

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ

к.т.н, доц. Михелєв І.Л.

" ____ " _____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "бакалавр"

на тему: «Системний аналіз технологій мобільної розробки для додатку
адміністратора оренди приміщень»

Виконав: студент групи

(підпис) Золін О.В.

Керівник роботи:

(посада, науковий ступінь вчене звання)

(підпис) Пармас В.К.

м. Миколаїв – 2022 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління
проектами
Кафедра інформаційних управляючих систем та технологій
Спеціальність 124 "Системний аналіз"
Освітня програма "Системний аналіз"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми

к.т.н, доц.НУК Приходько К.С.

" ____ " _____ 2022 р.

*ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття ступеня вищої освіти "бакалавр"*

Студенту Золіну Олексію Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Системний аналіз технологій мобільної розробки для додатку адміністратора оренди приміщень»

Керівник роботи Партас Віктор Кирилович

Затверджені наказом ректора № 269-уч від "18" травня 2022 року.

2. Термін подання роботи: 20 червня 2022 р.

3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем.

4. Перелік питань, що належать до розробки (найменування розділів)

Розділ 1: Розробка концепції інформаційної системи мобільного додатку адміністратора оренди приміщень. Розділ 2: Розробка проекту інформаційної системи. Розділ 3: Реалізація проекту автоматизованої системи. Розділ 4: Охорона праці.

5. Перелік презентаційних

матеріалів _____

	Прізвище, ініціали та посада консультанта		
		завдання видав	завдання прийняв
1	Партас В.К.		
2	Партас В.К.		
3	Партас В.К.		
4	Гурец Н.В.		

7. Дата видачі завдання 14 лютого 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1	Вивчення і опис предметної області	14.03.2022	
2	Розробка концепції системи	21.03.2022	
3	Розділ з розробки проектних рішень по системі	28.03.2022	
4	Реалізація проекту та розробка робочої документації	04.04.2022	
5	Вирішення питань охорони праці	18.04.2022	
6	Виконання графічних документів	25.04.2022	
7	Оформлення пояснювальної записки	05.05.2022	
8	Подання ДР на перевірку на плагіат	15.05.2022	
9	Подання ДР рецензенту	20.06.2022	
10	Подання ДР на державний захист	26.06.2022	

Студент _____
(підпис)

Золін О.В.
(ПІБ)

Керівник роботи _____
(підпис)

Партас В.К.
(ПІБ)

АНОТАЦІЯ

Основною темою дипломного проекту є проектування та програмна реалізація мобільного додатку інформаційної підтримки роботи менеджера з оренди приміщень.

Проаналізовано комерційні аналоги систем, що реалізовано у вигляді десктопних та web-додатків. Було враховано надмірну функціональність подібних систем та прийнято рішення щодо реалізації базового функціоналу у зручній формі мобільного додатку.

Ескізний та технічний проект додатку підготовлено з використанням нотацій UML 2.0.

Програмну реалізацію виконано з використанням Java, СУБД SQLite та MySQL. При розробці використано інтегроване середовище розробки Android – Studio, що сьогодні є офіційним додатком для розробки Android- додатків.

Додаток включає в себе модуль графічного інтерфейсу користувача, модуль реалізації моделі доступу до даних MVC та модуль синхронізації локальної проекції даних із основною серверною БД.

Додаток протестований з використанням стандартних засобів модульного та інтеграційного тестування та готовий до дослідної експлуатації.

Основна частина включає 59 сторінок, 18 рисунків, 3 таблиці. Також робота містить 3 додатки.

ANNOTATION

The main topic of the diploma project is the design and robust implementation of mobile add-on informational support for the work of the manager of the rental apartments.

Analyzed the commercial analogues of the systems that are implemented in the form of desktop and web add-on. The superior functionality of such systems was taken into account and the decision was made to implement the basic functionality in a user-friendly form of a mobile add-on.

An intuitive and technical design of the add-on was prepared using UML 2.0 notations.

The program is implemented using Java, SQLite and MySQL. During the development, the integrated development environment Android - Studio was used, which is today an official add-on for the development of Android add-ons.

The add-on includes the graphical user interface module, the MVC data access module implementation model and the synchronization module of the local data projection with the main server's database.

The addendum has been tested using standard tools of modular and integration testing and is ready for investigative exploitation.

The main part includes 59 pages, 18 figures, 3 tables. Also, the work includes 3 appendixes.

ВСТУП	8
1. ДОСЛІДЖЕННЯ І АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ. ПОСТАНОВКА ЗАДАЧІ.....	11
1.1 Дослідження та аналіз предметної галузі	11
1.1.1 Короткий опис роботи	11
1.1.2 Аналіз існуючих рішень	13
1.1.3 Попередній аналіз вимог	19
1.2 Постановка завдання.....	20
2 РОЗРОБКА ПРОЕКТУ АВТОМАТИЗОВАНОЇ СИСТЕМИ	22
2.1 Ескізний проект	22
2.1.1 Розробка контекстної діаграми.....	22
2.1.2 Діаграма варіантів використання	23
2.1.3 Специфікація варіантів використання	25
2.1.4 Діаграми діяльності.....	29
2.1.5 Графічні інтерфейси користувачів	32
2.2 Технічний проект	33
2.2.1 Статична модель ПЗ.....	33
2.2.2 Динамічна модель ПО.	35
2.3 Робочий проект.....	37
2.3.1 Вибір мови та середовища розробки.....	37
2.3.2 Діаграми компонентів.....	42
2.3.3 Діаграми розміщення.....	43
3. РЕАЛІЗАЦІЯ ПРОЕКТУ АВТОМАТИЗОВАНОЇ СИСТЕМИ	45

3.1 Реалізація класів і кодування	45
3.1.2 Графічний інтерфейс користувача.	51
3.2 Тестування функціональності і користувацького інтерфейсу	52
4 Охорона праці.....	54
4.1 Вступ.....	54
4.2 Аналіз шкідливих на небезпечних факторів	55
4.3 Розробка заходів щодо зменшення впливу шкідливих та небезпечних факторів при роботі з персональним комп'ютером	57
4.3.1 Заходи щодо зменшення впливу.....	57
4.3.2 Техніка безпеки при роботі з комп'ютером	60
ВИСНОВКИ.....	65
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	67
ДОДАТОК А – Технічне завдання	69
ДОДАТОК Б – Інструкція з використання	73
ДОДАТОК В – Текст програми	77

ВСТУП

Профіль ринку нерухомості суттєво змінився за роки незалежності України. Спочатку в оренду, переважно, здавали приміщення, що вдалося приватизувати. Мали небагато об'єктів та використовували безальтернативні паперові видання для пошуку орендарів, вели на сторінках блокнотів облік витрат та надходження коштів за оренду, а для зв'язку використовували дротові телефони.

Пізніше з'явилися підприємства, що спеціалізувалися на оренді приміщень різноманітного використання. Об'єкти оренди знаходилися на одній території, в одному будинку. Спеціалізовані підприємства спочатку викупували дрібні об'єкти у окремих орендодавців та модернізували їх, дещо пізніше почалось будівництво об'єктів для здачі в оренду. З'явилися торгівельні та бізнес-центри, складські комплекси, в обіг потрапив термін девелопмент. Виникла потреба в спеціалістах по роботі з орендарями, що створило професію менеджера з оренди приміщень. Фактично – окремий випадок менеджера по роботі з клієнтами, що виконує нерідко також певні функції обліку.

Збільшення кількості орендарів з одного боку, певний розвиток технологій з іншого боку створили потребу та можливість створення інформаційних систем, спрямованих на інформаційну підтримку менеджера з оренди приміщень. Стало простіше зберігати та оновлювати інформацію про клієнтів, повідомляти про нарахування та вести облік сплати за цими нарахуваннями. Інформаційні системи підтримують складання типових договорів, розсилання платіжних вимог електронною поштою, зберігання та оновлення контактів орендарів.

Фактично подібні системи є адаптованим інтерфейсом користувача, що працюють з представленнями даних з облікової системи підприємства.

Звичайно перші системи були представлені у вигляді десктопних додатків або розширень конфігурацій облікових систем на кшталт 1С:Підприємство або аналогічних. Така організація вимагала окремого робочого місця менеджера з оренди приміщень, обладнаного персональним комп'ютером, що підключений

до локальної мережі підприємства та глобальної мережі Інтернет. Певний час така організація роботи була безальтернативною з причини технічних обмежень та стану розвитку комп'ютерної техніки. Пізніше подібні системи отримали web-інтерфейси: працювати стало дещо простіше, використовуючи підключення з будь-якого комп'ютера та ноутбука лише знаючи атрибути доступу до системи.

Сьогодні розвинені системи для девелоперів також створені у вигляді десктопних або web-додатків з розвиненим функціоналом, що дозволяє вести детальний облік та відстежувати десятки різних показників.

Виглядає це насправді доречно та професійна, але у більшості випадків такий функціонал виявляється надмірним, не використовується та ускладнює використання обмеженого переліку дійсно потрібних менеджеру з оренди функцій.

З оглядом на таку ситуацію дипломна робота ставить мету проаналізувати та виділити дійсно потрібні у повсякденній роботі менеджера з оренди функції, спроектувати та реалізувати мобільний додаток, що виконує інформаційну підтримку менеджера з оренди приміщень.

Реалізація системи у формі мобільного додатку дозволить користувачеві використовувати функції системи у будь-якому місці, мати доступ та користуватися інформацією з екрана смартфона, інтегрувати певні функції системи з операційною системою смартфона. Наприклад, зручними виявляються можливості надсилання листів та здійснення дзвінків безпосередньо з використанням назначених додатків операційної системи. Також перевагу буде мати, наприклад, функція нагадувань. З одного боку, вона інтегрована із сутностями інформаційної системи, з другого боку, нагадування з'являються на екрані смартфона, що набагато зручніше, аніж нагадування у операційній системі десктопного комп'ютера. Таким чином, мобільний додаток надає оперативний доступ до потрібної в роботі інформації та усуває необхідність використання з цією метою стаціонарного комп'ютера або ноутбука, що досить часто не знаходяться "під рукою" спеціаліста.

Мета проекту. Метою роботи є системний аналіз робочих завдань менеджера з оренди приміщень, конструювання, проектування, вивчення технологій реалізації та реалізація мобільного додатку, що створює інформаційну підтримку роботи менеджера з оренди: забезпечує оперативний доступ до необхідної облікової інформації, контактної інформації орендарів, дозволяє змінювати певну інформацію з використанням мобільного інтерфейсу.

Моделювання, проектування, конструювання. Для розробки проекту мобільного додатку використано мову UML. Результати етапів проектування представлено у вигляді відповідних діаграм мовою UML.

Реалізація проекту. Спроектований мобільний додаток реалізовано з використанням мови програмування Java та середовища розробки Android Studio. Для забезпечення безперебійного доступу до даних використовується СУБД MySQL, розміщена на віддаленому сервері, та SQLite для зберігання обмеженої проекції даних безпосередньо у мобільному терміналі користувача.

Тестування додатку реалізовано стандартними методами Java.

Практична цінність проекту. Спроектований та реалізований додаток суттєво надає необхідну інформаційну підтримку менеджеру з оренди й є зручною альтернативою розгалуженим системам, що працюють у вигляді десктопних або web-додатків. Мобільність усуває потребу в постійному робочому місці, що повністю відповідає сучасним трендам віддаленою роботи та відсутності великих офісів.

Загальний обсяг основної частини складає 59 сторінок, включаючи 18 рисунків та 3 таблиці. Також пояснювальна записка включає 3 додатки, що містять технічну інформацію.

1. ДОСЛІДЖЕННЯ І АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ. ПОСТАНОВКА ЗАДАЧІ.

1.1 Дослідження та аналіз предметної галузі

1.1.1 Короткий опис роботи

В сучасне життя вже декілька десятиліть увійшло та закріпилось поняття девелопменту нерухомості. На оточують торгівельні та бізнес – центри, апартамент-готелі, комплекси апартаментів у моря, що належать низці власників, але також придбані для здачі в оренду на різні терміни.

Все це є прикладами, коли певна кількість об'єктів оренди згуртована о одному великому об'єкті. Власне, торгівельні площі ринка також є прикладом управління орендою, але замість закритих приміщень справа йде про відкриті ділянки, столи, контейнери або щось інше.

Усі ці приклади поєднує велика кількість об'єктів оренди, що знаходяться на одній закритій або відкритій території. Та, відповідно, велика кількість орендарів, з якими укладаються договори на різний термін. В рамках цих договорів орендарі сплачують послуги оренди, компенсують вартість комунальних послуг, зокрема електроенергії та водопостачання, іноді сплачують різноманітні збори, наприклад, ринковий, тощо.

У випадку великої кількості об'єктів, що здаються в оренду, власник або генеральний директор навряд чи будуть безпосередньо займатися роботою з орендарями. Саме тому з'являються, як правило, позиції менеджера з оренди приміщень та бухгалтера, що веде відповідний облік.

До робочих задач менеджера з оренди входить комунікація з поточними та потенціальними орендарями, пошук серед вільних об'єктів найбільш відповідних для нових орендарів, укладання договорів, контроль своєчасної оплати за договорами, вирушення інших питань, що виникають у орендарів.

Велика кількість орендарів, регулярно змінні тарифи на оплату комунальних послуг, купа інших питань роблять облік взаємовідносин з орендарями “на папірці” надскладним та незручним.

Тому виникають розширення облікових систем, що спрямовані на облік взаємовідносин з орендарями. Фактично мова йде про CRM – систему, що отримує певну інформацію з облікової системи та групує інформацію по договорах оренди. Саме договір оренди можна вважати обліковою одиницею, навколо якої групується інша необхідна інформація: контакти орендаря, заборгованість за послуги, строк дії договору, площа об'єкту, що зданий в оренду, тощо. А діяльність менеджера оренди фокусується навколо низки питань навколо договору та сплати зобов'язань за ним. Безумовно окрім цього існує окрема група завдань, що групуються навколо укладання нових договорів.

З вище наведених міркувань можна зробити висновок, що перш за все менеджер з оренди потребує інформаційної підтримки: отримання робочої інформації, що автоматично підтримується в актуальному вигляді. Актуалізація інформації означає підтримання в актуальному стані інформації по заборгованостям, строкам дії договорів, контактам орендарів, тощо. Для частини цієї інформації менеджер з оренди є джерелом й його обов'язком є підтримання інформації в актуальному стані для власного використання та використання іншими співробітниками. Для іншої частини інформації менеджер оренди є споживачем, а актуалізація інформації проводиться співробітниками, наприклад, бухгалтерії.

Позиція менеджера з оренди приміщень означає певну мобільність тому, що зазвичай комунікація з новими та поточними орендарями відбувається на території відповідних об'єктів оренди, а не в офісі орендодавця. Тому інформаційна система, виконана в форматі мобільного додатку, має переваги над системами, що потребують роботи за монітором персонального комп'ютера або ноутбука.

Підводячи підсумки системного аналізу робочих завдань менеджера з оренди приміщень, можна зробити висновок, що робота менеджера з оренди приміщень потребує автоматизації у вигляді інформаційного додатку, що виконаний у вигляді мобільного додатку та забезпечує можливості споживання

актуальної робочої інформації, а також оновлення певної інформації через мобільний інтерфейс.

1.1.2 Аналіз існуючих рішень

Зосередившись на оренді комерційної нерухомості в рамках одного об'єкта, розглянемо два комерційних рішення, що експлуатуються підприємствами України. Основною метою розгляду є вивчення функціональності існуючих рішень для формулювання вимог до програмного засобу, що має бути спроектований та створений.

Першим з рішень, що розглядається, є розробка Softpro: оренда +. Програмний комплекс позиціонується як універсальне веб-рішення для менеджерів та бухгалтерів, власників та управлінців комерційної нерухомості. Основними користувачами можуть бути торгівельні та бізнес-центри, агенції нерухомості, склади, ринки.

Основними розділами веб-додатку, що розглядається, є наступні:

Управління договорами. Після внесення інформації за договором програма сформує схему оплати за приміщення і супутні послуги. Простежить за нарахуваннями, платежами, боргами та депозитами.

Облік орендарів і орендодавців. З кожним новим орендарем/орендодавцем надходить інформація про супутні договори, контактні особи, реквізити для оплати. Звільняючи від необхідності вводити повторно, програма вбудовує такі дані в шаблони договорів так само, як і в решту супутніх реєстрів.

Інтерактивна план-схема. Налаштувати геометрію під потреби клієнта можна на інтерактивній план-схемі, попередньо завантаживши план приміщення.

Управління фінансами. Дізнавайтеся актуальну інформацію про нарахування, платежі, грошові надходження і заборгованості на онлайн-платформі.

Ведення комунальних послуг та лічильників. Назва підсистеми нагадує про необхідність обліку видатків на електрику, послуги з охорони, вивезення сміття

та інших індивідуальних витрат. Програма розподілить суми між орендарями та підготує для них протоколи з відповідними виписками.

Звіти. За цей день, тиждень, місяць або рік — власнику бізнесу потрібні звіти. Програма автоматично створює їх в Excel з можливістю конвертувати PDF, завантажити на комп'ютер і надіслати на email.

Синхронізація. Якщо ви користуєтеся системами бухгалтерського обліку, такими як 1С: Бухгалтерія, ми синхронізуємо виписки за нарахуваннями і платежами. Цими процесами ви також можете керувати в ручному режимі.

Розмежування прав і безпека. Щоб зберегти корпоративну таємницю, достатньо розмежувати доступ до інформації. Програма передбачає налаштування рівнів доступу до реєстрів на онлайн-платформі та фіксує кожну зміну.

З опису функціональних можливостей програмного комплексу можна бачити, що вони дещо виходять за рамки компетенцій адміністратора об'єкту нерухомості. Тому саме для адміністратора така функціональність є надмірною. З іншої сторони, додаток базується на web-інтерфейсі, що обмежує можливості роботи з ним з використанням смартфона — екран смартфона виявляється замалим навіть за умови адаптивної верстки.

В якості іншого рішення розглянемо продукт компанії “Торгсофт”.

Програмний продукт "Торгсофт: Оренда та управління нерухомістю" призначений для власників комерційної нерухомості, керуючих компаній, девелоперських структур. Продукт дозволяє підвищити ефективність бізнесу компаній за рахунок автоматизації процесів щодо ведення реєстрового обліку об'єктів нерухомості, управління договорами оренди та розрахунками з орендарями, експлуатації об'єктів нерухомості. Програма забезпечує вирішення завдань бухгалтерського, управлінського, юридичного та адміністративного обліку.

Програма дозволяє ефективно управляти нерухомістю різних типів: торговими та офісними центрами, земельними ділянками, виставковими площами, складами, ринками тощо.

Продукт розроблений на основі типової конфігурації "Бухгалтерія підприємства", ред. 3.0 забезпечує масштабованість, відкритість, простоту адміністрування та конфігурування. Програмний комплекс забезпечує можливість роботи у WEB-інтерфейсі, що робить роботу користувачів більш інтуїтивно зрозумілою та дозволяє працювати віддалено.

Короткий перелік функціональності додатку є таким:

1. Управління реєстром об'єктів нерухомості.

- a. Ведення ієрархічного довідника об'єктів нерухомості з широким набором характеристик та можливістю додавання довільних характеристик;
- b. Зручне та просте створення графічних планів розміщення об'єктів;
- c. Автоматичне створення об'єктів нерухомості у базі даних на основі графічних планів розміщення;
- d. Оперативний перегляд розміщення об'єкта на плані;
- e. Реєстрація довільних статусів об'єктів нерухомості на заданий період – вільний, у резерві, на ремонті, орендується тощо;
- f. Формування реєстру об'єктів нерухомості з можливістю гнучкого угруповання та відбору за різними характеристиками;
- g. формування діаграми станів об'єктів нерухомості;
- h. Формування графічної звітності на основі планів розміщення об'єктів з колірним виділенням об'єктів залежно від статусу та можливістю довільного відбору (за вільними площами, за площами у ремонті, резерві тощо);

- i. Можливість прив'язки об'єктів нерухомості до цінових категорій, реєстрація цін на послуги з прив'язкою до цінових категорій, збереження історії цін.
2. Управління договорами оренди.
- a. Облік первинних документів за договорами оренди (Укладання договору оренди, Додаткова угода, Розірвання договору оренди, Акт приймання-передачі орендованих приміщень, Акт приймання-повернення орендованих приміщень);
 - b. Поділ послуг на постійну та змінну частини;
 - c. Гнучке управління способами нарахування, ставками та тарифами на послуги, зберігання історії зміни ставок та тарифів;
 - d. Роздільне призначення термінів оплати та параметрів нарахування пені для послуг постійної та змінної частини, а також для конкретних послуг;
 - e. Друк графічних планів з виділенням об'єктів, що здаються в оренду, як додатки до договорів;
 - f. Зберігання списку телефонних номерів та їх прив'язка до договорів орендарів;
 - g. Можливість багаторазової зміни ставок на послуги в межах одного договору;
 - h. Можливість обліку оренди із перервами;
 - i. використання спеціальних курсів валют (фіксований курс, курс ЦП плюс заданий відсоток, валютний коридор);
 - j. Можливість управління договорами суборенди;
 - k. Формування графічної звітності на основі планів розміщення об'єктів з колірним виділенням об'єктів залежно від їх статусу та можливістю налаштування складу інформації, що виводиться (за орендарями, умовами договорів, характеристиками об'єктів тощо);

1. Перевірка зайнятості та контроль перевищення загальної площі об'єкта під час створення нового договору оренди;
 - m. Можливість здавати один об'єкт до найму декільком орендарям;
 - n. Групова пролонгація та розірвання договорів оренди з можливістю відбору за заданими характеристиками;
3. Управління взаєморозрахунками з оренди
- a. Гнучке настроювання деталізації взаєморозрахунків з орендарями: за договорами, періодами оплати, об'єктами, послугами, рахунками на оплату;
 - b. Облік первинних документів з оренди: рахунків, актів надання послуг, рахунків-фактур. Можливість друку документів, як у зведеній формі, так і з деталізацією щодо об'єктів та/або послуг;
 - c. Автоматичне формування рахунків на оплату та актів про надання послуг з можливістю довільного відбору за орендарями, послугами, об'єктами тощо, печатку сформованих документів та надсилання орендарям електронною поштою;
 - d. Розрахунок оренди за неповний період. Можливість розрахунку як за фактичною кількістю днів у зазначеному періоді, так і на підставі "середнього місяця";
 - e. Розрахунок оплати за тарифіковані послуги з урахуванням показань лічильників, можливість розподілу показань лічильника з кількох об'єктів чи орендарів;
4. Управління експлуатацією об'єктів нерухомості
- a. Облік конструктивних елементів та інженерних систем об'єктів нерухомості, що підлягають технічному обслуговуванню та ремонту;
 - b. Планування заходів з технічного обслуговування об'єктів нерухомості (плановий та поточний ремонт, регламентні роботи тощо)

5. Сервісні можливості

- a. Автоматичне створення нагадувань та повідомлень з різних подій (наприклад, нагадування про наближення дати закінчення договору оренди);

6. Інтеграція ведення обліку

- a. Облік торгових операцій
- b. Облік агентських договорів. Автоматизовано облік агентських послуг із боку агента (надання послуг від імені, але з допомогою принципала) і з боку принципала (надання послуг через агента).
- c. Ведення усіх розділів бухгалтерського та податкового обліку згідно діючу законодавства.
- d. Облік непрямих доходів від оренди нерухомості.

Як можна бачити з наведеного опису, функціональність другої розглянутої системи таке є досить розгалуженою та включає усі аспекти обліку роботи з орендою нерухомості. Програмний комплекс базується на десктопному або web-інтерфейсі, тому також ускладнено його використання з мобільного телефону.

Обидва комерційні рішення, що були оглянуті, призначені для використання майже усім колективом підприємства, не обмежуючись адміністратором. Окрім безпосередньо взаємовідносин з орендарями програмні комплекси пропонують можливості ведення документообігу, ремонту та модернізації об'єкту нерухомості, розрахунків за комунальними послугами. Другий з програмних продуктів дозволяє вести увесь бухгалтерський та податковий облік, що вимагається діючим законодавством. У той же час розглянуті програмні комплекси мають обмежені можливості мобільного використання та пропонують надлишковий, занадто ускладнений інтерфейс користувача коли мова йде безпосередньо про функціональні обов'язки адміністратора оренди приміщень.

Говорячи безпосередньо про адміністратора оренди приміщень, можна виділити такі його функціональні обов'язки, що пов'язані з певною функціональністю програмного комплексу.

1. Підбір приміщення для оренди.
2. Укладання договору оренди, в тому числі з розділом, враховуючим відшкодування комунальних послуг.
3. Комунікація з орендарями з поточних питань.
4. Контроль оплати оренди приміщень за укладеними договорами.
5. Підтримування складу матеріального забезпечення, потрібного за укладеними договорами оренди приміщень.

1.1.3 Попередній аналіз вимог

Складемо перелік вимог для подальшого аналізу системи на основі огляду предметної області, зробленого в попередньому розділі.

Процесно-орієнтовані функціональні вимоги

Перегляд списку об'єктів оренди цілком та за виділеними групами;

Відбір об'єктів оренди за певними властивостями з використанням фільтрів;

Перегляд строків закінчення договорів з метою планування нової задачі в оренду або пролонгації договору;

Перегляд заборгованості орендарів: в цілому за договорами та за окремими послугами, що надаються в рамках договорів;

Перегляд поточних справ з визначеними дедлайнами, можливість встановлення нагадування.

Інформаційно-орієнтовані функціональні вимоги

Перегляд списку орендарів, пошук контактної інформації.

Нефункціональні вимоги:

Система повинна працювати та надавати зручний для використання інтерфейс на екрані мобільного телефону.

Система повинна використовувати єдину базу даних, що дозволить інтегрувати додаток з обліковою системою підприємства – орендодавця.

1.2 Постановка завдання

Постановка завдання на реалізацію - це опис способу реалізації вихідних вимог, технічного завдання, архітектурного рішення, виклад вимог до влаштування спроектованого рішення (на цьому етапі вихідні вимоги вже опрацьовані).

Постановка завдання на реалізацію відбувається після визначення архітектури системи, як показано на рисунку.

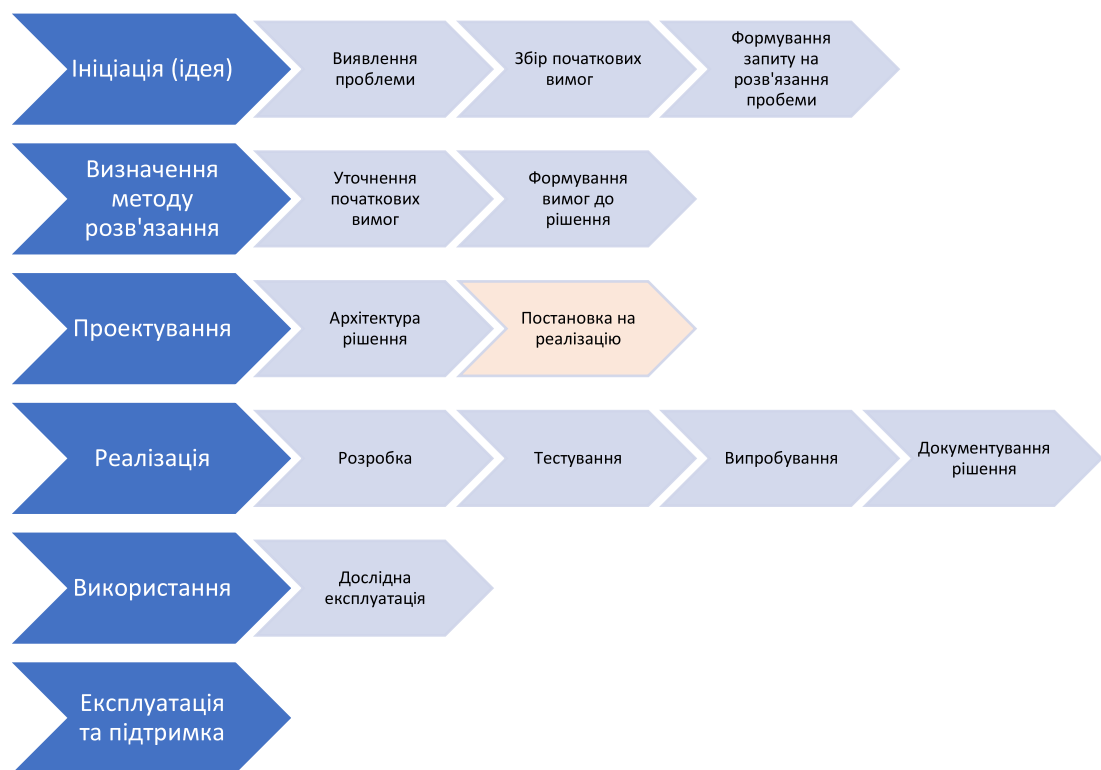


Рисунок 1 - Процес реалізації ІТ – рішення.

Тому визначимо архітектуру систему, що розроблюється, та виконаємо постановку на реалізацію.

Система складається з власне мобільного додатку, що реалізує визначені вимоги до системи та віддаленої бази даних, призначеної для зберігання даних щодо об'єктів оренди, орендарів, взаєморозрахунків, тощо. З одного боку, віддалена база даних надає відомий API для отримання необхідних даних, з

іншого боку, може бути створений мікро-сервіс, який реалізує синхронізацію бази даних додатка з інформаційною системою підприємства. Використання СУБД основної інформаційно – облікової системи підприємства не є оптимальним рішенням тому, що з одного боку, така БД може мати недокументований розробниками системи формат та не мати API для сторонніх додатків, з іншого боку, БД центральної облікової системи зберігає суттєво більше обсяги інформації, аніж потрібно. Ще одним фактором на користь використання декількох БД є безпека основного сховища даних за рахунок забезпечення одностороннього обміну (лише вивантаження даних) з основного інформаційного сховища.

Таким чином, розглядаємо дві компоненти системи: мобільний додаток Android та СУБД.

Мобільний додаток Android шляхом запитів до СУБД та їх обробки формує усі необхідні адміністраторові об'єкту оренди звіти та надає усі необхідні представлення інформації. В термінах патерну MVC формуються усі необхідні представлення – views. Окрім цього, мобільний додаток реалізує усі необхідні CRUD – операції за умови, що такі операції потрібні в системі. Прикладом CRUD – операції може бути коригування контактної інформації орендаря.

Також мобільний додаток реалізує додаткову функціональність для конкретного користувача: нагадування, тощо.

СУБД має бути розміщена на віддаленому сервері та надавати захищений API для отримання та збереження інформації.

Клієнтський мобільний додаток буде реалізовано з використанням Java у інтегрованому середовищі розробки Android Studio.

В якості СУБД буде використано MySQL з розміщенням у Docker контейнері. Для потреб тестування контейнер може бути розташований локально, в подальшому без змін перенесений на віддалений сервер.

2 РОЗРОБКА ПРОЕКТУ АВТОМАТИЗОВАНОЇ СИСТЕМИ

2.1 Ескізний проект

2.1.1 Розробка контекстної діаграми

Контекстна діаграма насамперед дозволяє швидко, коротко і точно описати призначення і межі системи, виявити та усунути колективні розбіжності у тому розумінні, показати і домовитися про її масштабі.

Друге її призначення - служити джерелом для швидкої генерації первинного набору системних функціональних вимог при необхідності проектування системи не «зверху-вниз», від бізнес-моделі, бізнес-вимог, моделі діяльності організації, вимог зацікавлених осіб, моделі використання, як пропонує нам системна інженерія, а «з середини».

Контекстна діаграма належить до категорії діаграм, що описують систему лише на рівні «чорного ящика» — лише зовнішні властивості (потоки даних).

Контекстна діаграма містить 3 основні компоненти:

1. Проектований об'єкт (наприклад, система).
2. Взаємодіючі з проєктованим об'єктом елементи оточення (групи користувачів, суміжні системи).
3. Потоки даних (вихідні та вхідні).

Контекстна діаграма проекту, що розробляється, подана на рисунку .



Рисунок 2 – Контекстна діаграма.

2.1.2 Діаграма варіантів використання

Діаграми використання були запропоновані Іваром Якобсоном у їх нинішній графічній формі ще 1986 року.

Діаграми використання є, безумовно, найстабільнішим елементом UML — вони не змінювалися вже двадцять років, фактично, набули закінченої форми задовго до появи мови. Одночасно ці діаграми мають найпростішу нотацію: лише два основних типи сутностей (дійові особи та варіанти використання) та три типи відносин (залежності, асоціації, узагальнення).

З синтаксичної точки зору дійова особа — це стереотип класифікатора, який позначається спеціальним позначкою. Для дійової особи вказується лише ім'я, що ідентифікує його у системі. Семантично дійова особа — це безліч логічно взаємопов'язаних ролей.

З прагматичного погляду головним є те, що дійові особи знаходяться поза проєктованою системою (або розглянутою частиною системи).

Виділення варіантів використання — ключ до всього подальшого моделювання. На цьому етапі визначається функціональність системи, тобто, що вона має робити.

Нотація для варіанта використання дуже мізерна — це просто ім'я, поміщене в овал (або поміщене під овалом — такий варіант теж допустимо). Інакше кажучи, функції, виконувані системою, лише на рівні моделювання використання не відкриваються — їм лише даються імена.

Семантично варіант використання (use case) — це опис безлічі можливих послідовностей дій (подій), що призводять до значущого для дійової особи результату.

Кожна конкретна послідовність дій називається сценарієм.

Третім типом сутності, що застосовується на діаграмі використання, є примітка. Зауважимо, що примітки є важливим засобом UML. На відміну від більшості мов програмування примітки в UML синтаксично оформлені за

допомогою спеціальної нотації і виступають на тих самих правах, що й інші сутності.

Зазвичай цілком ясно, що знаходиться всередині моделі, що моделюється, а що зовні. Якщо це чомусь неясно, або потрібно збільшити наочність діаграм, то можна скористатися спеціальною конструкцією, яка називається "межі системи".

Межі системи (system boundary) – це графічний коментар у формі прямокутної рамки, що застосовується на діаграмах використання та відокремлює внутрішню частину системи від зовнішнього оточення. Внутрішня частина, що виділяється межами, має в UML конкретну назву - суб'єкт.

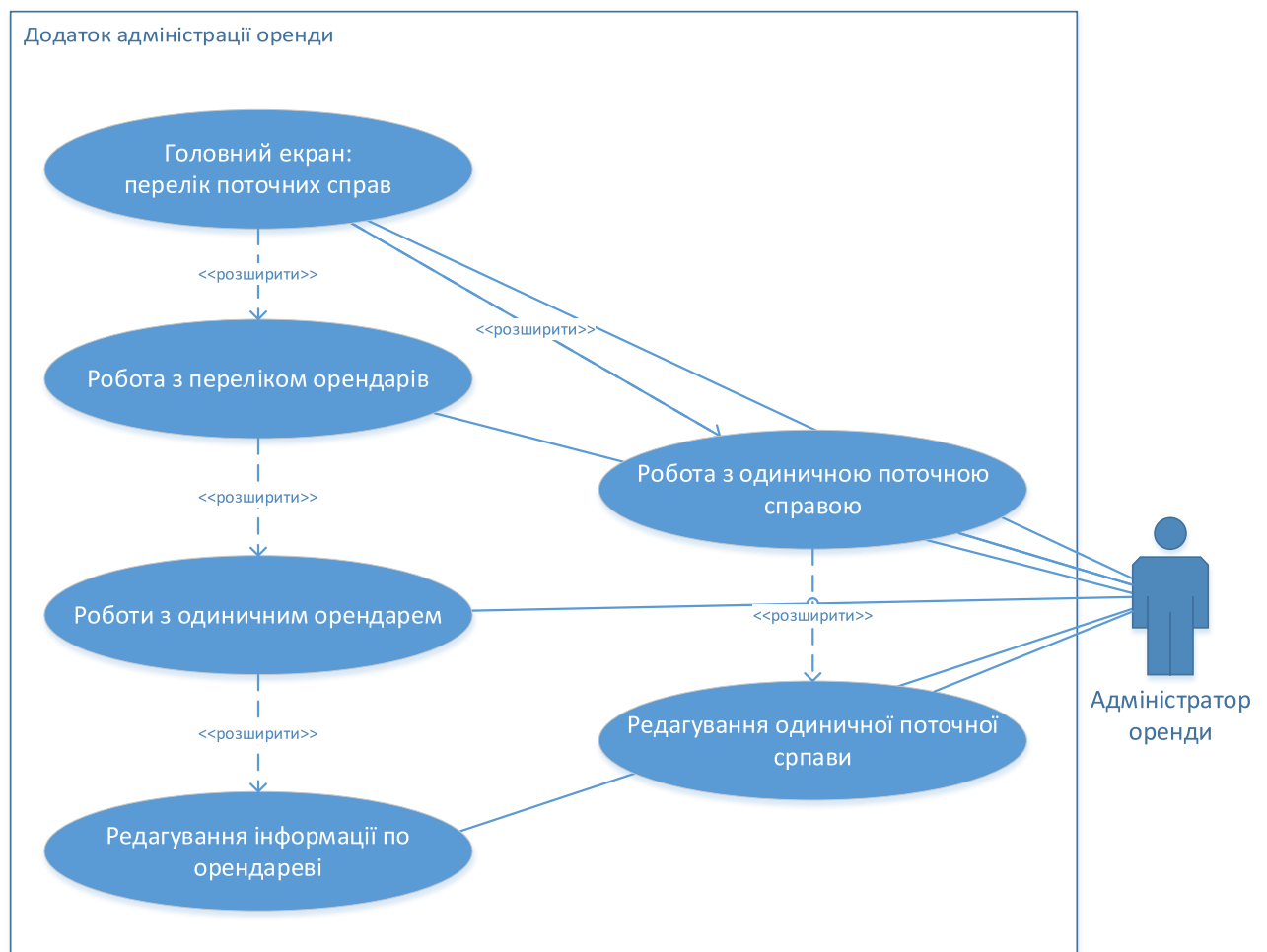


Рисунок 3 - Діаграма варіантів використання.

Діаграма використання демонструє перегляд користувачем системи переліків двох основних типів сутностей: поточних справ та орендарів.

Розширенням сценаріїв використання є перегляд окремих екземплярів сутностей – на екрані смартфона обмеженого розміру недоцільно відображати одразу багато інформації щодо окремої сутності, а також виконання CRUD операцій щодо окремих екземплярів сутностей системи.

2.1.3 Специфікація варіантів використання

Загальноприйнята структура опису/специфікації варіанта використання (ВВ):

Ім'я ВВ;

Ідентифікатор;

Короткий опис;

Актори:

Головні та другорядні актори;

Ініціюючі та беруть участь актори.

Передумови;

Основний потік;

Альтернативні потоки;

Постумови;

Спеціальні вимоги.

Розглянемо кожен із розділів докладніше.

Ім'я ВВ – коротке та ємне найменування, щоб чітко та однозначно розуміти призначення ВІ.

Ідентифікатор ВВ – незмінний, на відміну імені.

Короткий опис ВВ – абзац у 2-3 рядки, у якому викладено мету ВВ.

Актори, залучені до ВВ:

Головні актори – ініціюють ВІ;

Інші актори – взаємодіють з ВВ під час його виконання.

Передумови – умови, які мають бути виконані на початок ВВ; обмеження стану системи.

Головний (основний) потік (ГП) – діалог актора та системи; завжди починається з дії головного актора. Дії записуються в активній заставі з прямим порядком слів, наприклад, "Користувач вибирає рядок у таблиці А". Маленьке просте розгалуження основного потоку припустимо, так як може скоротити кількість альтернативних потоків. Корисні питання для уточнення формулювання дії: Хто саме? Що саме? Коли саме? Де саме?

Альтернативні потоки (АП) – обробляють помилки, розгалуження, переривання, виключення основного потоку. Можуть бути описані окремо або наприкінці опису ВВ. Альтернативних потоків може не бути. Для пошуку АП потрібно на кожному кроці ОП шукати можливі альтернативні дії, помилки, а також переривання, які можуть статися як на цьому кроці ОП, так і будь-якої миті. Кількість альтернативних потоків необхідно звести до мінімуму.

Постумови - стан системи після закінчення ВВ; повинні виконуватись обов'язково. Якщо АП повертається в ОП, його постумови додаються до постумов ОП.

Спеціальні вимоги – текстовий опис вимог, які не відображені у сценаріях ВВ.

Якщо наповнення якогось із розділів опису ВІ відсутнє в ньому пишуть «Ні» – це показує, що питання було опрацьовано.

Створимо специфікацію варіантів використання за викладеним вище патерном.

Ім'я ВВ: початковий екран додатку.

Ідентифікатор: **mainScreen**

Короткий опис: початковий екран додатку. На цей екран користувач потрапляє безпосередньо після запуску додатку. На екрані представлено перелік запланованих користувачем у контексті управління орендою задач. За допомогою меню додатку з головного екрану відбувається перехід до екрану операцій з переліком орендарів.

Актори:

Головні та другорядні актори: основний (єдиний) актор – користувач системи, адміністратор оренди приміщень.

Ініціюючі та беруть участь актори: ініціюючим актором є головний актор, адміністратор оренди приміщень.

Передумови: немає.

Основний потік: головний екран з'являється після запуску додатку користувачем. З головного екрану відбувається перехід до переліку орендарів з можливостями необхідної фільтрації або до редагування однієї з задач, що представлені на головному екрані. На головному екрані можна здійснити вибір задачі для редагування, створити нову задачу, помітити як виконану або видалити одну з існуючих задач.

Альтернативні потоки: фоновною задачею для цього та подальших варіантів використання є синхронізація локальної БД додатку з центральною БД, що розташована на сервері.

Постумови: немає.

Спеціальні вимоги: немає.

Ім'я ВВ: екран перегляду та редагування окремої задачі.

Ідентифікатор: **singleTaskScreen**

Короткий опис: надає інтерфейс редагування та запису окремої задачі, пов'язаної з процесом здачі в оренду приміщень.

Актори:

Головні та другорядні актори: основний (єдиний) актор – користувач системи, адміністратор оренди приміщень.

Ініціюючі та беруть участь актори: ініціюючим актором є головний актор, адміністратор оренди приміщень.

Передумови: вибір задачі для редагування на головному екрані додатку.

Основний потік: редагування та збереження окремої задачі.

Альтернативні потоки: див. вище.

Постумови: надсилання оновлення до БД.

Спеціальні вимоги: немає.

Ім'я ВВ: екран роботи з переліком орендарів.

Ідентифікатор: **listOfTenantsScreen**

Короткий опис: екран перегляду переліка орендарів та відбору підмножини орендарів за певним критерієм. З екрану також здійснюється пошук орендаря за назвою, створення нового та видалення одного з існуючих орендарів.

Актори:

Головні та другорядні актори: основний (єдиний) актор – користувач системи, адміністратор оренди приміщень.

Ініціюючі та беруть участь актори: ініціюючим актором є головний актор, адміністратор оренди приміщень.

Передумови: вибір відповідного пункту меню додатку.

Основний потік: маніпулювання переліок орендарів. Необхідні запити та фільтрація відбуваються в основному потоці без застосування альтернативних потоків.

Альтернативні потоки: див. вище..

Постумови: збереження змін – результату CRUD операцій – до БД.

Спеціальні вимоги: немає.

Ім'я ВВ: екран роботи з окремим орендарем.

Ідентифікатор: **singleTenantScreen**

Короткий опис: призначений для додавання/редагування інформації по орендареві, перегляду заборгованості за основною та додатковою послугами, відправки копії договору та даних про заборгованість електронною поштою.

Актори:

Головні та другорядні актори: основний (єдиний) актор – користувач системи, адміністратор оренди приміщень.

Ініціюючі та беруть участь актори: ініціюючим актором є головний актор, адміністратор оренди приміщень.

Передумови: вибір орендаря із списку орендарів.

Основний потік: перегляд та редагування інформації за окремим орендарем.

Альтернативні потоки: див. вище.

Додаткові потоки використовуються для уникнення “фрізів” основного інтерфейсу додатку. Перший з додаткових потоків передає інформацію до основного потоку. Другий виконується та завершається цілком незалежно від основного потоку сценарію використання. Може бути завершений й після завершення основного потоку.

Постумови: немає.

Спеціальні вимоги: немає.

2.1.4 Діаграми діяльності

При моделюванні проектованої системи часто необхідно деталізувати особливості алгоритмічної та логічної реалізації операцій, що виконуються системою. Зазвичай для цього використовуються блок-схеми.

У мові UML моделювання процесу виконання операцій використовуються діаграми діяльності (activity diagrams). Вони схожі на діаграми станів, тут також є стани і переходи, але стану використовуються для представлення дій. Тобто якщо уявити діаграму дій як графа, то вершини – це стану дії, а дуги – переходи з-поміж них.

Кожен стан на діаграмі діяльності відповідає виконанню деякої елементарної операції. Саме діаграми діяльності дозволяють реалізувати у мові UML особливості процедурного та синхронного управління. Метамодел ь UML надає для цього необхідні терміни та семантику. Основним напрямом використання діаграм діяльності є візуалізація особливостей реалізації операцій класів, коли необхідно уявити алгоритми їх виконання. При цьому кожен стан може бути виконанням операції деякого класу або її частини, дозволяючи використовувати діаграми діяльності для опису реакцій на внутрішні події системи.

Стан дії (action state) є спеціальним випадком стану з деякою вхідною дією і принаймні одним переходом, що виходить зі стану. Стан дії не може мати внутрішніх переходів, оскільки є елементарним. Дія може бути записана природною мовою, деяким псевдокодом або мовою програмування.

Діаграма діяльності для роботи з екземплярами сутності “Справи/задачі” представлена на рисунку

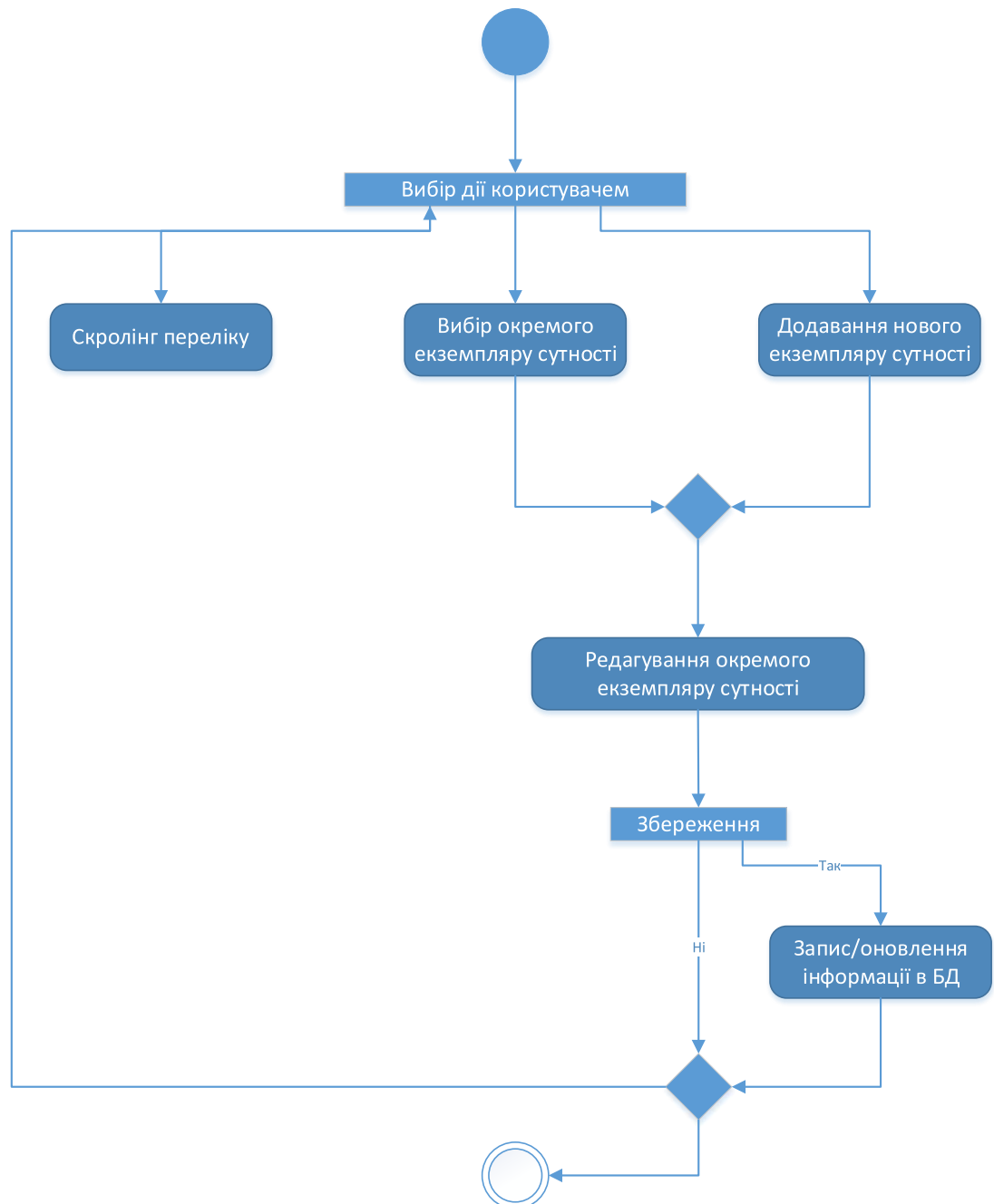


Рисунок 4 - Діаграма діяльності. Сценарій використання “Задачі”

Діаграма діяльності у випадку роботи з екземплярами сутності “Орендар” відрізняється перевіркою достатності інформації у локальній інформаційній базі та отриманням, за необхідності, додаткової інформації з центральної інформаційної бази.

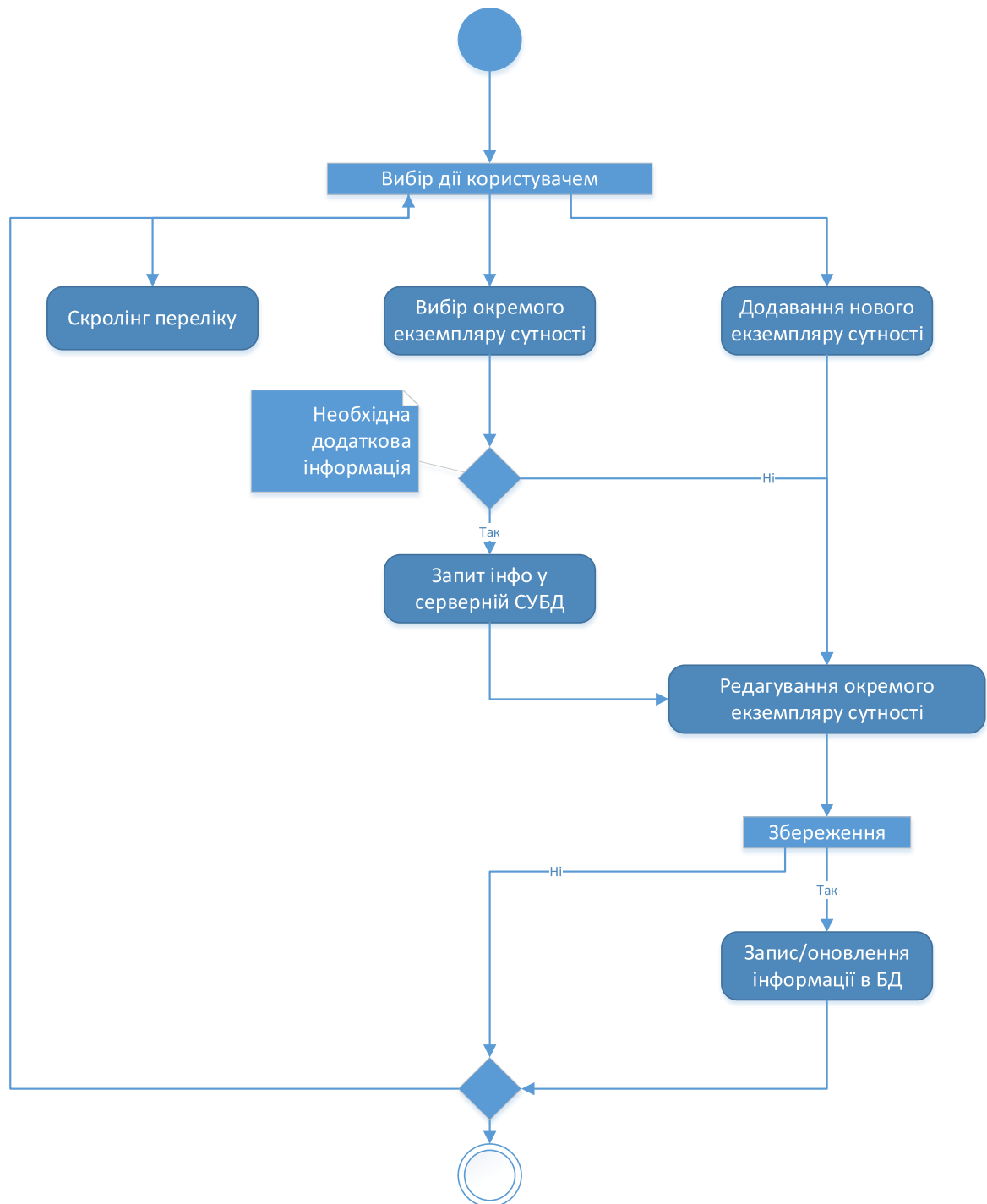


Рисунок 5 - Діаграма діяльності. Сценарій використання “Орендарі”

2.1.5 Графічні інтерфейси користувачів

Інтерфейс користувача (UI) — це «спосіб, яким ви виконуєте будь-яке завдання за допомогою будь-якого продукту, а саме дії, що вами вчиняються і те, що ви отримуєте у відповідь» (джерело: Джеф Раскін, «Інтерфейс. Нові напрямки в проектуванні комп'ютерних систем»).

Макети графічних інтерфейсів відповідають сценаріям використання додатку, що були визначені та описані в попередніх розділах. Отже, створюємо чотири основних макета, два з яких призначені для перегляду списку екземплярів сутностей “Задача” та “Орендар”, інші два – для перегляду та редагування екземплярів цих сутностей.

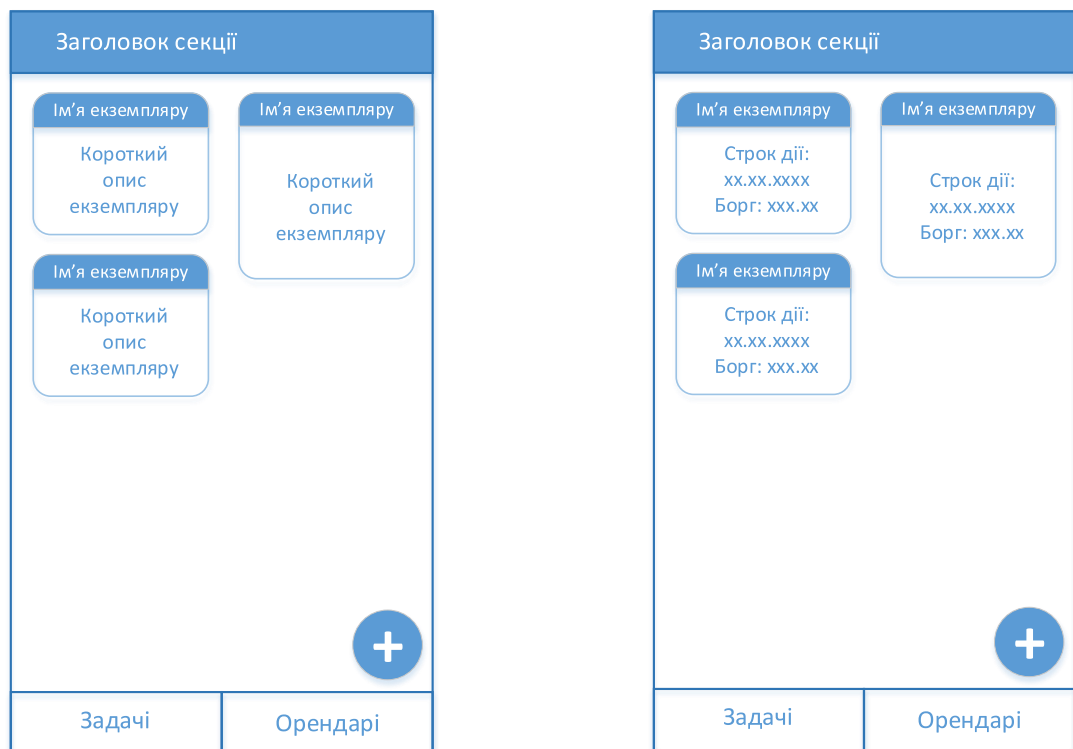


Рисунок 6 - Графічний інтерфейс перегляду списків.

Важливою особливістю інтерфейсу перегляду списків є знаходження найбільш важливої для користувача інформації безпосередньо на плашках, що представляють екземпляри сутностей. Це дозволяє переглядати інформацію, що потрібна в роботі користувача, без переходу до екрану, що представляє опис екземпляра сутності.

Редагування задачі		Інформація по орендареві	
Найменування задачі Текстове поле вводу		Орендар Текстове поле вводу	
Пов'язаний орендар Текстове поле вводу		Строк дії договору Поле вводу календар	
Дата виконання Поле вводу календар		Заборгованість За послуги оренди: xxxx.xx За інші оренди: xxxx.xx	
Опис Багатострокове текстове поле		Нотатки Багатострокове текстове поле	
Зберегти	Скасувати	Зберегти	Скасувати

Рисунок 7 - Графічний інтерфейс редагування екземплярів сутностей.

2.2 Технічний проект

2.2.1 Статична модель ПЗ

Діаграма класів визначає типи класів системи та різного роду статичні зв'язки, які існують між ними. На діаграмах класів зображуються також атрибути класів, операції класів та обмеження, що накладаються на зв'язки між класами. Вигляд та інтерпретація діаграми класів істотно залежить від точки зору (рівня абстракції): класи можуть представляти сутності предметної галузі (у процесі аналізу) або елементи програмної системи (у процесах проектування та реалізації).

Основними елементами є класи та зв'язки між ними. Класи характеризуються за допомогою атрибутів та операцій.

Атрибути описують характеристики об'єктів класу. Більшість об'єктів у класі отримують свою індивідуальність через відмінності в їх атрибутах та взаємозв'язку з іншими об'єктами. Однак, можливі об'єкти з ідентичними значеннями атрибутів та взаємозв'язків. Тобто. індивідуальність об'єктів визначається самим фактом їх існування, а чи не відмінностями у тому властивостях. Ім'я атрибута має бути унікальним у межах класу. Назва атрибута може містити його тип і значення за промовчанням.

Операція є функцією або перетворення. Операція може мати параметри та повертати значення.

Види зв'язків:

- асоціація
- агрегація
- успадкування.

Асоціація (association) – є відносини між екземплярами класів. Кожен кінець асоціації має кратність (синонім - потужність, ориг. - multiplicity), яка показує, скільки об'єктів, розташованих з відповідного кінця асоціації, може брати участь у цьому відношенні.

Асоціації може бути присвоєно ім'я. Як ім'я зазвичай вибирається дієслово або дієслівне словосполучення, що повідомляють сенс і призначення зв'язку. Також кінцях асоціації під кратністю може вказуватися ім'я ролі, тобто. яку роль виконують об'єкти, що перебувають з цього кінця асоціації.

Агрегація (aggregation) - це асоціація типу "ціле-частина". Агрегація в UML представляється як прямий з ромбом на кінці. Ромб на зв'язку вказує, який клас є агрегуючим (тобто «що складається з»); клас із протилежного кінця — агрегованим (тобто ті самі «частини»).

Композиція (composition) – це така агрегація, де об'єкти-частини що неспроможні існувати власними силами і знищуються при знищенні об'єкта агрегує класу. Композиція зображується так само, як асоціація, тільки ромб зафарбований.

Наслідування (inheritance) – це відношення типу «загальне-приватне». Дозволяє визначити таке відношення між класами, коли один клас має поведінку та структуру низки інших класів. При створенні похідного класу з урахуванням базового (одного чи кількох) виникає ієрархія успадкування. Реалізація принципів успадкування є ключовою передумовою можливості повторного використання коду, оскільки це основний інструмент досягнення поліморфізму.

Діаграму класів подано на рисунку 8.

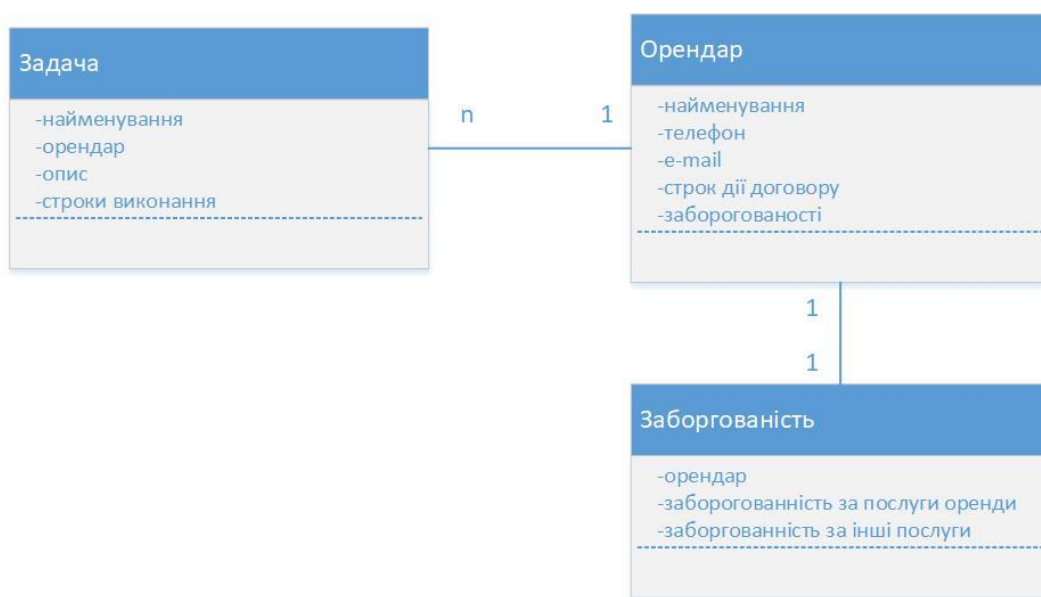


Рисунок 9 – Діаграма класів

2.2.2 Динамічна модель ПО.

Діаграми переходів станів (std) використовуються для моделювання поведінки системи STD (State Transition Diagrams) ми, що залежить від часу або реакцій системи на деякі події.

STD складається з наступних об'єктів:

Стан – модельована система в будь-який заданий момент часу повинна бути точно в одному з кінцевої множини станів

Початковий стан є стартовою точкою для початкового системного переходу, що відповідає стану системи після її інсталяції. STD повинна мати лише один початковий стан, а також будь-яке (кінцеве) число завершальних станів.

Перехід визначає переміщення моделі, що моделюється, з одного стану в інший. При цьому ім'я переходу ідентифікує подію, що є причиною переходу та управляє ним. Ця подія зазвичай складається з керуючого потоку (сигналу), що виникає як у зовнішньому світі, так: і всередині системи при виконанні певної умови.

Дія - це операція, яка може бути пов'язана з переходом, що виконується при виконанні переходу.

На STD стані представляються вузлами, а переходи - дугами. Умови ідентифікуються ім'ям переходу і збуджують виконання переходу. Дії чи відгуки на події прив'язуються до переходів і записуються під умовою. Початковий стан на діаграмі повинен мати вхідний перехід, що зображується потоком зі стартового вузла.

Застосовуються два способи побудови STD. Перший спосіб полягає в ідентифікації всіх можливих станів та подальшому дослідженні всіх безглузких зв'язків (переходів) між ними. За другим способом спочатку будується початковий стан, потім наступні його і т.д.

Діаграму переходу станів додатку подано на рисунку 10.



Рисунок 11 – Діаграма переходу станів.

2.3 Робочий проект

2.3.1 Вибір мови та середовища розробки

Розроблена розподілена клієнт - серверна система повинна складатися з клієнтського додатка, локальної бази даних та сервера, на якому розташовано основну базу даних.

Зв'язок пристроїв повинен відбуватися по одному з широко використовуваних протоколів, що не потребують спеціальної ліцензії на використання.

Як мобільна платформа для створення MVP програми буде використаний Android. З цієї причини в якості мови реалізації додатку використано Java.

Java - це об'єктно-орієнтована мова програмування загального призначення, заснована на класах, розроблена з меншими залежностями реалізації. Широко використовується для розробки програм у ноутбуках, центрах обробки даних, ігрових консолях, наукових суперкомп'ютерах, смартфонах, інтернеті речей і т.д.

Важливою причиною величезної популярності Java є незалежність від платформи, так звана мультиплатформенна підтримка. Програми Java можуть виконуватись на різних машинах, якщо існує JRE (Java Runtime Environment).

Java є вибором багатьох розробників, тому що поєднує інновації зі стабільністю, і що дуже важливо, має повну або майже повну зворотну сумісність: код, написаний на перших версіях Java, як і раніше, буде запускатися на сучасних JVM (Java Virtual Machine).

Оскільки в об'єктах відсутні посилання на зовнішні дані, Java може похвалитися надійністю.

Особливості Java:

Багатопотоковість. Можливості багатопотоковості вбудовані прямо в мову Java. Це означає, що можна створювати інтерактивні та чуйні програми з кількома паралельними потоками активності.

Висока продуктивність. Архітектура Java призначена для зменшення витрат накладних під час використання. Концепція багатопоточності Java також збільшує швидкість виконання програм Java

Незалежність від платформи. Програми Java компілюються у формат байтового коду, який залежить від архітектури машини, але може бути легко переведений на конкретну машину з допомогою віртуальної машини Java (JVM). Це значна перевага при розробці аплетів або програм, що завантажуються з Інтернету та необхідні для роботи в різних системах.

Повна реалізація патернів ООП. Заснований на досвіді C++, Java розширює функціональні зможу повної реалізації патернів об'єктно – орієнтованого програмування.

Надійність та безпека. Код Java є надійним, оскільки проходить перевірку на етапі компіляції та складання програми до моменту запуску на віртуальній машині. Він також надає концепт обробки винятків виявлення логічних помилок, які можуть призвести до збою системи. Безпека також у тому, що з конвертації коду між машинами відбувається перевірка коду до виконання.

Використання Java:

- a) Розробка мобільних додатків для Android
- b) Створення корпоративного програмного забезпечення
- c) Значна частина промисловості програмного забезпечення у різних проявах.
- d) Серверні технології: Apache, JBoss, GlassFish.

В якості локальної бази даних на терміналі користувача буде використано SQLite. SQLite – система управління базою даних, що зберігає таблиці у єдиному файлі. SQLite можна вважати стандартом де-факто для управління базами даних, розташованими на локальному терміналі.

В якості віддаленої бази даних буде використано MySQL. MySQL – вільна реляційна система управління базами даних. Розробку та підтримку MySQL здійснює корпорація Oracle, яка отримала права на торгову марку разом із

поглиненою Sun Microsystems, яка раніше придбала шведську компанію MySQL AB. MySQL – визнане рішення для малого та середнього бізнесу, що має величезну популярність. Серед плюсів цього рішення: простота використання та звичайно ж свобода використання.

Базу даних MySQL буде розгорнуто у Docker контейнері, що дозволить проводити тестування на локальному комп'ютері розробника з подальшим переносом БД на віддалений сервер без втрати даних та налаштувань.

Docker— програмне забезпечення з відкритим вихідним кодом, яке використовується для розробки, тестування, доставки та запуску веб-застосунків у середовищах з підтримкою контейнеризації. Він необхідний для більш ефективного використання системи та ресурсів, швидкого розгортання готових програмних продуктів, а також для їх масштабування та перенесення в інші середовища з гарантованим збереженням стабільної роботи.

Розробка Docker була розпочата в 2008 році, а в 2013 році він був опублікований як ПЗ, що вільно розповсюджується, під ліцензією Apache 2.0. Як тестовий додаток Docker був включений до дистрибутиву Red Hat Enterprise Linux 6.5. У 2017 році було випущено комерційну версію Docker з розширеними можливостями.

Docker працює в Linux, ядро яких підтримує cgroups, а також ізоляцію простору імен. Для встановлення та використання на платформах, відмінних від Linux, існують спеціальні утиліти Kitematic або Docker Machine.

Основний принцип роботи Docker – контейнеризація програм. Цей тип віртуалізації дозволяє упаковувати програмне забезпечення ізольованими середовищами — контейнерами. Кожен із цих віртуальних блоків містить усі необхідні елементи для роботи програми. Це дозволяє одночасного запуску великої кількості контейнерів на одному хості.

Переваги використання Docker:

1. Мінімальне споживання ресурсів – контейнери не віртуалізують усю операційну систему (ОС), а використовують ядро хоста та ізолюють програму

на рівні процесу. Останній споживає набагато менше ресурсів локального комп'ютера, аніж віртуальна машина.

2. Швидкісне розгортання - допоміжні компоненти можна не встановлювати, а використовувати вже готові образи (шаблони). Наприклад, немає сенсу постійно встановлювати і налаштовувати Linux Ubuntu. Достатньо 1 раз її встановлювати, створити образ і постійно використовувати лише оновлюючи версію при необхідності.
3. Зручне приховування процесів - для кожного контейнера можна використовувати різні методи обробки даних, приховуючи процеси фону.
4. Робота з небезпечним кодом – технологія ізоляції контейнерів дозволяє запускати будь-який код без шкоди для ОС.
5. Просте масштабування – будь-який проект можна розширити, впровадивши нові контейнери.
6. Зручний запуск - програму, що знаходиться всередині контейнера, можна запустити на будь-якому docker-хості.
7. Оптимізація файлової системи - образ складається з шарів, які дозволяють ефективно використовувати файлову систему.

В якості середовища розробки в роботі використовується Android – studio. Android-studio є офіційною IDE (скорочено англійською мовою, Integrated Development Environment) для платформи Android абсолютно безкоштовно через ліцензію Apache License 2.0 та кросплатформенність (Microsoft Windows, macOS та GNU/Linux).

Android-studio засновано на програмному забезпеченні IntelliJ IDEA JetBrains і було випущено як заміну Eclipse. Android -studio є офіційною IDE для розробки програм Android.

Цей програмний продукт пропонує повний інструмент для розробки та налагодження програм для мобільної операційної системи Google.

З його допомогою можна виконувати редагування коду, налагодження, використовувати інструменти підвищення продуктивності. Продукт має гнучку

систему компіляції та миттєве створення та розгортання, що дозволяє зосередитись саме на створенні програми.

Android-studio включає шаблони проектів та коду, які спрощують додавання усталених шаблонів, наприклад, бічна панель навігації та перегляд сторінки.

Серед основних характеристик продукту можна виділити:

- Функції інтеграції ProGuard та підписи додатків.
- Рендерінг у реальному часі
- Консоль розробника: поради щодо оптимізації, допомогу з перекладом, статистика використання.
- Підтримка збирання на основі Gradle. Gradle — система автоматичного складання, побудована на принципах Apache Ant та Apache Maven, але надає DSL мовами Groovy та Kotlin замість традиційної XML-подібної форми представлення конфігурації проекту.
- Рефакторинг для Android та швидкі виправлення.
- Багатий редактор макетів, який дозволяє користувачам перетягувати компоненти інтерфейсу користувача.
- Інструменти Lint для визначення продуктивності, зручності використання, сумісності версій та інших проблем.
- Шаблони для створення спільних макетів Android та інших компонентів.
- Підтримка програмування програм для Android Wear.
- Інтегрована підтримка Google Cloud Platform, що забезпечує інтеграцію з Google Cloud Messaging та App Engine.
- Віртуальний пристрій Android, що використовується для запуску та тестування програм.

Як працює Gradle Build. Конфігурація збірки Gradle зберігається у файлі з ім'ям `build.gradle`, який базується на доменно-специфічній мові (DSL). Це спеціалізована мова сценаріїв, що поєднує декларативні та імперативні конструкції. Цей файл конфігурації знаходиться в кореневій папці проекту та

визначає проект, його завдання, залежності та інші елементи, пов'язані з проектом. Gradle підтримує багатопроєктну збірку, де кожен проект має власний файл `build.gradle`, розташований у кореневій папці проекту.

«Завдання» - це елементарна одиниця роботи, яка визначає, що слід робити за її виклику. Складання можна описати як набір завдань. Кожне завдання складається з дій, входів та виходів; кожна частина не є обов'язковою.

Щоб побачити список усіх доступних завдань для проекту Gradle, ви можете виконати завдання `gradle` в командному рядку. Для простого Java-проекту ви можете вибрати одне з наперед визначених завдань з наступних категорій:

- Складання завдань
- Завдання налаштування збирання
- Поширення завдань
- Документування завдань
- Довідкові завдання
- Завдання перевірки
- Інші завдання

2.3.2 Діаграми компонентів

Діаграма компонентів дозволяє визначити склад програмних компонентів, ролі яких може виступати вихідний, бінарний і виконуваний код, і навіть встановити залежності з-поміж них.

При розробці діаграм компонентів переслідуються цілі:

- специфікація загальної структури вихідного коду системи;
- специфікація можливого варіанта системи.

Ця діаграма забезпечує узгоджений перехід від логічного до фізичного уявлення системи як програмних компонентів. Одні компоненти можуть бути тільки на етапі компіляції програмного коду, інші – на етапі його виконання. Основними елементами діаграми є компоненти, інтерфейси та залежності між

ними. Крім того, на ній можуть відображатися ключові класи, що входять до компонентів.

Компонент (англ. component) – це фізична частина системи. Компоненти, що являють собою файли з вихідним кодом класів, бібліотеки, модулі тощо, які повинні володіти узгодженим набором інтерфейсів. Для їхнього графічного уявлення використовуються стандартизовані графічні символи.

Діаграму компонентів додатку представлено на рисунку

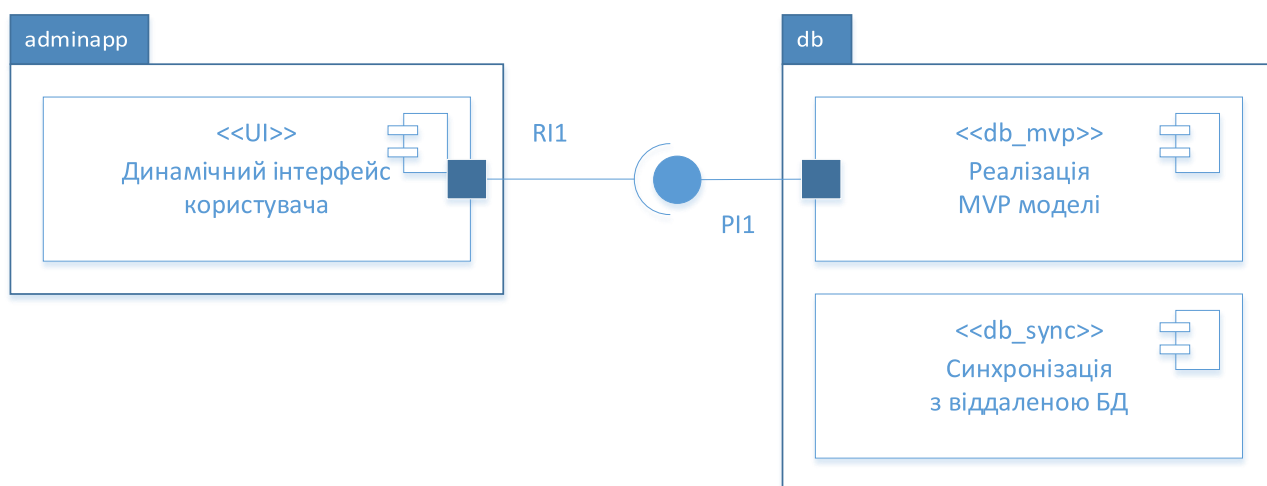


Рисунок 12 – Діаграма компонентів.

Додаток складається з двох пакетів, один з яких відповідає за динамічний інтерфейс користувача, другий – за роботу з БД. В першому пакеті присутній єдиний компоненті, інший складається з двох компонентів. Один компонент реалізує роботу з даними за патерном MVP, інший здійснює фонову синхронізацію даних з основною базою даних.

2.3.3 Діаграми розміщення

Діаграми розгортання використовуються для візуалізації апаратних процесорів/вузлів/пристроїв системи, каналів зв'язку між ними та розміщення програмних файлів на цьому апаратному забезпеченні.

Діаграма розгортання – це тип UML-діаграми, яка показує архітектуру виконання системи, включаючи такі вузли, як апаратні або програмні середовища виконання, а також проміжне програмне забезпечення, яке їх з'єднує.

Діаграми розгортання зазвичай використовуються для візуалізації фізичного апаратного та програмного забезпечення системи. Використовуючи його, можна зрозуміти, як система буде фізично розгорнута на апаратному забезпеченні.

Діаграми розгортання допомагають моделювати апаратну топологію системи порівняно з іншими типами UML-діаграм, які здебільшого описують логічні компоненти системи.

Діаграму розгортання додатку показано на рисунку. Вона складається з двох пристроїв: мобільного термінала користувача з встановленим додатком та локальною БД, а також віддаленого серверу с СУБД MySQL.

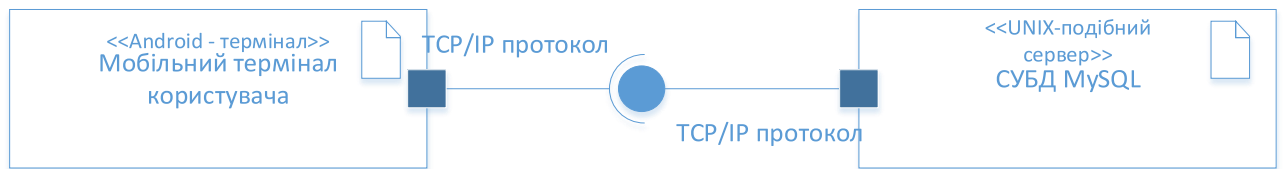


Рисунок 13 – Діаграма розгортання.

3. РЕАЛІЗАЦІЯ ПРОЕКТУ АВТОМАТИЗОВАНОЇ СИСТЕМИ

3.1 Реалізація класів і кодування

Згідно обраного в попередньому розділі інструментарію, додаток розроблено у Android Studio з використанням SDK 31 та Pixel 5 у якості емулятора Android – термінала.

Додаток складається з двох основних пакетів: `adminapp` та `mvc`, що відповідають за реалізацію інтерфейсу користувача та доступ до даних відповідно.

Спочатку розглянемо організацію роботи з даними, у тому числі представлення моделей, що використовуються, а потім – загальний графічний інтерфейс користувача. Це дозволить дати більше інформації відносно сутностей, інформацію про екземпляри та перелік яких відтворює графічний інтерфейс, після детального опису роботи з даними.

3.1.1 Робота з даними.

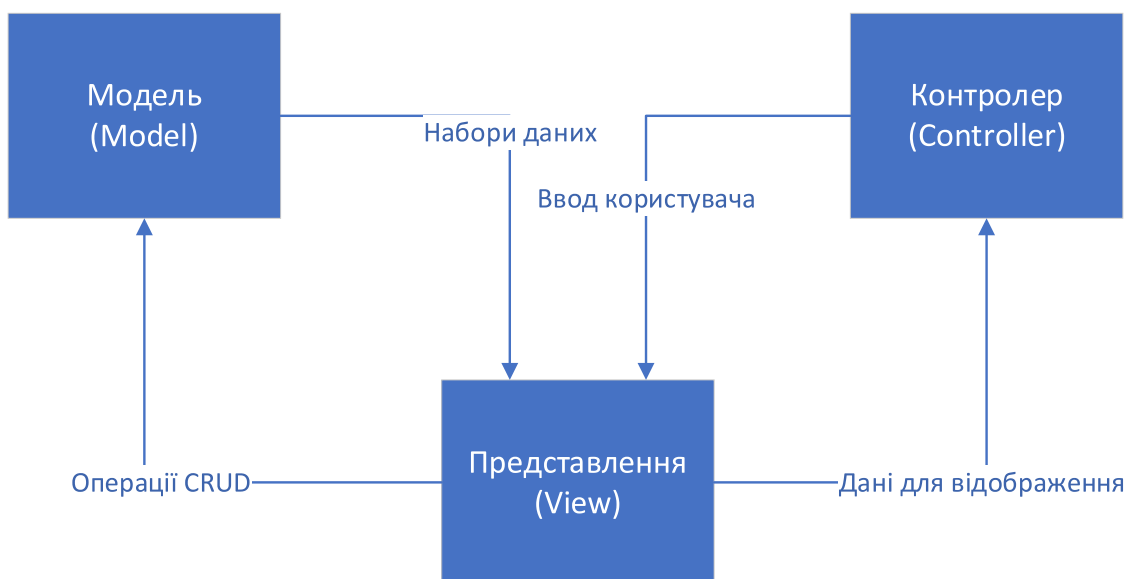


Рисунок 14 – Схема MVC моделі.

MVC - це конструкційний шаблон, який описує спосіб побудови структури нашого додатка, сфери відповідальності та взаємодії кожної з елементів у цій структурі. Ідея, яка лежить в основі конструкційного шаблону MVC, дуже

проста: потрібно чітко розділяти відповідальність за різну функціональність у додатку.

Додаток поділяється на три основні компоненти, кожен з яких відповідає за різні завдання.

Контролер керує запитам користувача, що створюються коли користувач натискає на елементи інтерфейсу до виконання різних дій. Його основна функція — викликати та координувати дію необхідних ресурсів та об'єктів, необхідних для виконання дій, що задаються користувачем. Зазвичай контролер викликає відповідну модель завдання і вибирає відповідний вид.

Модель - це дані та правила, що використовуються для роботи з даними, що представляють концепцію управління програмою. У будь-якому додатку вся структура моделюється як дані, що обробляються певним чином.

Вид забезпечує різні способи представлення даних, отриманих з моделі. Він може бути шаблоном, який наповнюється даними. Можливо кілька різних видів, і контролер вибирає, який найкраще підходить для поточної ситуації.

Розберемо використання MVC моделі у додатку на прикладі роботи з орендарем. Також до розгляду включимо два специфічних об'єкти, що є сервісними у роботі з БД: DAO та DB.

Модель орендаря описана у класі `com.example.adminapp.mvc.tenant.Tenant`. Опис моделі представляє включає назву моделі та визначення її полів з використанням декораторів `@Entity` та `@ColumnInfo`. Декоратор `@PrimaryKey` використовується для визначення первинного ключа таблиці БД.

Модель орендаря, що використовується в додатку, має чотири поля, включаючи унікальний ідентифікатор: найменування, контактні дані, суму заборгованості за оренду та суму заборгованості за інші послуги. Унікальний ідентифікатор потрібен у службових цілях, у тому числі для виконання синхронізації БД.

Для роботи з базою даних використовується бібліотека Room. Використання цієї бібліотеки також вважається рекомендованою практикою для Android – розробників.

Для роботи з даним окрім моделі використовується об'єкт роботи з даними та власне клас, що є описом таблиці бази даних, в якій зберігається інформація про відповідні екземпляри сутностей.

З'єднання з базою даних є однією з найважливіших складових програми. Завжди виділяється частина коду, модуль, що відповідає за передачу запитів у БД та обробку отриманих від неї відповідей. У випадку, визначення Data Access Object визначає його як прошарок між БД і системою. DAO абстрагує сутності системи та робить їх відображення на БД, визначає загальні методи використання з'єднання, його отримання, закриття та (або) повернення до Connection Pool.

В додатку створено клас DaoAccess, що описує необхідні запити до бази даних: отримання усіх елементів, отримання одного елемента.

Для опису роботи з БД використовуються класи TenantDatabase та TenantRepository. Перший з класів є успадкованим від RoomDatabase та містить виключно абстрактний клас, що визначає параметри БД з використанням декоратору. TenantRepository описує параметри доступу до таблиці бази даних та CRUD операції з цією таблицею.

Таким чином має три врапери для бази даних, таблиці бази даних та запису бази даних.

Використання враперу Room для СУБД SQLite суттєво спрощує та пришвидшує написання коду для роботи з базами даних за рахунок використання перелічених абстракцій.

Треба зауважити, що такий підхід найчастіше використовується при створенні мобільних та web-додатків.

Розглянувши абстракції роботи з БД, перейдемо до розгляду контролеру та представлення.

За відображення окремого екземпляру сутності відповідає клас AddTenantActivity. Це представлення дозволяє як створювати, так й редагувати раніше створені екземпляри сутностей.

Додати орендаря		Перегляд орендаря	
Магазин жіночої білизни		Магазин жіночої білизни	
Оксана, +380931932373		Оксана, +380931932373	
За оренду, грн.	---	За оренду, грн.	1000.00
Компенсація послуг, грн.	---	Компенсація послуг, грн.	800.00
10 чэрв. 2022 19:58:23		10 чэрв. 2022 19:41:27	

Рисунок 15 – Створення та редагування екземпляру сутності “Орендар”.

У випадку перегляду або редагування існуючої сутності користувачеві доступний перегляд сум заборгованості за оренду та компенсації комунальних та інших послуг.

Показаний на рисунку 15 фрагмент є контролером моделі, відображена інформація створюється з використанням представлення.

Якщо переглянути представлення переліку екземплярів сутностей, що показано на рисунку 16, бачимо інше представлення екземпляру сутності “Орендар”. У представленні відображене лише найменування та загальна сума заборгованості, що є сумою двох статей заборгованості орендаря, описаних вище. Це нагально демонструє використання різних представлень для однієї

сутності. У випадку, що розглядається, це дозволяє управляти деталізацією інформації. Безумовно є інші, більш складні використання подання,

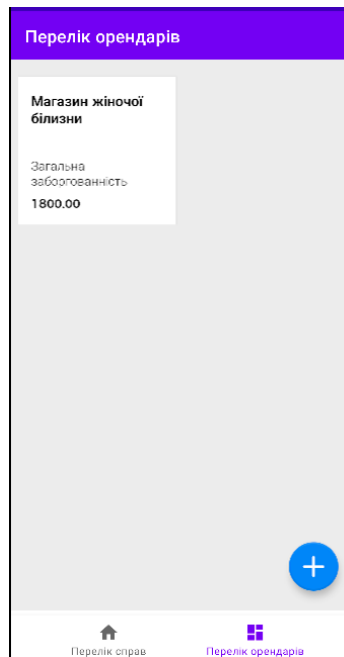


Рисунок 16 – Представлення переліку сутностей “Орендар”.

Представлення списку екземплярів сутності “Орендар” реалізовано класом `TenantListActivity`. Клас реалізує функції додавання нового екземпляру сутності при натисканні на кнопку “+” та редагування екземпляру сутності при натисканні на плашку, що є представленням відповідного екземпляру сутності “Орендар”.

Подібним образом реалізована робота й з переліком справ менеджера з оренди.

Говорячи про активності та представлення також треба звернути увагу на об’єкти `Layout`(макет, розмітка, компоновання).

Компоновання зберігається у вигляді файлу XML у папці `/res/layout`. Це зроблено для того, щоб відокремити код від дизайну, як це прийнято у багатьох технологіях (HTML та CSS). Окрім основного компоновання для всього екрана, існують дочірні елементи компоновання для групи елементів. По суті, компоновання – це якийсь візуальний шаблон для інтерфейсу користувача програми, який дозволяє керувати елементами управління, їх властивостями і

розташуванням. Android Studio включає спеціальний редактор для створення розмітки двома способами. Редактор має дві вкладки: одна дозволяє побачити, як відображатимуться елементи керування, а друга – створювати XML-розмітку вручну. Створюючи інтерфейс користувача в XML-файлі, можна відокремити представлення програми від програмного коду.

Представлення для списку справ та для окремої справи зображені на рисунку 17

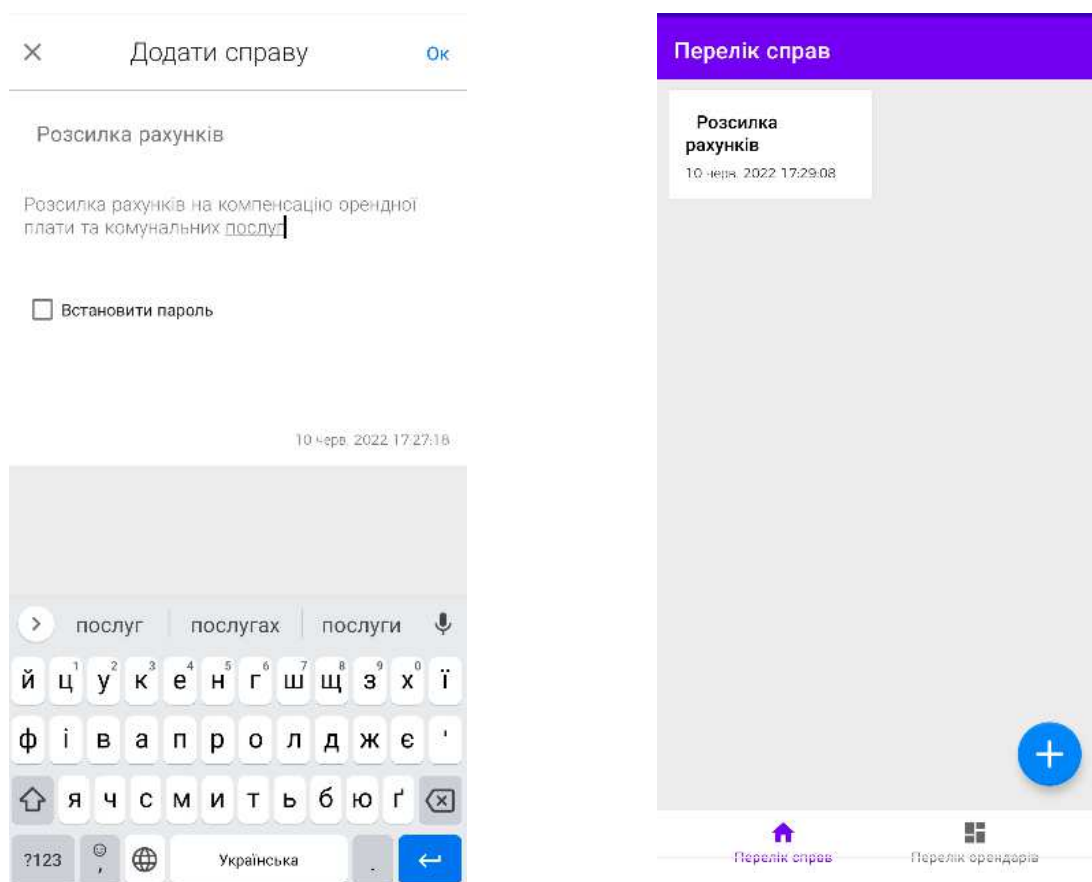


Рисунок 17 – Представлення списку та окремої справи.

Для справ можна задати час нагадування, що реалізується окремою активністю Notification. Notification використовує системні ресурси, для створення нагадувань користувачеві. Це дозволяє використовувати налаштування системи для подання нагадувань: використовувати звук, вібрацію, але просто додавати нагадування в шторку повідомлень.

3.1.2 Графічний інтерфейс користувача.

Графічний інтерфейс користувача реалізовано не тільки з використанням активностей, але й з використанням фрагментів, що вважається доброю практикою в розробці додатків для операційної системи Android.

Основна активність, посилання на яку додана до Android-маніфесту, реалізовано класом MainActivity. Клас реалізує навігацію (переключення) між фрагментами, що відповідають за роботу з переліком задач та з переліком орендарів.

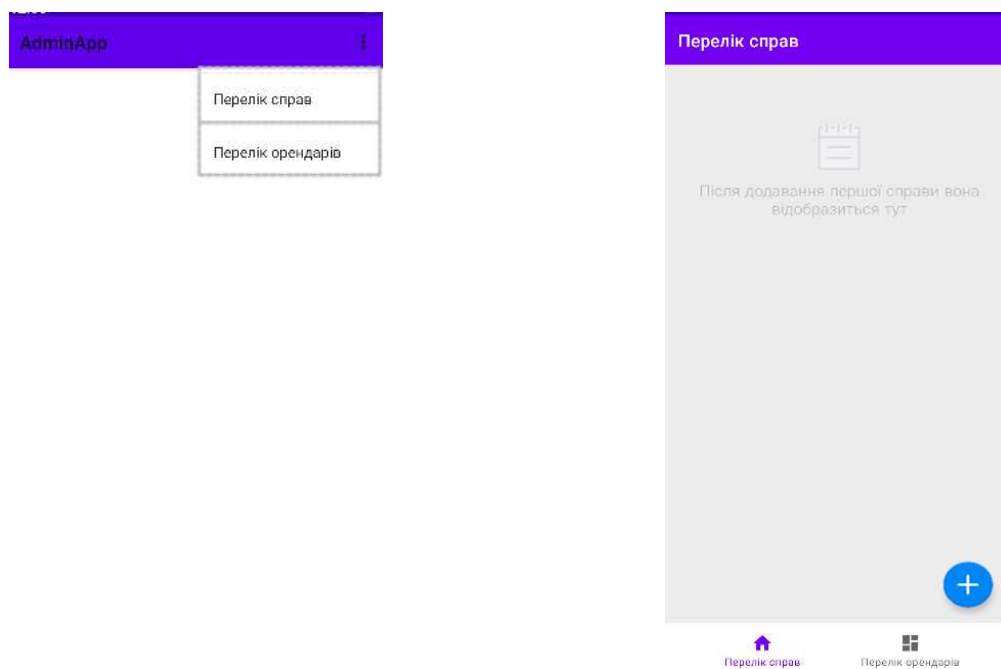


Рисунок 18 – Організація навігації в додатку.

Спочатку створюється звичайне меню, яке показано на рисунку 18 зліва, а потім з використанням класу `android.navigation.Navigation` перетворюється у вигляд, що поданий на рисунку 18 справа.

Таким чином досягається мета використання різних фрагментів одночасно залишаючи навігаційне меню в нижній частині екрану користувача. Заголовок вікна змінюється при переході до відповідної вкладки.

Фрагмент — це звичайний клас Java, який успадковується від класу `Fragment`. Однак, як і клас `Activity`, фрагмент може використовувати `xml`-файли `layout` для визначення графічного інтерфейсу. І таким чином, можна додати

окремо клас Java, який представляє фрагмент, і файл XML для зберігання в ньому розмітки інтерфейсу, який використовуватиме фрагмент.

З іншого боку, фрагмент можна розглядати як елемент компоновки сторінки інтерфейсу. У додатку, що розглядається, фрагменти використовуються для створення динамічної навігації по вкладках, залишаючи одночасно навігаційне меню на екрані без необхідності створювати його повторно.

Підводячи підсумки короткого огляду реалізації, можна стверджувати, що додаток створено у суворій відповідності з патернами об'єктно орієнтованого програмування.

Розподілення відповідальності між пакетами, що відповідають за графічний інтерфейс та за роботу з даними, є виконанням концепції інкапсуляції. Використання `room` й разом з цією бібліотекою абстракції даних є виконанням концепції абстракції.

Звичайно, в додатку виконується концепція успадковування при створенні майже усіх класів, що є спадкоємцями відповідних класів стандартних бібліотек `androidx`.

3.2 Тестування функціональності і користувацького інтерфейсу

Тестування функціональності проводимо з використанням юніт-тестування.

Локальні тести перевіряють роботу методу, класу, компонента. Тест не залежить від Android, по суті, ви перевіряєте код Java, який можна перевірити на звичайному комп'ютері без участі пристрою або емулятора.

Популярними інструментами для юніт-тестів є: JUnit, Mockito, PowerMock.

Для інструментальних тестів наявність пристрою або емулятора обов'язково, тому що потрібно тестувати натискання кнопки, введення тексту, прокручування, торкання та інші операції. Тести проводять у папці `androidTest`.

Інструменти для тестування: Espresso, UIAutomator, Robotium, Appium, Calabash, Robolectric.

Тести розташовано в папці `com.example.adminapp.test`.

До тестування користувацького інтерфейсу існує декілька підходів. В першу чергу тому, що користувацький інтерфейс тестується з точки зору зручності на етапі розробки – до створення додатку, та на етапі функціонального тестування додатку. Функціональне тестування додатку зазвичай проводиться на прикінцевих етапах його створення або вже після створення MVP.

Для тестування зручності інтерфейсу використовувався виключно так званий чек-лист, тестування паперових або інтерактивних прототипів не проводилось.

Під час тестування додатку проводилося інтеграційне тестування, що дозволяє виявити помилки у роботі графічного інтерфейсу користувача. Наприклад, при застосуванні певних моделей смартфонів, або взагалі в наслідок помилок у розробці.

Після усунення помилок перевірено відповідність створеного додатку функціональним вимогам, що сформульовано наприкінці першого розділу. Вимоги до додатку, як функціональні, так й не функціональні виконано.

4 ОХОРОНА ПРАЦІ

4.1 Вступ

Вимоги щодо охорони праці, зокрема охорони праці офісних працівників, містять Кодекс законів про працю, Закон України «Про охорону праці», а також інші підзаконні нормативно-правові акти. У відповідності до вимог ст. 153 Кодексу законів про працю України та ст. 6 Закону України «Про охорону праці» на всіх підприємствах, в установах, організаціях створюються безпечні і нешкідливі умови праці. Забезпечення безпечних і нешкідливих умов праці покладається на власника або уповноважений ним орган.

Умови праці на робочому місці, безпека технологічних процесів, машин, механізмів, устаткування та інших засобів виробництва, стан засобів колективного та індивідуального захисту, що використовуються працівником, а також санітарно-побутові умови повинні відповідати вимогам нормативних актів про охорону праці. Власник або уповноважений ним орган повинен впроваджувати сучасні засоби техніки безпеки, які запобігають виробничому травматизму, і забезпечувати санітарно-гігієнічні умови, що запобігають виникненню професійних захворювань працівників.

Стаття 158 Кодексу законів про працю України встановлює обов'язок власника або уповноваженого ним органу вживати заходів щодо полегшення і оздоровлення умов праці працівників шляхом впровадження прогресивних технологій, досягнень науки і техніки, засобів механізації та автоматизації виробництва, вимог ергономіки, позитивного досвіду з охорони праці, зниження та усунення запиленості та загазованості повітря у виробничих приміщеннях, зниження інтенсивності шуму, вібрації, випромінювань тощо. А згідно з ч. 1 ст. 13 Закону України «Про охорону праці» роботодавець зобов'язаний створити на робочому місці в кожному структурному підрозділі умови праці відповідно до нормативно-правових актів, а також забезпечити додержання вимог законодавства щодо прав працівників у галузі охорони праці.

Робочі місця офісних працівників, обладнані персональними комп'ютерами, повинні відповідати вимогам «Правил охорони праці під час експлуатації електронно-обчислювальних машин», затверджених Наказом Державного комітету України з промислової безпеки, охорони праці та гірничого нагляду від 26.03.2010 року № 65 (Правила), та «Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин», затверджених постановою Головного державного санітарного лікаря України від 10.12.98 N 7 (ДСанПіН 3.3.2-007-98). Правила поширюються на всіх суб'єктів господарювання незалежно від форм власності, які у своїй діяльності здійснюють роботу, пов'язану з персональними комп'ютерами, у тому числі на тих, які мають робочі місця, обладнані персональними комп'ютерами і периферійними пристроями. Зазначені нормативно-правові акти встановлюють санітарно-гігієнічні вимоги до приміщення, в якому розташоване робоче місце, власне до робочого місця, освітлення, рівнів вібрації і шуму, мікроклімату в приміщенні тощо.

4.2 Аналіз шкідливих на небезпечних факторів

За виконання робіт на персональному комп'ютері (ПК). Небезпечні та шкідливі виробничі фактори. Класифікація” можуть мати такі фактори:

- підвищена температура поверхонь ПК;
- підвищена чи знижена температура повітря робочої зони;
- виділення повітря робочої зони низки хімічних речовин;
- підвищена чи знижена вологість повітря;
- підвищений або знижений рівень негативних та позитивних аероіонів;
- підвищене значення напруги в електричному ланцюзі, замикання;
- підвищений рівень статичної електрики;
- підвищений рівень електромагнітних випромінювань;
- підвищена напруженість електричного поля;
- відсутність або нестача природного світла;

- недостатня штучна освітленість робочої зони;
- підвищена яскравість світла;
- підвищена контрастність;
- пряма та відбита блискітність;
- зорове напруження;
- монотонність трудового процесу;
- нервово-емоційне навантаження.

До фізичних шкідливих та небезпечних факторів відносяться: підвищені рівні електромагнітного, рентгенівського, ультрафіолетового та інфрачервоного випромінювання; підвищений рівень статичної електрики та запиленості повітря робочої зони; підвищений вміст позитивних аеронів та знижений вміст негативних аероіонів у повітрі робочої зони; підвищений рівень блискоті та засліпленості; нерівномірність розподілу яскравості у полі зору; підвищена яскравість світлового зображення; підвищене значення напруги в електричному ланцюзі, замикання якого може статися через тіло людини.

Хімічні шкідливі та небезпечні фактори такі: підвищений вміст у повітрі робочої зони двоокису вуглецю, озону, аміаку, фенолу та формальдегіду.

Психофізіологічні шкідливі та небезпечні фактори: напруга зору та уваги; інтелектуальні, емоційні та тривалі статичні навантаження; монотонність праці; великий обсяг інформації, що обробляється в одиницю часу; нераціональна організація робочого місця.

Типовими відчуттями, які відчують до кінця робочого дня оператори ПЕОМ, є: перевтома очей, головний біль, біль у м'язах шиї, рук і спини, зниження концентрації уваги.

Вже в перші роки комп'ютеризації було відзначено специфічну зорову втому у користувачів дисплеїв, що отримала загальну назву "комп'ютерний зоровий синдром". Однією з причин є те, що зорова система людини, що сформувалася за мільйони років еволюції, пристосована для сприйняття об'єктів у відбитому світлі (друковані тексти, малюнки тощо), а не для роботи за

дисплеєм. Зображення на дисплеї принципово відрізняється від звичних оку об'єктів спостереження - воно світиться, мерехтить, складається з дискретних точок, а кольорове комп'ютерне зображення не відповідає природним кольорам. Але не тільки особливості зображення на екрані викликають втому. Велике навантаження орган зору відчуває при введенні інформації, так як користувач змушений часто переводити погляд з екрана на текст і клавіатуру, що знаходяться на різній відстані і освітлені по-різному. Зорова втома проявляється скаргами на затуманювання зору, труднощі при перенесенні погляду з ближніх предметів на дальні і з далеких на ближні, зміни забарвлення предметів, їх двоїння, почуття печіння, «піску» в очах, почервоніння повік, болі при русі очей.

4.3 Розробка заходів щодо зменшення впливу шкідливих та небезпечних факторів при роботі з персональним комп'ютером

4.3.1 Заходи щодо зменшення впливу

Персональні комп'ютери, периферійні пристрої, інше устаткування (апарати управління, контрольно-вимірювальні прилади, світильники), електропроводи та кабелі за виконанням і ступенем захисту мають відповідати класу зони, мати апаратуру захисту від струму короткого замикання та інших аварійних режимів.

Під час монтажу та експлуатації ліній електромережі необхідно повністю унеможливити виникнення електричного джерела загоряння внаслідок короткого замикання та перевантаження проводів, обмежувати застосування проводів з легкозаймистою ізоляцією і, за можливості, застосовувати негорючу ізоляцію. Лінія електромережі для живлення персональних комп'ютерів і периферійних пристроїв виконується як окрема групова трьох-провідна мережа шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Не допускається використовувати нульовий робочий провідник як нульовий захисний провідник. Нульовий захисний провідник прокладається від стійки групового розподільного щита,

розподільного пункту до розеток електроживлення. Не допускається підключати на щиті до одного контактного затискача нульовий робочий та нульовий захисний провідники. Площа перерізу нульового робочого та нульового захисного провідника в груповій трипровідній мережі має бути не менше площі перерізу фазового провідника. Усі провідники мають відповідати номінальним параметрам мережі та навантаження, умовам навколишнього середовища, умовам розподілу провідників, температурному режиму та типам апаратури захисту.

У приміщенні, де одночасно експлуатуються понад п'ять персональних комп'ютерів і периферійних пристроїв, на помітному та доступному місці встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення. Персональні комп'ютери і периферійні пристрої повинні підключатися до електромережі тільки за допомогою справних штепсельних з'єднань і електророзеток заводського виготовлення. У штепсельних з'єднаннях та електророзетках, крім контактів фазового та нульового робочого провідників, мають бути спеціальні контакти для підключення нульового захисного провідника. Їхня конструкція має бути такою, щоб приєднання нульового захисного провідника відбувалося раніше, ніж приєднання фазового та нульового робочого провідників. Порядок роз'єднання при відключенні має бути зворотним.

Устаткування, що становить джерело шуму (АЦП, принтери тощо), слід розташовувати поза приміщеннями, де знаходяться робочі місця. Для забезпечення допустимих рівнів шуму на робочих місцях слід застосовувати засоби звукопоглинання, вибір яких має обґрунтовуватись спеціальними інженерно-акустичними розрахунками. Значення характеристик вібрації на робочих місцях мають не перевищувати допустимі відповідно до СН 3044-84, ГОСТ 12.1.012-90.

Під час використання обчислювальної техніки необхідно дотримуватись режими праці та відпочинку.

При організації праці, що пов'язана з використанням персональних комп'ютерів, для збереження здоров'я працюючих, запобігання професійним захворюванням і підтримки працездатності слід передбачити внутрішньозмінні регламентовані перерви для відпочинку. Внутрішньозмінні режими праці і відпочинку мають передбачати додаткові нетривалі перерви в періоди, що передують появі об'єктивних і суб'єктивних ознак стомлення і зниження працездатності. За основну роботу з персональним комп'ютером слід вважати таку, що займає не менше 50% часу впродовж робочої зміни. Протягом дня мають передбачатися:

- перерви для відпочинку і вживання їжі (обідні перерви);
- перерви для відпочинку і особистих потреб (згідно з трудовими нормами);
- додаткові перерви, що вводяться для окремих професій з урахуванням особливостей трудової діяльності.
- для розробників програм слід призначати регламентовану перерву для відпочинку тривалістю 15 хвилин через кожну годину роботи за персональним комп'ютером;
- для операторів персональних комп'ютерів слід призначати регламентовані перерви для відпочинку тривалістю 15 хвилин через кожні дві години;
- для операторів комп'ютерного набору слід призначати регламентовані перерви для відпочинку тривалістю 10 хвилин після кожної години роботи за персональним комп'ютером.

У всіх випадках, коли виробничі обставини не дозволяють застосувати регламентовані перерви, тривалість безперервної роботи з персональним комп'ютером не повинна перевищувати 4 години. При 12-годинній робочій зміні регламентовані перерви повинні встановлюватися в перші 8 годин роботи аналогічно перервам при 8-годинній робочій зміні, а протягом останніх 4-х годин роботи, незалежно від характеру трудової діяльності, через кожну годину

тривалістю 15 хвилин. З метою зменшення негативного впливу монотонності є доцільним застосовувати чергування операцій усвідомленого тексту і числових даних (зміна змісту роботи), чергування вводу даних та редагування текстів. Для зниження нервово-емоційного напруження, стомлення зорового аналізатору, поліпшення мозкового кровообігу, подолання несприятливих наслідків гіподинамії, запобігання втоми доцільні деякі перерви використовувати для виконання комплексу вправ. В окремих випадках – при хронічних скаргах працюючих на зореве стомлення, незважаючи на дотримання санітарно-гігієнічних вимог до режимів праці і відпочинку, а також застосування засобів локального захисту очей – допускаються індивідуальних підхід до обмеження часу робіт з персональним комп'ютером, зміни характеру праці, чергування з іншими видами діяльності, не пов'язаними з персональним комп'ютером. Активний відпочинок має полягати у виконанні комплексу гімнастичних вправ, спрямованих на зняття нервового напруження, м'язове розслаблення, відновлення функцій фізіологічних систем, що порушуються протягом трудового процесу, зняття втоми очей, поліпшення мозкового кровообігу і працездатності.

За умови високого рівня напруженості робіт з персональним комп'ютером показане психологічне розвантаження у спеціально обладнаних приміщеннях (в кімнатах психологічного розвантаження) під час регламентованих перерв або в кінці робочого дня.

4.3.2 Техніка безпеки при роботі з комп'ютером

Щодня перед початком роботи необхідно очищати монітор від пилу та інших забруднень. Після закінчення роботи персональний комп'ютер і периферійні пристрої повинні бути відключені від електричної мережі. У разі виникнення аварійної ситуації необхідно негайно відключити персональний комп'ютер і периферійні пристрої від електричної мережі. Не допускається:

- виконувати обслуговування, ремонт та налагодження персонального комп'ютеру та периферійних пристроїв безпосередньо на робочому місці оператора;
- зберігати біля персонального комп'ютеру та периферійних пристроїв папір, будь-які носії інформації (диски, флешки тощо), запасні блоки, деталі тощо, якщо вони не використовуються для поточної роботи;
- відключати захисні пристрої, самочинно проводити зміни у конструкції та складі персонального комп'ютеру та периферійних пристроїв або їх технічне налагодження;
- працювати з персональним комп'ютером, у яких під час роботи з'являються нехарактерні сигнали, нестабільне зображення на моніторі тощо;
- працювати з матричним принтером за відсутності вібраційного килимка та зі знятою (піднятою) верхньою кришкою.

Мікроклімат

Приміщення для роботи з персональними комп'ютерами мають бути обладнані системами опалення, кондиціонування повітря, або припливно-витяжною вентиляцією.

У приміщеннях на робочих місцях мають забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря у відповідності до ГОСТ 12.1.005-88, СН 4088-86.

Таблиця 1 – Параметри мікроклімату

Пора року	Категорія робіт	Температура повітря, град. С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
		оптимальна	оптимальна	оптимальна

Холодна	легка-1 а	22 – 24	40 – 60	0,1
	легка-1 б	21 – 23	40 – 60	0,1
Тепла	легка-1 а	23 – 25	40 – 60	0,1
	легка-1 б	22 – 24	40 – 60	0,2

Рівні позитивних і негативних іонів у повітрі мають відповідати санітарно-гігієнічним нормам № 2152-80.

Таблиця 2 – Кількість іонів

Рівні	Кількість іонів в 1 см куб. повітря	
	n +	n –
Мінімально необхідні	400	600
Оптимальні	1500 – 3000	3000 – 5000
Максимально допустимі	50000	50000

Для підтримки допустимих значень мікроклімату та концентрації позитивних та негативних іонів необхідно передбачати установки або прилади зволоження та/або штучної іонізації, кондиціонування повітря.

Освітлення

Приміщення, в яких встановлені персональні комп'ютери, повинні мати природне та штучне освітлення відповідно до СНиП II-4-79.

Природне освітлення має здійснюватися через світлові прорізи, орієнтовані переважно на північ чи північний схід і забезпечувати коефіцієнт природною освітленості (КПО) не нижче ніж 1,5%. Розраховується КПО за методикою, викладеною в СНиП II-4-79.

Штучне освітлення в приміщеннях з робочими місцями має здійснюватися системою загального рівномірного освітлення. Зазначення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300-500лк.

Якщо ці значення освітленості неможливо забезпечити системою загального освітлення, допускається використовувати місцеве освітлення. При цьому світильники місцевого освітлення слід встановлювати таким чином, щоб не створювати відблисків на поверхні екрана, а освітленість екрана має не перевищувати 300лк. Як джерела світла в разі штучного освітлення мають застосовуватись переважно люмінесцентні лампи типу ЛБ. Допускається використання світильників таких класів світлорозподілу:

- прямого світла – П;
- переважно прямого світла – Н;
- переважно відбитого світла – В.

Для загального освітлення слід застосовувати світильники серії ЛПО 36 із дзеркальними ґратами, що укомплектовані високочастотними пускорегулювальними апаратами (ВЧ ПРА). Допускається застосовувати світильники цієї серії без ВЧ ПРА тільки в модифікації «Кососвітло». При відсутності світильників серії ЛПО36 з ВЧ ПРА і без ВЧ ПРА модифікації «Кососвітло» допускається застосування світильників загального освітлення серії:

- ЛПО13 – 2×40/Б – 01;
- ЛПО13 – 4×40/Б – 01;
- ЛПО13 – 2×40 – 06;
- ЛПО13 – 2×65 – 06;
- ЛСО05 – 2×40 – 001;
- ЛСО05 – 2×40 – 003;
- ЛСО04 – 2×36 – 008;
- ЛПО34 – 4×36 – 002;
- ЛПО34 – 4×58 – 002;
- ЛПО31 – 2×31 – 002,

а також їх вітчизняні та зарубіжні аналогів. Застосування світильників без розсіювачів та екрануючих ґрат заборонено. Яскравість світильників загального

освітлення в зоні кутів випромінювання від 50 до 90 градусів з вертикаллю в повздовжній та поперечній площинах має становити не більше ніж 200 кд/м, захисний кут світильників – не менше ніж 40 градусів. Світильники місцевого освітлення повинні мати відбивач, що просвічує, із захисним кутом, не меншим ніж 40 градусів. Слід передбачити обмеження прямої блискості від джерел природного та штучного освітлення. При цьому яскравість світлих поверхонь (вікна, джерела штучного освітлення), що розташовані в полі зору повинна бути не більше ніж 200 кд/м

Необхідно обмежувати відбиту блискість на робочих поверхнях відносно джерел природного і штучного освітлення. При цьому яскравість відблисків на екрані ВДТ має не перевищувати 40 кд/м², а яскравість стелі в разі застосування системи відбитого освітлення – 200 кд/м

Показник осліпленості у разі використання джерел загального штучного освітлення у виробничих приміщеннях має не перевищувати 20, а показник дискомфорту в адміністративно-громадських приміщеннях має бути не більше за 40. Необхідно обмежувати нерівномірність розподілу яскравості в полі зору працюючих з ВДТ. При цьому співвідношення яскравостей робочих поверхонь має бути не більшим ніж 3:1, а співвідношення яскравостей робочих поверхонь та поверхонь стін, обладнання тощо – 5:1. Коефіцієнт запасу для освітлювальних установок загального освітлення має дорівнювати 1,4. Коефіцієнт пульсації має не перевищувати 5%, що забезпечується застосуванням газорозрядних ламп у світильниках загального та місцевого освітлення з ВЧ ПРА для світильників будь-яких типів.

ВИСНОВКИ

В ході виконання роботи було отримано наступні результати:

1. Виконано системний аналіз предметної області оренди нерухомості та типові повсякденні завдання менеджера з оренди нерухомості.
2. Вивчено вітчизняні програмні комплекси автоматизації обліку оренди приміщень.
3. На базі перших двох пунктів було зроблено висновок щодо доцільності розробки мобільного додатку, призначеного для інформаційної підтримки повсякденної роботи менеджера з оренди приміщень:
 - a. мобільний продукт дозволяє отримувати доступ до потрібної актуалізованої інформації щодо орендарів, стану розрахунків з орендарями, тощо;
 - b. мобільний продукт має вбудований планувальник задач, що дозволяє оперувати екземплярами сутності “Орендар”. Такої функціональності неможливо досягти з використанням стандартних планувальників роботи у Android.
 - c. використання локальної проекції загальної інформаційної бази дозволяє забезпечити продовження роботи у випадку відсутності підключення до мережі Інтернет або слабого сигналу стільникової мережі, що унеможливорює обмін даними.
 - d. Розташування основної інформаційної бази на віддаленому сервері надає можливість створення мікросервісу синхронізації з обліковою системою підприємства.
4. Розроблено ескізний та технічний проекти додатку, що полегшує та систематизує його розробку, дозволяє за необхідності залучати додаткових розробників у команду розробки та забезпечує результат, що відповідає очікуванням та технічному завданню.
5. Надано детальний опис взаємодії користувача з системою через діаграму прецедентів та діаграми діяльності.

6. Обґрунтовано вибір засобів розробки та програмного середовища розробки, додаткових інструментів, необхідних для тестування та промислової експлуатації додатку.

7. При розробці архітектури обрано патерн проектування MVC, що повністю реалізовано засобами об'єктно – орієнтованого програмування Java.

8. При розробці архітектури системи обрана клієнт-серверна модель, яка забезпечує можливість одночасного використання інформаційних ресурсів багатьма користувачами за допомогою мобільного додатку

9. Розроблено архітектура системи в цілому, а також архітектури серверної та клієнтської частин.

10. Прийнято рішення, що взаємодія серверної частини з базою даних здійснюється з використанням SQL-запитів, на основі СУБД MySQL, яка є широко розповсюдженою та популярною альтернативою дорогим комерційним СУБД.

11. Проведено програмну реалізацію складових компонентів та системи в цілому. Виправлення помилок програмного коду проводилось за результатами тестування. Програмний продукт готовий до використання.

12. Проаналізовано законодавство щодо охорони праці, виявлено та проаналізовано шкідливі та небезпечні фактори при роботі з персональним комп'ютером, виділено заходи щодо зменшення впливу таких факторів.

Таким чином, в результаті дипломної роботи було розроблено мобільний додаток інформаційної підтримки роботи менеджера з оренди приміщень. Програмний продукт забезпечує зручний мобільний доступ до необхідної інформації з використанням смартфона.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Джеф Раскін, «Інтерфейс. Нові напрямки в проектування комп'ютерних систем».
2. Developers [Електронний ресурс] – Режим доступу: [https:// developer.android.com/](https://developer.android.com/)
3. Metanit.com [Електронний ресурс] – Режим доступу: [https:// metanit.com/java/android/8.1.php](https://metanit.com/java/android/8.1.php)
4. Fandroid.info [Електронний ресурс] – Режим доступу: <https:// android-broadcast-receivers-for-beginners/>
5. Java Rush [Електронний ресурс] – Режим доступу: <https:// javarush.ru/>
6. Филлипс Билл, Марсикано Кристин, Стюарт Крис, Гарднер Брайан Android. Программирование для профессионалов. 4-е издание. [Електронний ресурс] – Режим доступу: <https://books.google.com.ua/>
7. Герберт Шилдт Java. Полное руководство. 10-е издание. – Москва: Диалектика, 2020. – 730 с.
8. Гриффитс Дэвид, Гриффитс Дон Head First. Программирование для Android. – Питер, 2018. – 912 с.
9. Дейтел Пол, Дейтел Харви, Уолд Александер Android для разработчиков. – Питер, 2016. – 512 с.
10. MySQL [Електронний ресурс] – Режим доступу : <https://ru.wikipedia.org/wiki/MySQL>
11. Java Is the Language of Possibilities [Електронний ресурс] – Режим доступу: <https://www.oracle.com/java/technologies/>
12. Мова програмування - Java [Електронний ресурс] – Режим доступу: <https://uk.wikipedia.org/wiki/J%20ava>
13. Програмування під Android [Електронний ресурс]. – Режим доступу: <https://habr.com/ru/post/335332/>

14. Мацяшек Л. Аналіз і проектування інформаційних систем за допомогою UML 2.0, 2020 – Діалектика Вільямс – 816 стор.: іл.

15. Ларман К. Застосування UML 2.0 і шаблонів проектування, 3-є видання– Діалектика Вільямс, 2020 – 736 стор.: іл.

ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ

Вступ

Завданням даної дипломної роботи є розробка мобільного додатку інформаційної підтримки роботи менеджера з оренди приміщень. В таблиці 1 представлено загальні відомості про систему.

Таблиця 1.
Загальні відомості про систему.

Повна назва системи	Мобільний додаток інформаційної підтримки роботи менеджера з оренди приміщень
Умове позначення системи	МД МОП
Розроблювач	Золін Олексій
Замовник	Кафедра інформаційних управляючих систем та технологій

1. ПІДСТАВИ ДЛЯ РОЗРОБКИ.

МД МОП розробляється на підставі завдання для дипломного проектування за спеціальністю 124 – «Системний аналіз».

2. ПРИЗНАЧЕННЯ РОЗРОБКИ.

Об'єкт автоматизації: робота менеджера з оренди приміщень.

Основними завданнями менеджера з оренди приміщень є:

- підбір приміщень для орендарів, узгодження умов та підписання договорів оренди відповідних приміщень;
- контроль надходження платежів за оренду приміщень та компенсації комунальних послуг своєчасно та в повному обсязі;
- забезпечення надання додаткових послуг орендарям.

3. МЕТА СТВОРЕННЯ СИСТЕМИ

Основною метою створення мобільного додатку для інформаційної підтримки роботи менеджера з оренди приміщень є надання доступу для потрібної в роботі інформації з використанням мобільного терміналу. Що в свою

чергу усуває необхідність знаходження безпосередньо на робочу місті в офісі для доступу к інформації.

Отримання необхідної інформації з використанням мобільного терміналу дозволяє мати доступ до інформації безпосередньо під час знаходження у приміщеннях, що здаються в оренду, під час спілкування з орендарями, тощо.

4. ВИМОГИ ДО ПРОГРАМИ

4.1 Вимоги до функціональних характеристик

ІС, що буде спроектована, повинна відповідати наступним вимогам:

Процесно – орієнтовані функціональні вимоги

Перегляд строків закінчення договорів з метою планування нової задачі в оренду або пролонгації договору;

Перегляд заборгованості орендарів: в цілому за договорами та за окремими послугами, що надаються в рамках договорів;

Перегляд поточних справ з визначеними дедлайнами, можливість встановлення нагадування.

Інформаційно – орієнтовані функціональні вимоги

Перегляд списку орендарів, пошук контактної інформації.

Нефункціональні вимоги:

Система повинна працювати та надавати зручний для використання інтерфейс на екрані мобільного телефону.

4.2 Вимоги до надійності

При проектуванні повинні бути використані типові проектні рішення там, де це можливо. На стадії технічного проектування повинні прийматися рішення, що зменшують трудомісткість розробки та експлуатації системи.

4.3 Умови експлуатації

АРМ розроблена на мові програмування Java в якості СКБД використано SQLite та MySQL.

4.4. Вимоги до інформаційної та програмної сумісності

Клієнтський термінал повинен використовувати операційну систему Android не нижче версії 10 (Android X).

Вимоги для роботи мобільного додатку:

Наявність періодичного підключення до глобальної мережі «Інтернет», постійне підключення не є обов'язковою умовою функціонування додатку. Швидкість підключення не менше 512 Кбіт / сек.

Вимоги до віддаленого серверу (VPS):

- Двоядерний процесор;
- Оперативна пам'ять 2GB, рекомендується 8 Gb.
- Ємність накопичувача залежить від розмірів основної інформаційної бази, але не повинна бути менше за 25 Gb.

4.5. Вимоги до технічного забезпечення

Технічні вимоги до клієнтських терміналів визначаються вимогами операційної систем.

5. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Усі регламентуючі документи є елементами МД МОП, які безпосередньо входять до цього. В таблиці 2 наведено склад документів.

Таблиця 2.
Склад документів організаційно-методичного
та експлуатаційного забезпечення.

№	Назва	Стадія
1	Керівництво користувача	Реалізація системи

6. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

В таблиці 3 представлено стадії та етапи по створенню МД МОП.

Таблиця 3
Стадії та етапи робіт по створенню АРМ СВК

№	Стадія	Документи
1	Аналіз предметної області	Звіт з описом предметної області
2	Системне проектування	Технічне завдання
3	Ескізне проектування	Ескізний проект
4	Технічне проектування	Технічний проект
5	Робоче проектування	Робочий проект

ДОДАТОК Б – ІНСТРУКЦІЯ З ВИКОРИСТАННЯ

Мобільний додаток призначено для інформаційної підтримки роботи менеджера з оренди приміщень.

Додаток дозволяє вести перелік справ, пов'язаних з певними орендарями, а також перелік орендарів. У переліку орендарів користувач може знайти певного орендаря за назвою. Під час перегляду орендаря можна отримати інформацію щодо його контактних даних, а також про поточний стан заборгованості за послуги оренди, а також за суму несплаченої компенсації за інші послуги – електроенергію, водопостачання, користування загальними вбиральнями, тощо.

При запуску додатка користувач бачить початковий екран, що показаний на рисунку.

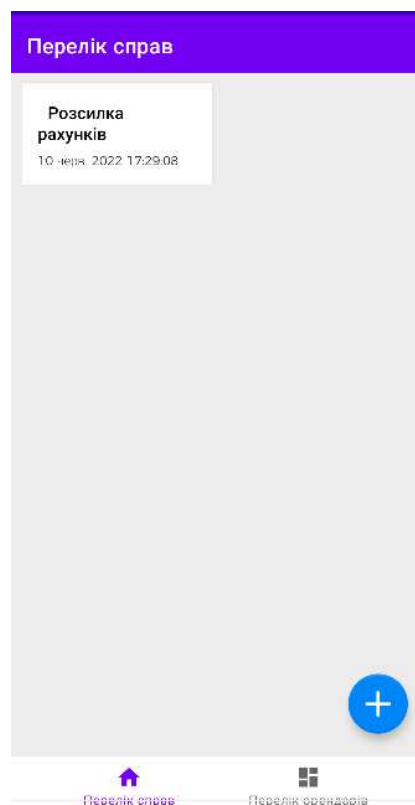


Рисунок Б.1 – Початковий екран додатку.

Екран складається з:

- навігаційної панелі, що розташована у нижній частині екрану;

- переліку поточних справ, кожна з яких представлена окремою плашкою з описом;
- кнопки додавання нової справи, що розташовану в нижньому правому куті.

Шляхом натискання “+” можна додати нову справу. Екран додавання нової справи показано на рисунку Б.2

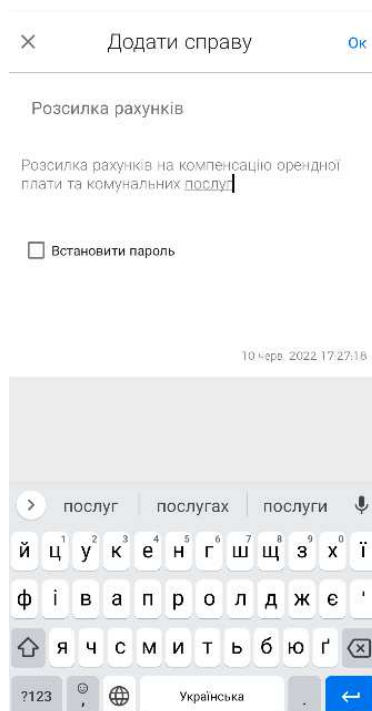


Рисунок Б.2 – Додавання нової справи.

Для зберігання створеної нової справи необхідно натиснути “Ok” в правому верхньому куті. Для скасування створення нової справи треба натиснути “X” в лівому верхньому куті екрану.

Для редагування раніше створеної справи необхідно натиснути плашку з відповідною справою на екрані переліку справ (рисунок Б.1).

При натисканні на плашку із назвою потрібної справи відбувається перехід до її редагування. Екран редагування аналогічний показаному на рисунку Б.2 при заповненні полів нової справи.

Натискання “Ok” зміни будуть збережені, натискання “X” скасовує зроблені зміни.

Для переходу до переліку орендарів необхідно натиснути кнопку “Перелік орендарів” на навігаційній панелі.

При натисканні навігаційної кнопки відбувається перехід до екрану, що показаний на рисунку Б.3.

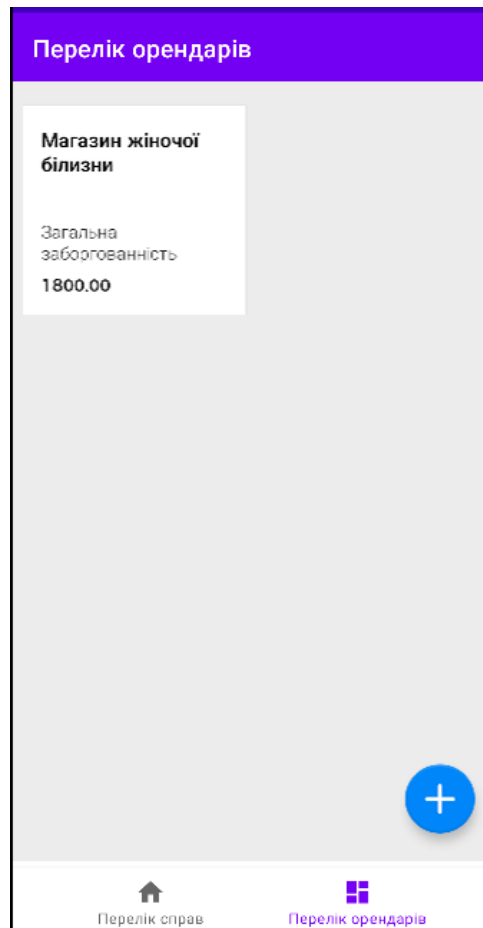


Рисунок Б.3 – Екран переліку орендарів.

На екрані переліку знаходяться плашки з найменування орендарів та сумою загальної поточної заборгованості.

Створення та редагування окремих орендарів відбувається аналогічно створенню та редагуванню справ.

При перегляді або редагування існуючого орендаря відображаються суми заборгованості. Окремо за послугу оренди, окремо компенсації за інші послуги.

Екран редагування/перегляду інформації по орендареві показаний на рисунку Б.4.

×

Перегляд орендаря

Ок

Магазин жіночої білизни

Оксана, +380931932373

За оренду, грн.

1000.00

Компенсація послуг, грн.

800.00

10 черв. 2022 19:41:27

Рисунок Б.4 – Перегляд/редагування орендаря.

При створенні нового орендаря, відповідно, інформація про заборгованості не відображається.

ДОДАТОК В – ТЕКСТ ПРОГРАМИ

MainActivity.java

```
package com.example.adminapp;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;
import androidx.navigation.NavController;
import androidx.navigation.Navigation;
import androidx.navigation.ui.AppBarConfiguration;
import androidx.navigation.ui.NavigationUI;

import com.example.adminapp.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        // BottomNavigationView navView =
        findViewById(R.id.nav_view);
        // Passing each menu ID as a set of Ids because each
        // menu should be considered as top level destinations.
        AppBarConfiguration appBarConfiguration = new
        AppBarConfiguration.Builder(
            R.id.navigation_home,
            R.id.navigation_dashboard)
            .build();
        NavController navController =
        Navigation.findNavController(this,
            R.id.nav_host_fragment_activity_main);
        NavigationUI.setupActionBarWithNavController(this,
        navController,
            appBarConfiguration);
        NavigationUI.setupWithNavController(binding.navView,
        navController);
    }

}
```

TenantFragment.java

```
package com.example.adminapp.ui.dashboard;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

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

import com.example.adminapp.databinding.FragmentDashboardBinding;

public class TenantFragment extends Fragment {

    private FragmentDashboardBinding binding;

    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle
savedInstanceState) {
        TenantViewModel dashboardViewModel =
            new
ViewModelProvider(this).get(TenantViewModel.class);

        binding = FragmentDashboardBinding.inflate(inflater,
container, false);
        View root = binding.getRoot();

        final TextView textView = binding.textDashboard;
        dashboardViewModel.getText().observe(getViewLifecycleOwner(),
textView::setText);
        return root;
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        binding = null;
    }

}
```

TenantViewModel.java

```
package com.example.adminapp.ui.dashboard;
```

```

import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.ViewModel;

public class TenantViewModel extends ViewModel {

    private final MutableLiveData<String> mText;

    public TenantViewModel() {
        mText = new MutableLiveData<>();
        mText.setValue("This is tenant fragment (dummy)");
    }

    public LiveData<String> getText() {
        return mText;
    }
}

```

HomeFragment.java

```

package com.example.adminapp.ui.home;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

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

import com.example.adminapp.databinding.FragmentHomeBinding;
import com.example.adminapp.databinding.NoteListItemBinding;

public class HomeFragment extends Fragment {

    private FragmentHomeBinding binding;

    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle
savedInstanceState) {
        HomeViewModel homeViewModel =
            new
ViewModelProvider(this).get(HomeViewModel.class);

        binding = FragmentHomeBinding.inflate(inflater,
container, false);
        View root = binding.getRoot();

```

```

        final TextView textView = binding.textHome;

homeViewModel.getText().observe(getViewLifecycleOwner(),
textView::setText);
        return root;
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        binding = null;
    }
}

```

HomeViewModel.java

```

package com.example.adminapp.ui.home;

import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.ViewModel;

public class HomeViewModel extends ViewModel {

    private final MutableLiveData<String> mText;

    public HomeViewModel() {
        mText = new MutableLiveData<>();
        mText.setValue("This is home fragment (dummy)");
    }

    public LiveData<String> getText() {
        return mText;
    }
}

```

DaoAccess.java

```

import android.arch.lifecycle.LiveData;
import android.arch.persistence.room.Dao;
import android.arch.persistence.room.Delete;
import android.arch.persistence.room.Insert;
import android.arch.persistence.room.Query;
import android.arch.persistence.room.Update;

import com.an.room.model.Tenant;

import java.util.List;

```

```

@Dao
public interface DaoAccess {

    @Insert
    Long insertTask(Tenant note);

    @Query("SELECT * FROM tenants ORDER BY created_at desc")
    LiveData<List<Tenant>> fetchAllTasks();

    @Query("SELECT * FROM tenants WHERE id =:taskId")
    LiveData<Tenant> getTask(int taskId);

    @Update
    void updateTask(Tenant tenant);

    @Delete
    void deleteTask(Tenant tenant);
}

```

TenantDatabase.java

```

import android.arch.persistence.room.Database;
import android.arch.persistence.room.RoomDatabase;

import com.an.room.dao.DaoAccess;
import com.an.room.model.Tenant;

@Database(entities = {Tenant.class}, version = 1, exportSchema
= false)
public abstract class TenantDatabase extends RoomDatabase {

    public abstract DaoAccess daoAccess();
}

```

Tenant.java

```

import android.arch.persistence.room.ColumnInfo;
import android.arch.persistence.room.Entity;
import android.arch.persistence.room.PrimaryKey;
import android.arch.persistence.room.TypeConverters;

import com.an.room.util.TimestampConverter;

```

```

import java.io.Serializable;
import java.util.Date;

@Entity(tableName = "tenants")
public class Tenant implements Serializable {

    @PrimaryKey(autoGenerate = true)
    private int id;

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

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

    @ColumnInfo(name = "main_payment")
    private float main_payment;

    @ColumnInfo(name = "compensations")
    private float compensations;

    @ColumnInfo(name = "created_at")
    @TypeConverters({TimestampConverter.class})
    private Date createdAt;

    @ColumnInfo(name = "modified_at")
    @TypeConverters({TimestampConverter.class})
    private Date modifiedAt;

    public int getId() {
        return id;
    }

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

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {

```



```

        this.description = description;
    }

    public Date getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }

    public Date getModifiedAt() {
        return modifiedAt;
    }

    public void setModifiedAt(Date modifiedAt) {
        this.modifiedAt = modifiedAt;
    }

    public float getMain_payment() {return main_payment;}

    public void setMain_payment(float paymentSum)
    {this.main_payment = paymentSum;}

    public float getCompensations() {return compensations;}

    public void setCompensations(float compensationSum)
    {this.compensations = compensationSum;}
}

```

TenantRepository.java

```

import android.arch.lifecycle.LiveData;
import android.arch.persistence.room.Room;
import android.content.Context;
import android.os.AsyncTask;

import com.an.room.db.TenantDatabase;
import com.an.room.model.Tenant;
import com.an.room.util.AppUtils;

import java.util.List;

public class TenantRepository {

    private String DB_NAME = "db_tenants";

    private TenantDatabase noteDatabase;
    public TenantRepository(Context context) {

```

```

        noteDatabase = Room.databaseBuilder(context,
TenantDatabase.class, DB_NAME).build();
    }

    public void insertTask(String title,
        String description) {

        insertTask(title, description, false, null);
    }

    public void insertTask(String title,
        String description,
        boolean encrypt,
        String password) {

        Tenant note = new Tenant();
        note.setTitle(title);
        note.setDescription(description);
        note.setCreatedAt(AppUtils.getCurrentDateTime());
        note.setModifiedAt(AppUtils.getCurrentDateTime());
        insertTask(note);
    }

    public void insertTask(final Tenant note) {
        new AsyncTask<Void, Void, Void>() {
            @Override
            protected Void doInBackground(Void... voids) {
                noteDatabase.daoAccess().insertTask(note);
                return null;
            }
        }.execute();
    }

    public void updateTask(final Tenant note) {
        note.setModifiedAt(AppUtils.getCurrentDateTime());

        new AsyncTask<Void, Void, Void>() {
            @Override
            protected Void doInBackground(Void... voids) {
                noteDatabase.daoAccess().updateTask(note);
                return null;
            }
        }.execute();
    }

    public void deleteTask(final int id) {
        final LiveData<Tenant> task = getTask(id);
        if(task != null) {
            new AsyncTask<Void, Void, Void>() {
                @Override
                protected Void doInBackground(Void... voids) {

```

```

noteDatabase.daoAccess().deleteTask(task.getValue());
        return null;
    }
    }.execute();
}

public void deleteTask(final Tenant note) {
    new AsyncTask<Void, Void, Void>() {
        @Override
        protected Void doInBackground(Void... voids) {
            noteDatabase.daoAccess().deleteTask(note);
            return null;
        }
    }.execute();
}

public LiveData<Tenant> getTask(int id) {
    return noteDatabase.daoAccess().getTask(id);
}

public LiveData<List<Tenant>> getTasks() {
    return noteDatabase.daoAccess().fetchAllTasks();
}
}

```

AddTenantActivity.java

```

import android.app.Activity;
import android.content.Intent;
import android.graphics.Color;
import android.graphics.PorterDuff;
import android.graphics.PorterDuffColorFilter;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.AppCompatCheckBox;
import android.text.method.PasswordTransformationMethod;
import android.view.MotionEvent;
import android.view.View;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;

import com.an.room.AppConstants;
import com.an.room.R;

```

```

import com.an.room.model.Tenant;
import com.an.room.util.AppUtils;

public class AddTenantActivity extends AppCompatActivity
implements CompoundButton.OnCheckedChangeListener,
View.OnClickListener,
View.OnTouchListener ,AppConstants {

    private EditText editTitle, editDesc, editPwd;
    private TextView textTime, btnDone, toolbarTitle;
    private ImageView btnDelete;

    private Tenant note;
    private boolean pwdVisible;

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

        textTime = findViewById(R.id.text_time);
        toolbarTitle = findViewById(R.id.title);
        editTitle = findViewById(R.id.edit_title);
        editDesc = findViewById(R.id.edit_desc);

        btnDelete = findViewById(R.id.btn_close);
        btnDelete.setOnClickListener(this);

        btnDone = findViewById(R.id.btn_done);
        btnDone.setOnClickListener(this);

        note = (Tenant)
getIntent().getSerializableExtra(INTENT_TASK);
        if(note == null) {

toolbarTitle.setText(getString(R.string.add_task_title));
            btnDelete.setImageResource(R.drawable.btn_done);
            btnDelete.setTag(R.drawable.btn_done);

textTime.setText(AppUtils.getFormattedDateString(AppUtils.getCurre
ntDateTime()));

        } else {

toolbarTitle.setText(getString(R.string.edit_task_title));
            btnDelete.setImageResource(R.drawable.ic_delete);
            btnDelete.setTag(R.drawable.ic_delete);
            if(note.getTitle() != null &&
!note.getTitle().isEmpty()) {

```

```

        editTitle.setText(note.getTitle());

editTitle.setSelection(editTitle.getText().length());
    }
    if(note.getDescription() != null &&
!note.getDescription().isEmpty()) {
        editDesc.setText(note.getDescription());

editDesc.setSelection(editDesc.getText().length());
    }
    if(note.getCreatedAt() != null) {

textTime.setText(AppUtils.getFormattedDateString(note.getCreatedAt()
));
    }
}

AppUtils.openKeyboard(getApplicationContext());
}

@Override
public void onCheckedChanged(CompoundButton
compoundButton, boolean b) {
    if(b) {
        editPwd.setVisibility(View.VISIBLE);
        editPwd.setFocusable(true);
    } else {
        editPwd.setVisibility(View.INVISIBLE);
    }
}

@Override
public void onClick(View view) {
    AppUtils.hideKeyboard(this);
    if(view == btnDelete) {

        if((Integer)btnDelete.getTag() ==
R.drawable.btn_done) {
            setResult(Activity.RESULT_CANCELED);

        } else {
            Intent intent = getIntent();
            intent.putExtra(INTENT_DELETE, true);
            intent.putExtra(INTENT_TASK, note);
            setResult(Activity.RESULT_OK, intent);
        }

        finish();
        overridePendingTransition(R.anim.stay,
R.anim.slide_down);

```

```

        } else if(view == btnDone) {
            Intent intent = getIntent();
            if(note != null) {

note.setTitle(editTitle.getText().toString());

note.setDescription(editDesc.getText().toString());
                intent.putExtra(INTENT_TASK, note);

            } else {
                intent.putExtra(INTENT_TITLE,
editTitle.getText().toString());
                intent.putExtra(INTENT_DESC,
editDesc.getText().toString());
            }
            setResult(Activity.RESULT_OK, intent);
            finish();
            overridePendingTransition(R.anim.stay,
R.anim.slide_down);
        }
    }

    @Override
    public boolean onTouch(View view, MotionEvent event) {
        return false;
    }
}

```

TenantActivityList.java

```

import android.app.Activity;
import android.arch.lifecycle.Observer;
import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.design.widget.FloatingActionButton;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.StaggeredGridLayoutManager;
import android.view.View;
import android.widget.TextView;

import com.an.room.AppConstants;
import com.an.room.R;
import com.an.room.model.Tenant;
import com.an.room.repository.TenantRepository;
import com.an.room.ui.adapter.NotesListAdapter;
import com.an.room.util.NavigatorUtils;
import com.an.room.util.RecyclerItemClickListener;

```

```

import java.util.List;

public class TenantsListActivity extends AppCompatActivity
implements View.OnClickListener,
RecyclerViewItemClickListener.OnRecyclerViewItemClickListener,
AppConstants {

    private TextView emptyView;
    private RecyclerView recyclerView;
    private NotesListAdapter notesListAdapter;
    private FloatingActionButton floatingActionButton;

    private TenantRepository noteRepository;

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

        noteRepository = new
TenantRepository(getApplicationContext());

        recyclerView = findViewById(R.id.task_list);
        recyclerView.setLayoutManager(new
StaggeredGridLayoutManager(2
StaggeredGridLayoutManager.VERTICAL));
        recyclerView.setOnItemClickListener(new
RecyclerViewItemClickListener(this, this));

        floatingActionButton = findViewById(R.id.fab);
        floatingActionButton.setOnClickListener(this);

        emptyView = findViewById(R.id.empty_view);

        updateTaskList();
    }

    private void updateTaskList() {
        noteRepository.getTasks().observe(this, new
Observer<List<Tenant>>() {
            @Override
            public void onChanged(@Nullable List<Tenant>
notes) {
                if(notes.size() > 0) {
                    emptyView.setVisibility(View.GONE);
                    recyclerView.setVisibility(View.VISIBLE);
                    if (notesListAdapter == null) {

```

```

        notesListAdapter = new
NotesListAdapter(notes);

recyclerView.setAdapter(notesListAdapter);

        } else notesListAdapter.addTasks(notes);
    } else updateEmptyView();
    }
    });
}

private void updateEmptyView() {
    emptyView.setVisibility(View.VISIBLE);
    recyclerView.setVisibility(View.GONE);
}

/*
 * New note to be added
 * */
@Override
public void onClick(View view) {
    Intent intent = new Intent(TenantsListActivity.this,
AddTenantActivity.class);
    startActivityForResult(intent, ACTIVITY_REQUEST_CODE);
}

/*
 * update/delete existing note
 * */
@Override
public void onItemClick(View parentView, View childView,
int position) {
    Tenant note = notesListAdapter.getItem(position);
    NavigatorUtils.redirectToEditTaskScreen(this, note);
}

@Override
protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode == ACTIVITY_REQUEST_CODE && resultCode
== Activity.RESULT_OK) {

        if(data.hasExtra(INTENT_TASK)) {
            if(data.hasExtra(INTENT_DELETE)) {
                noteRepository.deleteTask((Tenant)
data.getSerializableExtra(INTENT_TASK));

            } else {

```



```

        noteRepository.updateTask((Tenant)
data.getSerializableExtra(INTENT_TASK));
    }
    } else {
        String title =
data.getStringExtra(INTENT_TITLE);
        String desc =
data.getStringExtra(INTENT_DESC);
        String pwd = data.getStringExtra(INTENT_PWD);
        boolean encrypt =
data.getBooleanExtra(INTENT_ENCRYPT, false);
        noteRepository.insertTask(title, desc,
encrypt, pwd);
    }
    updateTaskList();
}
}
}

```

TenantListAdapter.java

```

import android.support.annotation.NonNull;
import android.support.v7.util.DiffUtil;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import com.an.room.R;
import com.an.room.model.Tenant;
import com.an.room.util.TenantDiffUtil;

import java.util.List;

public class TenantListAdapter extends
RecyclerView.Adapter<TenantListAdapter.CustomViewHolder> {

    private List<Tenant> notes;
    public TenantListAdapter(List<Tenant> notes) {
        this.notes = notes;
    }

    @NonNull
    @Override
    public CustomViewHolder onCreateViewHolder(@NonNull
ViewGroup parent, int viewType) {
        View view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.note_list_item, null);
    }
}

```

```

        CustomViewHolder viewHolder = new
CustomViewHolder(view);
        return viewHolder;
    }

    @Override
    public void onBindViewHolder(@NonNull CustomViewHolder
holder, int position) {
        Tenant note = getItem(position);

        holder.itemTitle.setText(note.getTitle());
        holder.itemSum.setText("Загальна заборгованність");
        holder.itemOtherSum.setText("1800.00");
    }

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

    public Tenant getItem(int position) {
        return notes.get(position);
    }

    public void addTasks(List<Tenant> newNotes) {
        TenantDiffUtil noteDiffUtil = new
TenantDiffUtil(notes, newNotes);
        DiffUtil.DiffResult diffResult =
DiffUtil.calculateDiff(noteDiffUtil);
        notes.clear();
        notes.addAll(newNotes);
        diffResult.dispatchUpdatesTo(this);
    }

    protected class CustomViewHolder extends
RecyclerView.ViewHolder {

        private TextView itemTitle, itemSum, itemOtherSum;
        public CustomViewHolder(View itemView) {
            super(itemView);

            itemTitle =
itemView.findViewById(R.id.item_title);
            itemSum =
itemView.findViewById(R.id.item_mainsum);
            itemOtherSum =
itemView.findViewById(R.id.item_othersums);
        }
    }
}

```

NavigationUtils.java

```
import android.app.Activity;
import android.content.Intent;

import com.an.room.AppConstants;
import com.an.room.model.Tenant;
import com.an.room.ui.activity.AddTenantActivity;

public class NavigatorUtils implements AppConstants {

    public static void redirectToEditTaskScreen(Activity
activity,
                                Tenant note) {
        Intent intent = new Intent(activity,
AddTenantActivity.class);
        intent.putExtra(INTENT_TASK, note);
        activity.startActivityForResult(intent,
ACTIVITY_REQUEST_CODE);
    }

    public static void redirectToViewNoteScreen(Activity
activity,
                                Tenant note) {
        Intent intent = new Intent(activity,
AddTenantActivity.class);
        intent.putExtra(INTENT_TASK, note);
        intent.addFlags(Intent.FLAG_ACTIVITY_FORWARD_RESULT);
        activity.startActivity(intent);
        activity.finish();
    }
}
```

RecyclerItemClickListener.java

```
import android.content.Context;
import android.support.v7.widget.RecyclerView;
import android.view.GestureDetector;
import android.view.MotionEvent;
import android.view.View;

public class RecyclerItemClickListener implements
RecyclerView.OnItemTouchListener {
    private OnItemClickListener mListener;
    private OnRecyclerViewItemClickListener
recyclerViewItemClickListener;

    public interface OnItemClickListener {
        void onItemClick(View view, int position);
    }

    public interface OnRecyclerViewItemClickListener {
```

```

        void onItemClick(View parentView, View childView, int
position);
    }

    GestureDetector mGestureDetector;

    public RecyclerItemClickListener(Context context,
OnItemClickListener listener) {
        mListener = listener;
        mGestureDetector = new GestureDetector(context, new
GestureDetector.SimpleOnGestureListener() {
            @Override
            public boolean onSingleTapUp(MotionEvent e) {
                return true;
            }
        });
    }

    public RecyclerItemClickListener(Context context,
OnRecyclerViewItemClickListener listener) {
        recyclerViewItemClickListener = listener;
        mGestureDetector = new GestureDetector(context, new
GestureDetector.SimpleOnGestureListener() {
            @Override
            public boolean onSingleTapUp(MotionEvent e) {
                return true;
            }
        });
    }

    @Override
    public boolean onInterceptTouchEvent(RecyclerView view,
MotionEvent e) {
        View childView = view.findViewById(e.getX(),
e.getY());
        if (childView != null && mListener != null &&
mGestureDetector.onTouchEvent(e)) {
            mListener.onItemClick(childView,
view.getChildPosition(childView));
        } else if (childView != null &&
recyclerViewItemClickListener != null &&
mGestureDetector.onTouchEvent(e)) {
            recyclerViewItemClickListener.onItemClick(view,
childView, view.getChildPosition(childView));
        }
        return false;
    }

    @Override
    public void onTouchEvent(RecyclerView view, MotionEvent
motionEvent) {

```

```
        }

        @Override
        public void onRequestDisallowInterceptTouchEvent(boolean
b) {

        }

    }
```