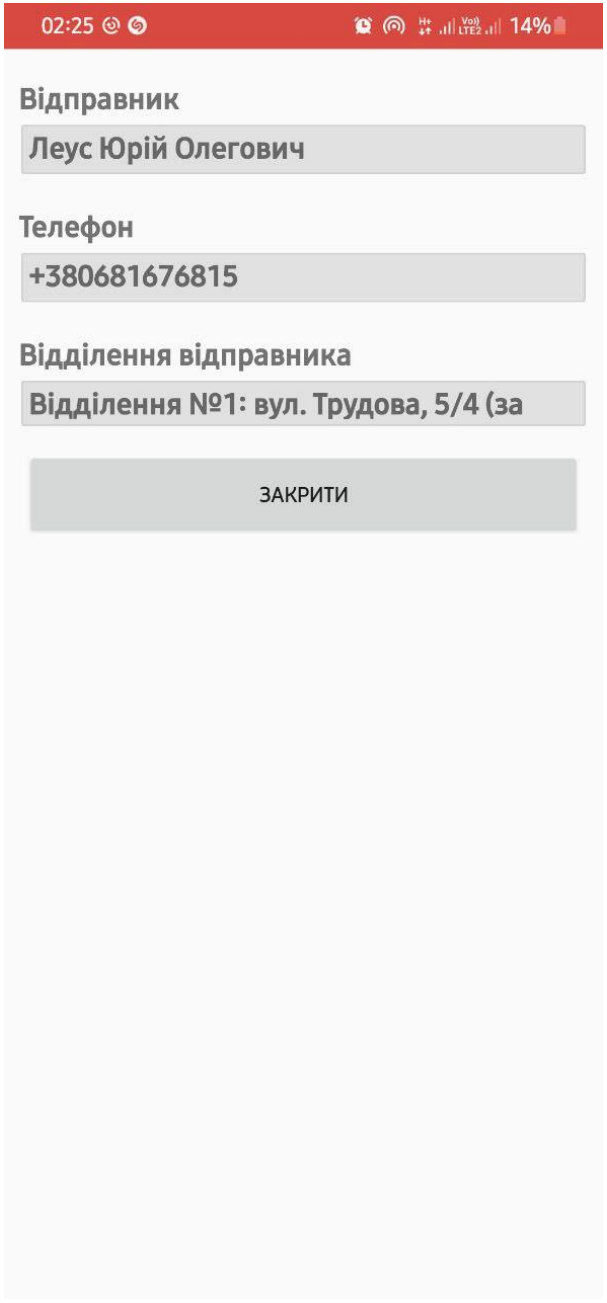

02:31
12%
ЗАМОВЛЕННЯ №...
85,00 грн.
ПІДТВЕРДЖЕННЯ
ДОСТАВКА
Місце № 1
Розміри: 25 x 20 x 15 см (1,875)
Вага: 6.3
Експрес-накладна №20450712482595 Від 18.05.2023
ДРУКУВАТИ

02:31
ЗАМОВЛЕННЯ №... 85,00 грн.
ДОСТАВКА
ПІДТВЕРДЖЕННЯ
Місце № 1
Розміри: 25 x 20 x 15 см (1,875)
Вага: 6,3
ПРИКЛАДНИЙ ПРИКЛАД
СКЛ 4
ХМЕЛЬНИЦЬКИЙ
ВІД 19.09.2023
20 4507 7248 2595
6,3 кг
ЗАКРИТИ
ДРУКУВАТИ
НАЗАД
Експрес-накладна №20450717482595 від 18.05.2023

02:21
ЗАМОВЛЕННЯ... 4 045,50 грн.
Код
Назва
К-сть
Кільк. підтверджених.
000013 KRON-BORG Рушник STIDSVIG 50x100см м'ята. 4,0 0,5
0000-021301 Samsung 1234 5,0 0,0
000016 Компьютерный стол Ирма 80 + Ника Мебель Вишня Оксфорд 6,0 3,0
00001973 Швейна машинка 2,0 2,0
Ш002 Шорты мужские "Adidas" (Ш002), Т.Серый П 3,0 2,0
Ш0026 Шорты мужские батал "Adidas" (Ш0026), Т.Серый П 3,0 3,0
Ш0036 Шорты мужские батал "Puma" (Ш0036), Т.Серый П 6,0 5,0
Ш0016 Шорты мужские батал, "Nike AIR" (Ш0016), Т.Серый П 3,0 3,0
Ш005 Шорты мужские однотонные (Ш005), Красный П 6,0 3,0
Історія
Коментар

02:25
Служби доставки
Нова Пошта 2
Приватна особа
Приватна особа
УкрПошта 0
Meest Express 0
Delivery 0
Test 0
Justin 0


02:24
14%
Торгове обладнан...
ПРИНТЕРА
СКАНЕРА ШТР...
ВАГИ
Тип з'єднання
Назва
WiFi
Test


02:25
14%
Відправник
Леус Юрій Олегович
Телефон
+380681676815
Відділення відправника
Відділення №1: вул. Трудова, 5/4 (за
ЗАКРИТИ

ДОДАТОК В
(обов'язковий)

ПРОХОДЖЕННЯ АНТИПЛАГІАТУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 3.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 12%**

	Документ		Сумарний збіг по Базі Даних	
ID: 114601 Назва: БКР Мобільний застосунок на платформі Android для автоматизації бізнес- процесів торгового підприємства Додано в БД: 2023-06-02 Автора: Леус Ю.О. Керівники: Гурман І.В. к.т.н. доц Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	93990	798	3923 (4%)	41 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1015399564

Дата перевірки:
02.06.2023 17:07:39 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2023 17:42:15 EEST

ID користувача:
100005589

Назва документа: КвР Леус(1) плагіат

Кількість сторінок: 71 **Кількість слів:** 13349 **Кількість символів:** 111451 **Розмір файлу:** 570.53 KB **ID файлу:** 1015063492

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.23%
Схожість

Найбільша схожість: 2.27% з джерелом з Бібліотеки (ID файлу: 1015045630)

5.59% Джерела з Інтернету	409	Сторінка 73
3.93% Джерела з Бібліотеки	109	Сторінка 75

0% Цитат

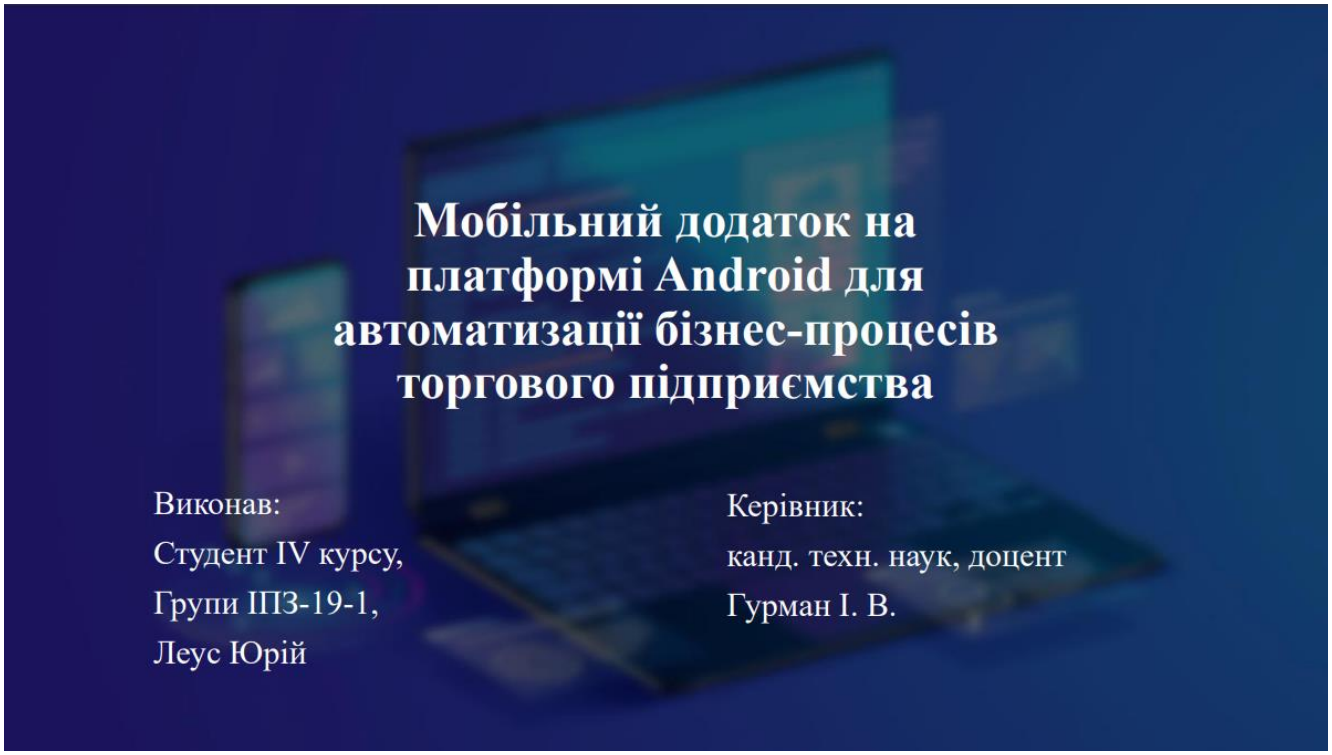
Вилучення цитат вимкнене

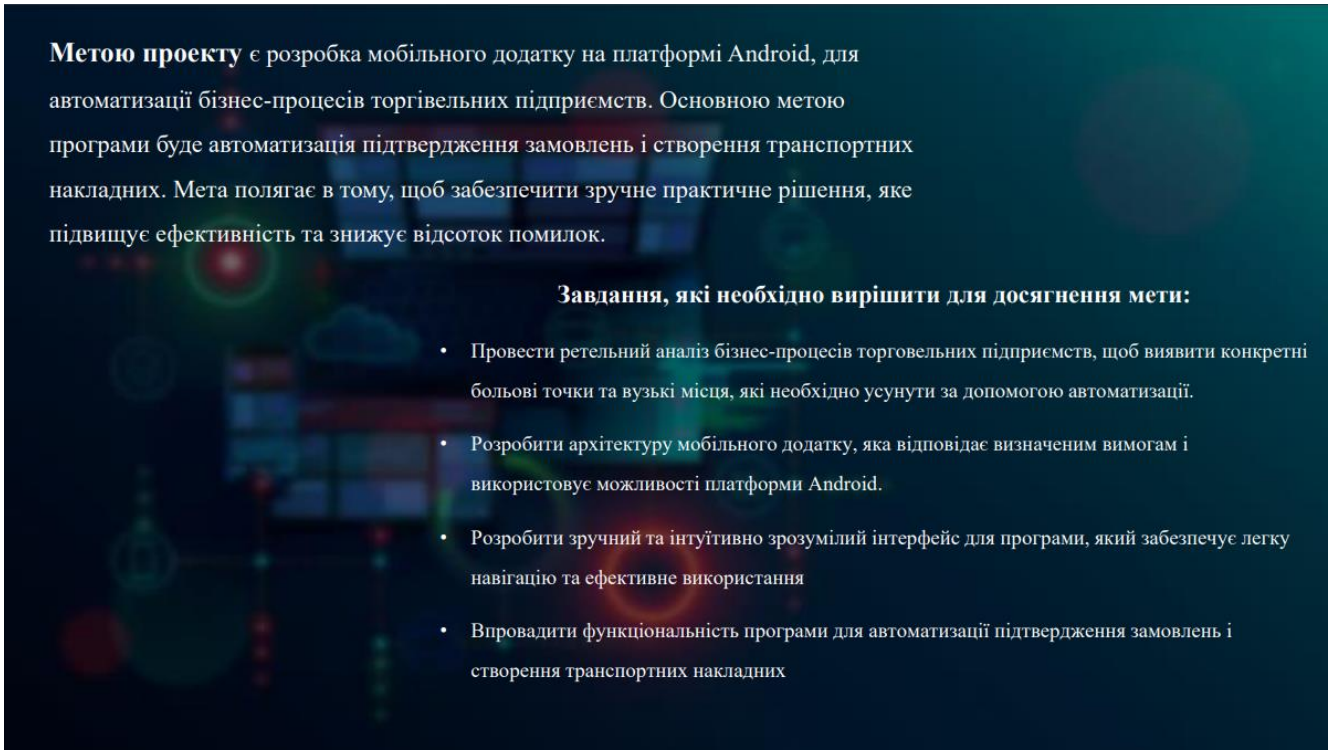
Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

ДОДАТОК Г


Мобільний додаток на платформі Android для автоматизації бізнес-процесів торгового підприємства
Виконав:
Студент IV курсу,
Групи ІПЗ-19-1,
Леус Юрій
Керівник:
канд. техн. наук, доцент
Гурман І. В.


Метою проекту є розробка мобільного додатку на платформі Android, для автоматизації бізнес-процесів торгівельних підприємств. Основною метою програми буде автоматизація підтвердження замовлень і створення транспортних накладних. Мета полягає в тому, щоб забезпечити зручне практичне рішення, яке підвищує ефективність та знижує відсоток помилок.
Завдання, які необхідно вирішити для досягнення мети:
• Провести ретельний аналіз бізнес-процесів торговельних підприємств, щоб виявити конкретні больові точки та вузькі місця, які необхідно усунути за допомогою автоматизації.
• Розробити архітектуру мобільного додатку, яка відповідає визначеним вимогам і використовує можливості платформи Android.
• Розробити зручний та інтуїтивно зрозумілий інтерфейс для програми, який забезпечує легку навігацію та ефективне використання
• Впровадити функціональність програми для автоматизації підтвердження замовлень і створення транспортних накладних

У сучасному бізнес-середовищі, яке швидко розвивається, організаціям необхідно впроваджувати інноваційні технології, щоб залишатися конкурентоспроможними та оптимізувати свою діяльність. Мобільні програми стали важливим інструментом для бізнесу, надаючи їм можливість оптимізувати процеси та підвищити ефективність.

Тема кваліфікаційної роботи є вкрай актуальною та необхідною в сучасному бізнес-ландшафті. Тому що існують проблеми, з якими стикаються торговельні компанії в ефективному та результативному управлінні своїми операціями. Ручне підтвердження замовлень і створення транспортних накладних може зайняти багато часу та бути схильним до помилок. Автоматизація цих процесів не тільки підвищить ефективність і точність, але й забезпечить своєчасну доставку товарів і послуг клієнтам

Manhattan SCALE

Переваги:

- Масштабованість: Manhattan SCALE може працювати з різними типами складів та міркувань - від невеликих до дуже великих.
- Налагодження та підтримка: Компанія Manhattan Associates надає високоякісне налагодження та підтримку своїх продуктів

Недоліки:

- Складність: Manhattan SCALE має багато функціональності, що може бути складним для вивчення та реалізації.
- Залежність від постачальника: У разі використання Manhattan SCALE компанії стають залежними від компанії Manhattan Associates

SAP Extended Warehouse Management

Переваги:

Основною перевагою EWM є можливість інтеграції з іншими програмними продуктами SAP, що дозволяє створювати повністю інтегровану систему управління складом, яка забезпечує зведення складу до мінімуму та підвищення ефективності операцій.

Недоліки:

- Висока складність встановлення та налаштування
- Висока вартість
- Система може бути надто складною для менших складів
- Високі вимоги до обладнання та програмного забезпечення

Архітектура

Архітектура програми для Android відіграє вирішальну роль у визначенні її якості та успіху.

Добре розроблена архітектура сприяє багаторазовому використанню коду, що скорочує час і вартість розробки.

Таким чином, вибір правильної архітектури для програми Android є важливим для забезпечення її успіху та довговічності.

Були розглянуті такі типи архітектур:

- Model-View-Controller
- Model-View-Presenter
- Model-View-ViewModel
- Flux

Обрана архітектура - (MVVM)

- Однією з головних переваг архітектури MVVM є її чіткий розподіл проблем. Компонент Model представляє дані та бізнес-логіку програми, компонент View представляє інтерфейс користувача, а компонент ViewModel діє як посередник між компонентами Model і View. Такий розподіл завдань полегшує обслуговування, тестування та модифікацію програми в майбутньому.

- Крім того, архітектура MVVM полегшує зв'язування даних, що є потужною функцією розробки Android. Зв'язування даних дозволяє програмі автоматично оновлювати інтерфейс користувача на основі змін даних, не вимагаючи ручного втручання розробника. Це спрощує процес розробки та знижує ризик помилок.

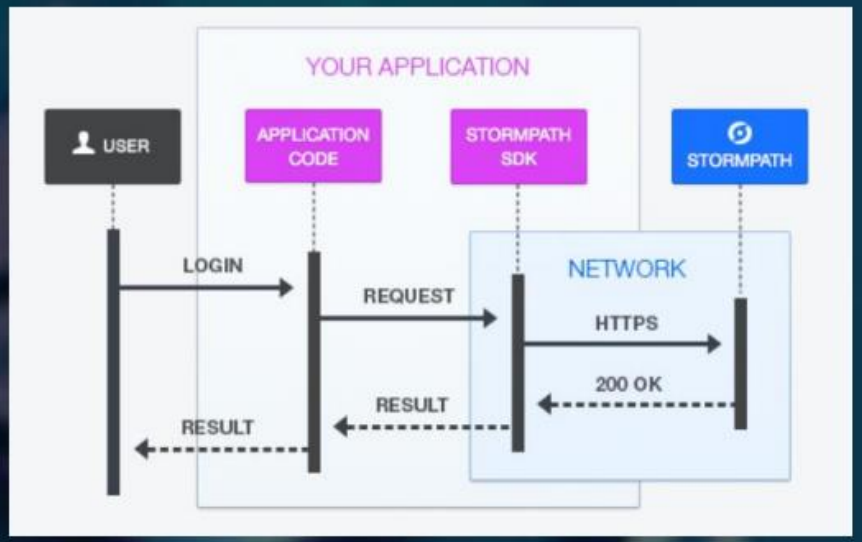
Деякі ключові аспекти модуля автентифікації включають:

Вхід: зареєстровані користувачі можуть увійти в програму за допомогою своїх облікових даних. Модуль перевіряє надану інформацію та надає доступ авторизованим користувачам.

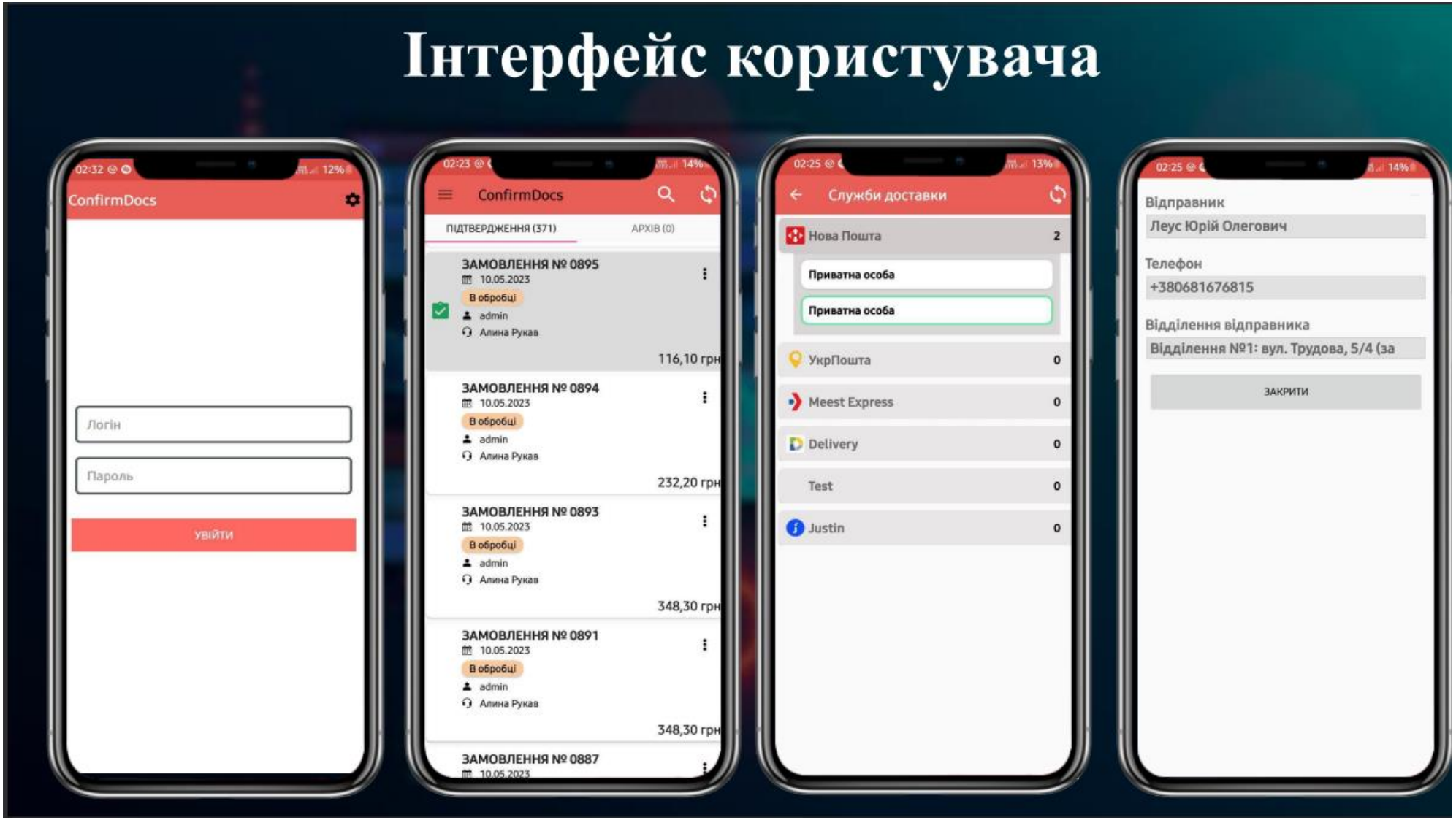
Безпека пароля: модуль автентифікації використовує розширені заходи безпеки для захисту паролів користувачів. Він використовує такі методи, як хешування.

Аутентифікація на основі маркерів: для підвищення безпеки та покращення взаємодії з користувачем модуль використовує автентифікацію на основі маркерів. Після успішного входу генерується унікальний маркер, який використовується для перевірки наступних запитів.

Керування сеансами: модуль ефективно керує сеансами користувачів, щоб забезпечити безперебійну та безпечну роботу користувача



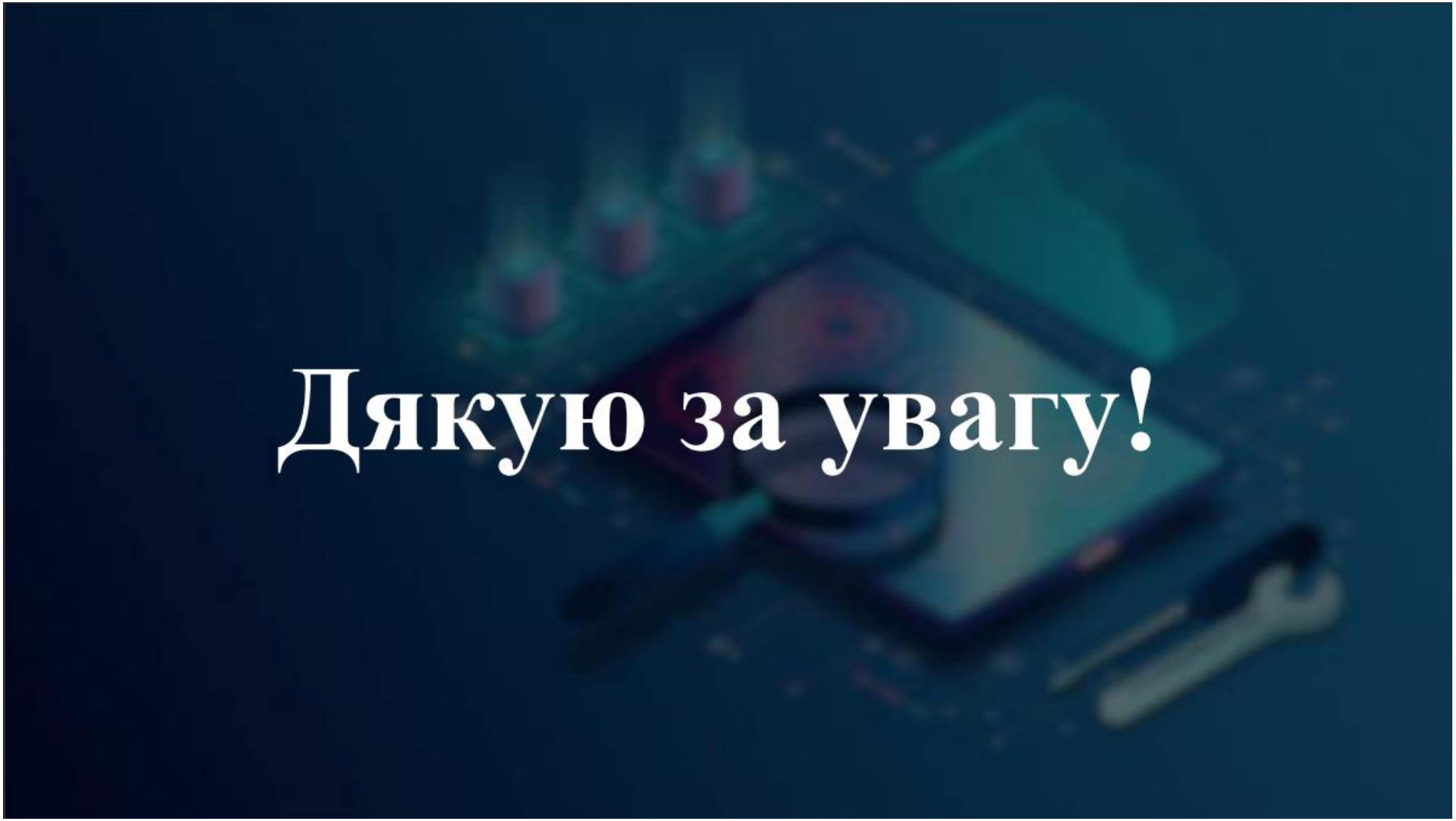
Інтерфейс користувача



Інтерфейс користувача

Висновки

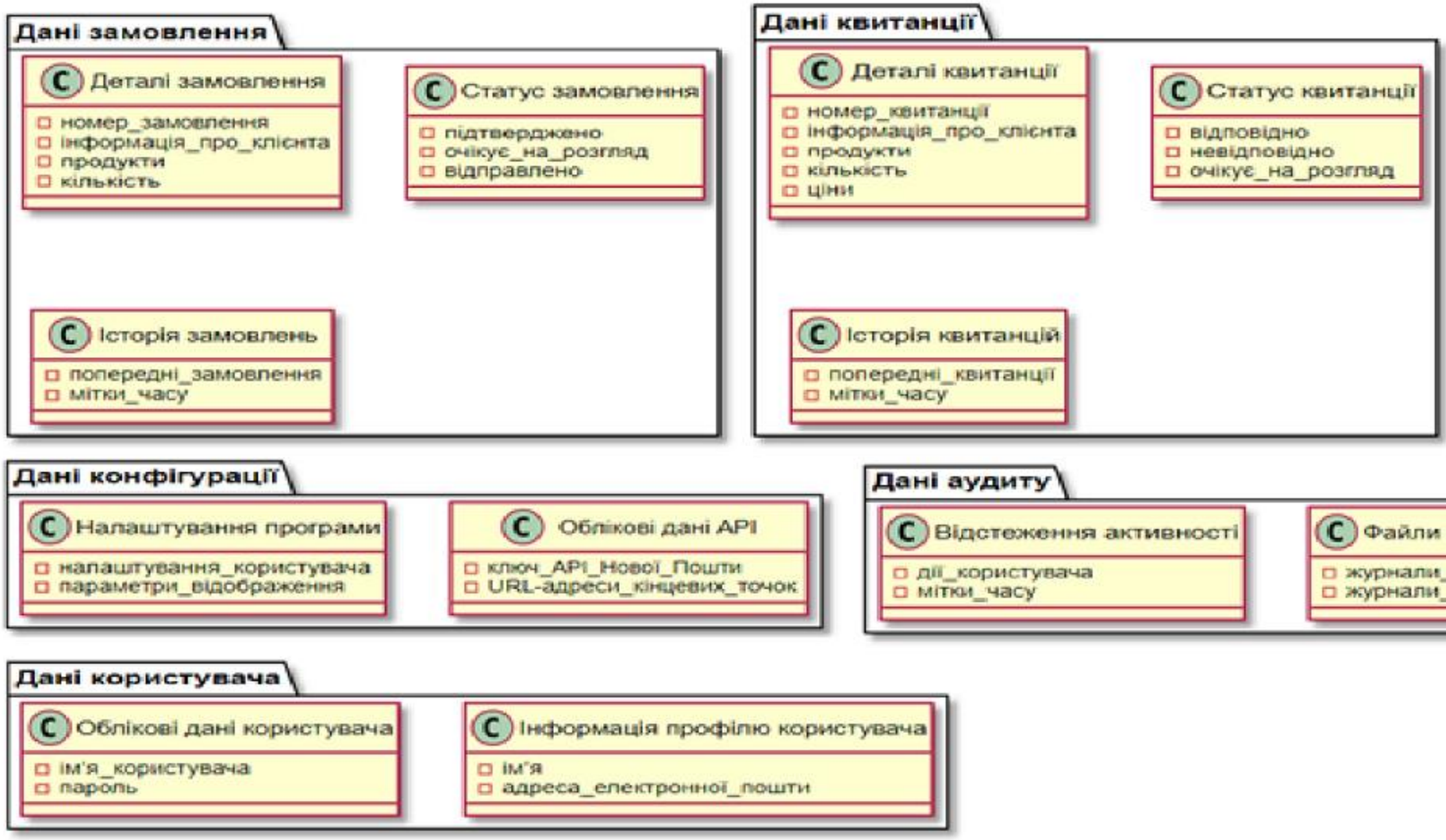
- У підсумку дана кваліфікаційна робота була присвячена розробці мобільного додатку на платформі Android для автоматизації бізнес-процесів на підприємствах торгівлі. Основною метою проекту було створити практичне та ефективне рішення, яке оптимізує процеси підтвердження замовлення та створення транспортних накладних, що призведе до підвищення ефективності і зниження витрат.
- Протягом усього процесу розробки були виконані різні етапи, включаючи збір вимог, аналіз системи, архітектурний дизайн, декомпозицію модулів, розробку бази даних і впровадження ключових функцій, таких як автентифікація, інтеграція з API Нової Пошти, підтвердження замовлення та обробка помилок.
- Завдяки використанню архітектури MVVM і відповідних технологій і бібліотек, таких як, Room для керування базами даних і Retrofit для інтеграції API, додаток було розроблено таким чином, щоб дотримуватися найкращих практик і забезпечити масштабованість, зручність обслуговування та повторне використання коду.
- Загалом ця кваліфікаційна робота успішно досягла поставленої мети – розробити мобільний додаток для автоматизації бізнес-процесів на підприємствах торгівлі. Він служить цінним інструментом, який спрощує та оптимізує підтвердження замовлень і створення транспортних накладних, що зрештою призводить до покращення ефективності бізнесу.


Дякую за увагу!

ДОДАТОК Д

(Обов’язковий)

ГРАФІЧНІ МАТЕРІАЛИ



					КвРІПЗ.190133.19.12.Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Діаграма декомпозиції даних	Літера / Маса / Масштаб
Розробив		Леус Ю. О.				
Керівник		Гурман І. В.				Аркуш 1 / Аркушів 3
Консульт.						
Н. Контр.		Форкун Ю. В.				
Зав. каф.		Бедратюк Л.П.				

UnsynchronizedOrdersDTO

- Назва Тип
- guid String
- en_ref String

UnsynchronizedOrdersItemDTO

1. guid String
2. order_guid String
3. confirm_count Double

NPErrorDTO

- code String
- massage String
- massage_ru Stri
- massage_ua Stri

ServicesDeliveryDTO

- guid String
- type Int
- title String

CabinetDeliveryDTO

- guid String
- api_key String
- contact_sender_ref String
- contact_sender_name String
- sender_phone String
- sender_ref String
- sender_name String
- sender_city_ref String
- sender_department_ref String
- delivery_services_id String
- title String (за замовчуванням)

users

- guid String
- name String
- login String
- password String
- token String

LogActionsDTO

- id Int
- date Long
- login String
- type Int
- order_guid String
- comment String

TemplatesDeliveryDTO

- guid String
- title String
- length Int
- width Int
- height Int
- weight Double

PrinterDTO

- guid String
- type_connection Int
- name String
- address String
- active Int
- is_use_scale Boolean
- type_paper Int
- type_printer Int
- port String

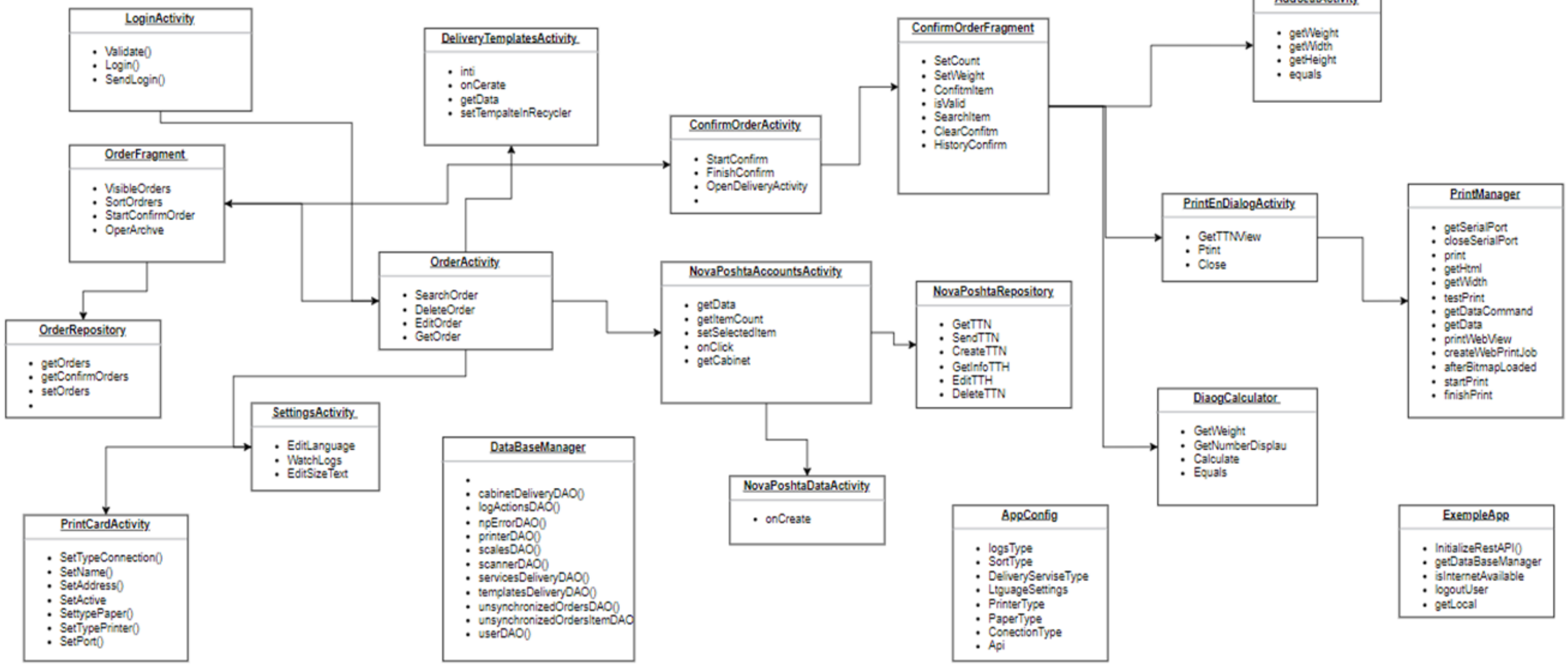
ScannerDTO

ScannerDTO
- guid String
- name String
- address String
- active Int

ScalesDTO

- guid String
- name String
- active Int
- protocol Int

					КвРІПЗ.190133.19.12.Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	ER – діаграма бази даних	Літера / Маса / Масштаб
Розробив		Леус Ю. О.				
Керівник		Гурман І. В.				Аркуш 2 / Аркушів 3
Консульт.						
Н. Контр.		Форкун Ю. В.				
Зав. каф.		Бедратюк Л.П.				



					КвРІПЗ.190133.19.12.Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	ER – діаграма класів	Літера / Маса / Масштаб
Розробив		Леус Ю. О.				
Керівник		Гурман І. В.				Аркуш 3 / Аркушів 3
Консульт.						
Н. Контр.		Форкун Ю. В.				
Зав. каф.		Бедратюк Л.П.				

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Леус Ю. О.
Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-19-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

30.05.2023
дата

підпис

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 3.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 12%**

	Документ		Сумарний збіг по Базі Даних	
ID: 114601 Назва: БКР Мобільний застосунок на платформі Android для автоматизації бізнес- процесів торгового підприємства Додано в БД: 2023-06-02 Автора: Леус Ю.О. Керівники: Гурман І.В. к.т.н. доц Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	93990	798	3923 (4%)	41 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

Дата перевірки:
02.06.2023 17:07:39 EEST

Дата звіту:
02.06.2023 17:42:15 EEST

ID перевірки:
1015399564

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100005589

Назва документа: **КвР Леус(1) плагіат**

Кількість сторінок: **71** Кількість слів: **13349** Кількість символів: **111451** Розмір файлу: **570.53 KB** ID файлу: **1015063492**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.23%
Схожість

Найбільша схожість: **2.27%** з джерелом з Бібліотеки (ID файлу: **1015045630**)

5.59% Джерела з Інтернету	400	Сторінка 73
3.63% Джерела з Бібліотеки	100	Сторінка 71

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн звіті.

Замінені символи	106
Підозріле форматування	13 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Леус Юрій Олегович

Тема Мобільний застосунок на платформі Android для автоматизації бізнес-процесів торгівельного підприємства

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 69

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі було досліджено і проаналізовано предметну область, визначено усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих програм на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Розглянуто інструменти для реалізації спроектованих рішень, в результаті чого було вибрано інструментарій для створення програмного забезпечення. Також було проведено тестування програми, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експлуатації

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз сучасних архітектур, розглянуто їх переваги і недоліки та визначено архітектуру, яка буде використовуватися при розробці. У третьому розділі виконано практичну розробку програмних модулів і описано їх особливості, в результаті чого створено програмний продукт. Після цього було проведено тестування системи

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки сьогодні існує попит на різні android-застосунки для автоматизації бізнес-процесів торгового підприємства. Під час розробки застосунку було враховано недоліки відомих рішень, а також застосовано сучасні технології розробки.

5. Негативні сторони роботи У роботі реалізовано інтеграцію лише з однією службою доставки, було б доцільно інтегрувати ще інші службами доставки.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Бобровнікова Кіра Юліївна, кандидат технічних наук, доцент, кафедри комп'ютерної інженерії та інформаційних систем (КІІС) ХНУ

"05" 06 2023 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом (ами) перевірки текстів на плагіат:

Назва: «Мобільний застосунок на платформі Android для автоматизації бізнес-процесів торговельного підприємства»
Автор: Леус Юрій Олегович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Гурман Іван Васильович, кандидат технічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титульний аркуш, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 8,23% і адресується до 409 джерел, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 5.06.23

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Іван ГУРМАН