МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»**

Виконав(ла): студент 2 курсу, групи 8.1219-з

спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

освітньої програми інженерія програмного забезпечення
(назва освітньої програми)

О.В.Колоколов
(ініціали та прізвище)

Керівник завідувач кафедри фундаментальної математики, доцент, д.т.н., Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра інженерія програмного забезпечення

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри
програмної інженерії,
к.ф.-м.н., доцент
_____ Лісняк А.О.
(підпис)

« ____ » _____ 2020 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ (СТУДЕНТЦІ)

Колоколов Олександр Володимирович
(прізвище, ім'я та по батькові)

1. Тема роботи Розробка мобільного додатку для управління персоналом

керівник роботи Гребенюк Сергій Миколайович, д.т.н., доцент
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » 05 2020 року № 577

2. Строк подання студентом роботи 30.11.2020

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Перелік задач до розв'язання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз існуючих систем.
2. Основні теоретичні відомості про БД.
3. Огляд проекту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 30.05.2019

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	05.06.2020	
2.	Аналіз літературних та електроних джерел	20.06.2020	
3.	Створення структури додатку	25.09.2020	
4.	Програмна реалізація додатку	18.10.2020	
6.	Оформлення та нормоконтроль	16.11.2020	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	9.12.2020	

Студент _____ О.В.Колоколов
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.М. Гребенюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ _____
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка мобільного додатку для управління персоналом»: 58 с., 2 таблиці, 16 див.рис., 9 джерела, 1 додаток.

БАЗИ ДАНИХ, ДОДАТОК, УПРАВЛІННЯ, СИСТЕМИ КЕРУВАННЯ, ANDROID.

Об’єкт дослідження – процес створення БД

Мета роботи: розробити структуру та програмно реалізувати базу даних персоналу.

Методи дослідження – об’єктно-орієнтованого програмування, реляційні БД.

В кваліфікаційній роботі наводитися аналітичний огляд БД та СКБД. Розглядаються переваги та недоліки застосування тих чи інших СКБД у мобильних додатках. Наводитися порівняння най популярніших СКБД та їх інструменти та можливості. Виконано проектування системи управлінням персоналу на основі принципів роботи з БД та СКДБ за допомогою SQLite. Реалізована базова система керування персоналом яка може бути використана у подальшому для розвитку повноцінного додатку .

ABSTRACT

Master’s qualification paper «Development of a Mobile Application for Personnel Management»: 58 pages, 2 tables, 16 figures, 9 references, 1 supplements.

MANAGEMENT, APPENDIX, ANDROID, DATABASES, MANAGEMENT SYSTEMS.

Object of study – the process of creating a database

Aim of the study: is to develop the structure and programmatically implement the personnel database.

Methods of research – object-oriented programming, relational databases.

The qualification of the work provides an analytical review of the database and DBMS. The advantages and disadvantages of using certain DBMSs in mobile applications are considered. A comparison of the most popular databases and their tools and capabilities. Personnel management system design based on the principles of working with databases and DBMS using SQLite. Implemented a basic personnel management system that can be used in the future to develop a full-fledged application.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД	База даних
РСКБД	Реляційна система керування базами даних
СКБД	Система керування базами даних
HR	Людські ресурси
CRM	Керування відносин з клієнтами
SQL	Язик структурованих запитів

ВСТУП

Розробка сучасних інформаційних систем базується на використанні баз даних, які стали потужним інструментом зберігання різноманітних даних, та створення систем, які аналізують, сортують та відокремлюють ті дані, які необхідні більш усього, роблячи запити та пришвидшуючи пошук інформації. На основі отриманої інформації можна приймати рішення, та будувати різноманітні системи машинного навчання. Зі збільшенням об'ємів діджиталізації все найбільш актуальними є додатки для управління персоналом дозволяючи власнику компанії або підприємства бути освіченими у справах своєї компнії.

Об'єктом дослідження процес управління та відстеження змін даних персоналу.

Предмет дослідження найбільш ефективні способи зберігання даних.

Метою кваліфікаційної роботи є розробка додатка для управління персоналом.

Для досягнення поставленої мети сформовані наступні задачі:

а) провести аналіз існуючих систем;

б) спроектувати архітектуру додатку;

в) розробити додаток на основі аналізу існуючих систем.

Новизна роботи полягає у поєднанні обліку персоналу з фінансовим аналізом у мобільному додатку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Робота з кадрами є одною з складових бізнес діяльності кожного підприємства. Тому у просторі Інтернету існує велика кількість програмам для PC, наприклад, такі як:

- 1С;
- «Зарплата и кадры»;
- «Кадры Плюс»;
- E-Staff.

Існує багато інших програм, які мають як свої переваги, так і недоліки.

В основі керування персоналу лежить облік персоналу, відносно якого можливо доповнити системи модулями, що впроваджують додаткову функціональність. Але створити такий комплекс програм дуже важко в силу ускладнення механізму роботи з кадрами та вартості «важких» програм може бути за високою для деяких підприємств. Тому в більшості випадків системи розділяють за функціоналом для специфічних завдань та цілей. Як приклад, можна привести таке.

Програмний пакет «1С: Зарплата та керування персоналом» – це комплекс, який дозволяє автоматизувати практично весь бізнес – процеси, пов’язані з кадровим обліком та нарахуванням заробітної плати, облік податків, кадрова документація. Ця система розрахована на великі компанії. Програм є модулем и легко інтегрується з іншими програмними продуктами компанії 1С. Дозволяє реєструвати спеціальні облікові записи для керівників та для роботи з декількома юридичними лицями одночасно [8].

Реалізація програми «Зарплата і кадри» схожа за функціями з реалізаціями програми «1С: Зарплата та керування персоналом», але на відміну від останньої програму «Зарплата і кадри» розраховано на малі підприємства. Для розгортання системи потрібен лише один ПК [8].

Програмний пакет «E-Staff» на відміну від інших прикладів є представником рішення для HR – спеціалістів та рекрутенгу – і за своєю суттю це CRM – система, яка спеціалізується на управлінні взаємовідношеннями з клієнтом. Система дозволяє планувати співбесіди, розсилати листи кандидатам, облік кандидатів та вакансій і т.д. У результаті роботи з цією програмою рекрутер отримує повну інформацію про кандидата у виді звіту. Ця програма дозволяє звільнити HR-менеджера від рутинної роботи з купою даних та резюме, підбираючи кандидата, який відповідає всім вимогам компанії. Також існують модулі цієї програми які допомагають аналізувати ситуації за вакансіями, ефективність роботи рекрутерів та інше [9].

1.1 Аналіз баз даних

Отже, керуючись аналізом можливостей існуючих систем керування персоналом, для реалізації проекту обираємо функціонал, який буде реалізовано. У проекті буде реалізовано облік персоналу, який дозволить створити базу даних персоналу з обліком заробітної плати, посади та аналізу заробітний плати персоналу за посадою, середню заробітну плату із використанням візуального представлення необхідної інформації (графіки тощо).

Бази даних (БД) – це уявлення даних у формі сукупності об’єктів самостійних матеріалів, які систематизовані таким чином, щоб необхідний матеріал було можливо знайти та обробити за допомогою комп’ютера або фізичного документування.

БД повинні не лише мати можливість розташовувати безліч об’єктів у своїй системі, а й також дозволяти керувати цими даними. Ці системи називають СКБД.

З кожним роком кількість інформації, яку необхідно обробляти, зростає, та постає питання про ефективну обробку та каталогізацію інформації. Таким чином, застарілі засоби зберігання інформації, такі як фізичне документування, втрачають свою актуальність виходячи за межі тієї кількості інформації, яку може обробляти людський мозок.

На даний момент найбільш популярним способом обробки даних є комп'ютеризований спосіб. На даний момент такі системи використовуються у фінансовій сфері, торгівлі, промисловості та інших галузях.

Таким чином, створення класифікацій баз даних (див. рис. 1.1) має найбільшу актуальність бо від кількості обробленої інформації залежить прибуток компанії, і де втрата невеликої кількості інформації може призвести до великих втрат, як прибутку, так і темпу росту компанії. Класифікацію баз даних виконують за різними ознаками, такими як: спосіб зберігання інформації, спосіб доступу до БД, сфера застосування, структура організації баз даних.

За способом зберігання БД розділяються на централізовану, розподільну, локальні.

Централізовані БД – це БД, які зберігаються у одному місці. Такі БД знаходяться на одному носії у вигляді одного інформаційного масиву. Централізована база даних доступна лише для одного користувача, таким чином, виключаючи одночасний доступ до бази декількома користувачами. Недолік такої системи це необхідність передачі великого потоку даних та низький ступінь надійності. До переваг можна віднести можливість швидко коректувати дані у базі даних.

Розподільні БД – це БД, які фізично розподілені за взаємопов'язаними локальними ресурсами даючи змогу використовувати базу даних різними користувачами.

Локальні БД – це БД, котрі розташовані безпосереднє на комп'ютері користувача. Такі БД використовуються у випадках, якщо база даних невелика, або комп'ютер здатен її обробити без великих затрат ресурсів, або

як тимчасове сховище для даних на час відсутності зв'язку з сервером БД (після підключення до сервера всі файли переносяться до головної БД).

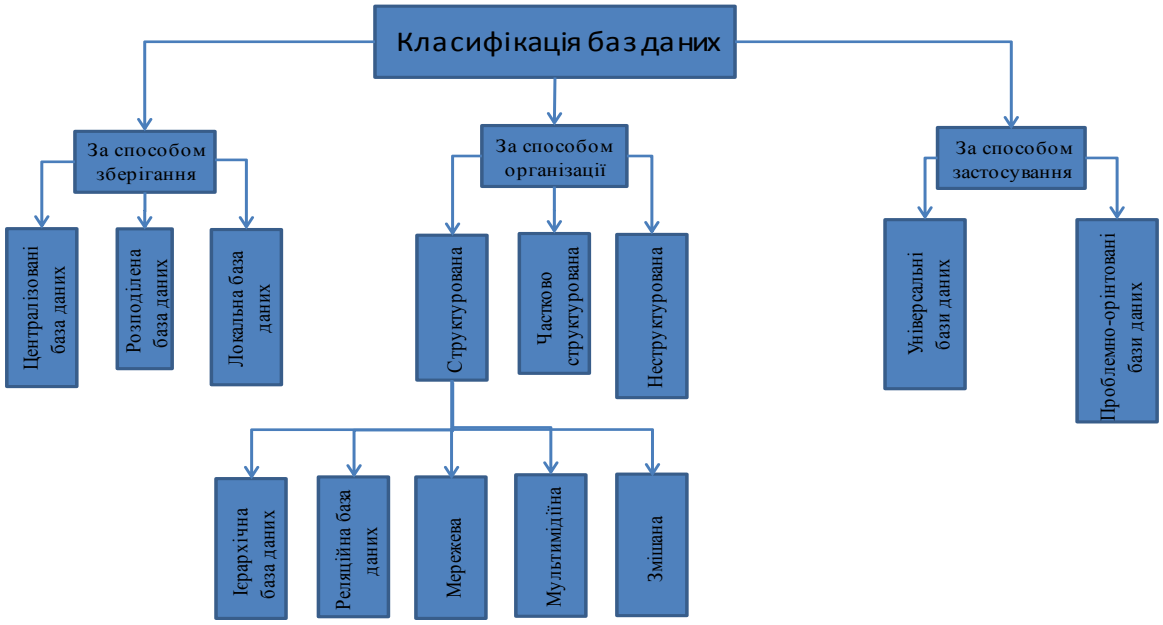

Рисунок 1.1 – Схема класифікації баз даних

За характером організації бувають структуровані, частково структуровані та неструктуровані. *Структуровані БД* є найбільш складним типом організацій та вимагають планування структури з урахуванням всіх факторів, планування розширення та добудови схеми структури. *Частково структурована база даних* – це БД, які структуровані у вигляді звичайного тексту або гіперпосилання. *Неструктурована база даних*, що виконана у вигляді семантичної мережі, у якій концептуально ця структура може перетворитися в семантичну павутину за допомогою Інтернету, створюючи з неї базу знань глобального масштабу.

Структуровані БД поділяються на ієрархічні, реляційні, мережеві, мультимедійні та змішані.

Ієрархічна база даних – це база даних, яка графічно може бути представлена як дерево, яке складається з різних рівнів. На верхньому рівні

знаходиться тільки один об’єкт, на другому два рівня і т.і. Між об’єктами, котрі розташовані на різних рівнях існують горизонтальні зв’язки. Усі об’єкти можуть містити у себе декілька об’єктів, які розташовані нижче рівня даного об’єкта. Такі об’єкти знаходяться у відношеннях «предок» до «нащадка», при цьому можлива ситуація, коли об’єкт «предок» не має «нащадка» чи може мати декілька у тій ситуації, коли об’єкт «предок» обов’язково повинен мати лише одного «нащадка». Ієрархічна база даних має кореневу папку, поступово розгортуючись до низу. Прикладом такої БД може бути файлова система, така база даних ефективно використовується для виконання різних операцій над даними, які зберігаються у пам’яті комп’ютера. Ієрархічна модель має недолік: це дуже «важка» структура, що ускладнює проектування логічних зв’язків між об’єктами [4].

Реляційні БД – це база даних, яка представляє собою сукупність взаємозв’язаних таблиць, кожна з котрих містить інформацію про об’єкт даного типу. Таблиця складається з рядків, які містять данні, стовбців, які є атрибутами. Таблиці у реляційній базі даних мають низку властивостей: в таблиці не може бути двох однакових рядків, у таблиці може не бути жодного рядка, але обов’язково повинен бути один стовбець, стовбці не залежать один від одного, усі значення у стовбці мають один тип даних, кожна таблиця повинна мати первинний ключ. Первинний ключ – це поле чи комбінація полів таблиці, що ідентифікує кожен рядок таблиці індивідуально. Ключ може складатися з декількох полів таблиць, такий ключ називають складеним. Таким чином, ключ дозволяє реалізувати процес пошуку та упорядкування інформації БД. Таблиця в реляційній базі даних повинна відповідати вимогам нормалізації форми, яка дозволяє виключити можливість дублювання даних та суперечливі дані [4].

Отже, до основних плюсів використання реляційних баз даних є: моделі даних, що зберігаються, надаються у найбільш простих та зрозумілих для користувача формах, у основі баз даних лежить розвинений математичний апарат, котрий дозволяє доступно описати основні операції,

які виконуються над даними, при маніпулюванні та доступі до даних використовують мову не процедурного типу, маніпулювання даними на рівні вихідної інформації та можливість динамічної зміни даних.

Однак така система має великий недолік при розширені меж модулювання інформаційних систем, на етапі проектування потрібно закласти можливості розширення, також є низка проблем, таких як трудомісткість їх проектування і розробки, повільний доступ до даних, проблеми з модулюванням та реалізацією складних зв'язків між даними. Результатом запиту до БД часто є надання логічного висновку на основі даних, що зберігаються.

Мережева база даних – це база даних, яка є узагальненою ієрархічною базою даних за рахунок дозволу мати об'єкту більш ніж одного «нащадка». Прикладом мережевої БД, може бути представлені у інтернеті гіперпосилання, що пов'язують між собою велику кількість документів у єдину розподільну мережеву базу даних, доступ до котрої можливий з будь-якого робочого місця користувача [4].

Мультимедійні БД – це БД, які зберігають мультимедійну інформацію. Основною відмітністю цих баз є спосіб зберігання та обробки інформації не лише у вигляді чисел, символів та масивів інформації, а й розширення на такі дані як документи, відео, звукозаписи та інше. Головним недоліком є те, що для мультимедійних даних баз даних є проблемою чітке розділення програми та даних, це потребує спеціальних методів обробки даних, їх пошуку та інших маніпуляції [4].

За сферою застосування поділяються на універсальні та проблемо-орієнтовані БД.

Універсальні БД – це база даних, яка призначена для розв'язання універсальних задач. До основних задач можна віднести забезпечення збереження інформації, отримання даних, дублювання даних, забезпечення цілісності БД.

Проблемо-орієнтовані БД – це БД, які містять тематично зв’язані документи і дані, призначені для розв’язання прикладних задач певного виду визначеного предмета.

1.2 Системи керування базами даних

Система керування базами даних – це програмного-апаратний комплекс інтегрованої сукупності даних, необхідних для створення, зберігання, використовування.

Системи управління базами даних (див. рис. 1.2) надають користувачу можливість маніпулювати даними, філірувати дані, використовувати різні обчислення, використовування інтерфейсу вводу та виводу, надання візуальної інформації. Дані функції системи реалізуються за допомогою спеціальних мов маніпулювання даними, які входять у склад даних систем управління даними або за допомогою графічного інтерфейсу.

Основні функції системи управління базою даних полягають в управлінні даними у зовнішній і оперативної пам'яті, журналізації змін даних, в забезпеченні резервного копіювання та відновлення БД після збоїв, тобто некоректного завершення роботи з нею, а також у підтримці мов маніпулювання даними, які призначені для роботи з інформацією, що зберігається в базі даних.

Важливою властивістю СКБД є можливість забезпечити два незалежних один від одного виду представлення БД для користувача, які є логічним уявленням даних та фізичним виглядом. Впровадження фізичної незалежності даних дає можливість змінити способи організації БД у пам’яті персонального комп’ютера, й не викликає необхідності зміни логічного уявлення наданих даних. Таким чином, головною цілю реалізацій даних функцій є забезпечення достовірності інформації. Для забезпечення

достовірності даних в системі є незалежні обмеження цілісності, які у деяких випадках не дають ввести у базу даних недостовірну інформацію. Тобто, у всіх СКБД виконується перевірка типу введених даних. Система не дозволяє ввести символ у поле числового типу, вести недопустиму дату.

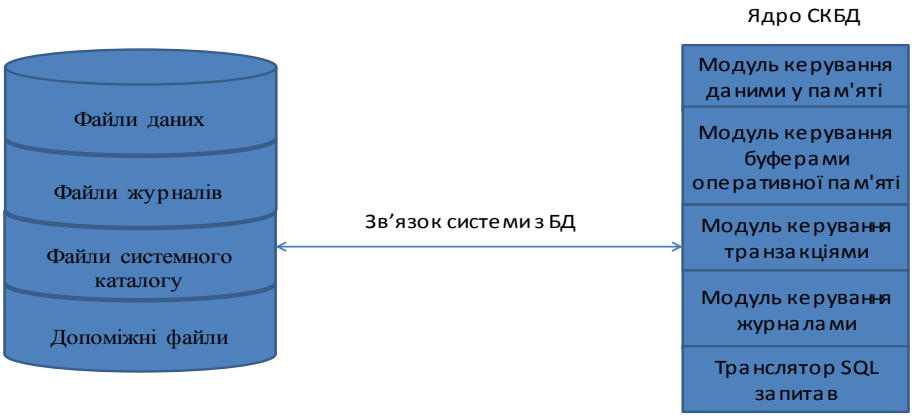

Рисунок 1.2 – Схема зв'язку СКБД

Захист фізичної цілісності полягає в тому, що при роботі комп'ютера можливі збої в його функціонуванні, пошкодження машинних носіїв даних, в наслідок чого можуть бути порушені зв'язки між даними, що призводить до неможливості подальшої роботи з базою даних. Системи управління базами даних мають інструменти, які дозволяють відновити базу даних з резервної копії та відновити її працездатність.

Різні користувачі можуть мати абсолютно різні повноваження по роботі з даними, тобто, деякі дані повинні бути недоступні певним користувачам у вигляді заборони на процес оновлення даних. В СКБД передбачається механізм розмежування повноважень доступу користувачів, заснований на принципах паролів, або на описі повноважень користувачів. Синхронізація роботи декількох користувачів достатньо часто може мати місце у ситуаціях, коли кілька користувачів одночасно виконують операцію поновлення одних і тих же даних. Таке оновлення даних може привести до порушення логічної цілісності, тому система повинна передбачати заходи,

які не допустять оновлення даних іншим користувачем, поки працює з цими даними інший користувач й поки він повністю не закінчить роботу з ними. Найбільш використовуваним тут поняттям є «блокування». Блокування оновлення даних необхідне для того, щоб заборонити різним користувачам можливість одночасно працювати з базою даних, так як це може привести до серйозних помилок.

Управління ресурсами середовища зберігання. База даних розташовується у зовнішньої пам'яті комп'ютера. При роботі з базою даних в неї заносяться нові дані (займається пам'ять) і видаляються дані (звільняється пам'ять). Системи управління базами даних виділяють ресурси пам'яті для нових даних, перерозподіляють звільнилася пам'ять, організовують ведення черги запитів до зовнішньої пам'яті.

При експлуатації БД може виникати необхідність зміни параметрів системи управління базою даних, вибору нових методів доступу, зміни структури збережених даних, а також виконання ряду інших загальносистемних дій. Система управління базами даних надає можливість виконання цих та інших дій для підтримки діяльності БД обслуговуючому її системному персоналу, званого адміністратором БД

За способом доступу до БД системи управління базами даних класифікуються на файл-серверні, клієнт-серверні і вбудовані.

У *файл-серверних СКБД* файли даних зберігаються централізовано в одному місці на робочому файл-сервері. СКБД розташовується на кожному клієнтському комп'ютері. Доступ системи управління базою даних до даних здійснюється через локальну мережу. Синхронізація читань і оновлень здійснюється за допомогою файлових блокувань. Істотною перевагою даної архітектури є низьке навантаження на процесор файлового сервера. До недоліків відносяться потенційно високе завантаження локальної мережі, складність або неможливість централізованого управління. Такі системи використовуються найчастіше в локальних додатках, які застосовують функції управління базою даних, а також в системах з низькою інтенсивністю

обробки даних і низькими піковими навантаженнями на базу даних. На даний момент файл-серверна технологія вважається застарілої, а її використання в великих інформаційних системах є істотним недоліком. До файл-серверних систем управління базами даних можна віднести Microsoft Access, Paradox, dBase, FoxPro, Visual FoxPro.

Клієнт-серверна СКБД розташовується на сервері разом з базою даних. Доступ до БД здійснює безпосередньо в монопольному режимі, тобто робота з базою даних здійснюється одним користувачем. Усі запити на обробку даних обробляються клієнт-серверної системою управління централізовано. Недолік клієнт-серверних СКБД є підвищені вимоги до обладненя серверу. Перевагами даних систем це зручність централізованого управління, забезпечення таких важливих характеристик як висока надійність, висока доступність, низьке завантаження локальної мережі і висока безпека. До клієнт-серверних СКБД відносяться Oracle, Firebird, Interbase, Informix, MS SQL Server, My SQL.

Вбудована система управління базами даних – це система, яка може поставлятися як складова частина деякого програмного продукту, не вимагаючи процедури самостійної установки БД. Вбудована СКБД призначена для локального зберігання даних, та нерозрахована на колективне використання в мережі. Фізично вбудована система управління базою даних частіше всього реалізована у вигляді підключення до бібліотеки з даними. Доступ до даних здійснюється через SQL запити або через спеціальні програмні інтерфейси. До вбудованих систем управління базами даних відносяться OpenEdge, BerkeleyDB, Microsoft SQL Server Compact.

На підставі вищевикладеного теоретичного матеріалу по базах даними можна зробити наступні висновки. В історії розвитку і становлення баз даних, що тривала більше п'ятдесяти років, виділяється чотири основних етапи. Кожен етап характеризується появою і розвитком нових інформаційних технологій в даній області. Історія розвитку база даних починається з появою файлів і файлових систем, що дозволило створювати,

редагувати, копіювати і переміщати файли в пам'яті персонального комп'ютера. В даний час файлові системи входять в структуру будь-якої операційної системи. Перший етап історії розвитку баз даних пов'язаний з появою баз даних на великих електронних обчислювальних машинах. Другий етап характеризується появою персональних комп'ютерів, що послужило поштовхом для розвитку настільних систем управління базами даних. Настільні системи управління базами даних дозволили в упорядкованому вигляді зберігати великий обсяг інформації, мали зручний, призначений для користувача, інтерфейс, який дозволяв заповнювати базу даних і генерувати різні звіти по заданим критеріям. Третій етап – розподільні, розраховані на багато користувачів, БД, які зберегли в собі всі переваги попереднього етапу і в той же час дозволяли організувати паралельну обробку інформації і підтримку цілісності БД. Четвертий заключний етап характеризується появою нової технології доступу до даних – інтернет. Основна перевага полягає в тому, що відпадає необхідність використання спеціалізованого програмного забезпечення, для роботи з віддаленою базою даних використовується стандартний браузер для доступу в інтернет. Різноманітність характеристик і видів баз даних породжує різноманіття їх класифікації. Класифікація баз даних може бути проведена за різними ознаками, які належать до різних компонентів і сторонам їх функціонування. До даних ознаками можна віднести характер інформації, що зберігається, спосіб зберігання даних, структуру організації даних, спосіб доступу до даних, а також сферу застосування. В адмініструванні баз даних передбачається, що їх створення, підтримка і забезпечення доступу користувачів до даних, що зберігаються здійснюються централізовано за допомогою спеціального програмного інструментарію – систем управління базами даних. Якщо прагнути класифікувати існуючі області застосування баз даних, а також дати оцінку перспективам їх розвитку в найближчим часом, то можна отримати перелік найбільш поширених видів, які знайшли застосування у всіх областях людської життєдіяльності. Тенденції еволюції

сучасних інформаційних технологій привели до того, що БД стали однією з найбільш популярних тем при вивченні автоматизованих інформаційних систем. В останні десятиліття сплеск популярності інтернету і стрімкий розвиток новітніх технологій для його використання зробили знання технологій баз даних для багатьох одним з актуальних шляхів розвитку кар'єри. Проектування і розробка баз даних вимагає і мистецтва, і вміння. Розуміння призначених для користувача вимог і перетворення їх в реальний і ефективний проект БД можна назвати творчим процесом, а перетворення цих проектів в реальні БД за допомогою функціональних і високопродуктивних додатків – інженерним процесом. Сучасні БД є основою численних автоматизованих інформаційних систем і знаходять застосування в широкому спектрі додатків і галузей. В даний час триває активне вивчення методів ефективної обробки баз даних з точки зору вилучення з них додаткових знань, так як область застосування баз даних з кожним днем збільшується, і вони застосовуються всюди, де є необхідність в актуальній інформації та швидкого доступу до неї. Бази даних здатні поліпшити продуктивність праці співробітників, а також істотно прискорити процес обробки інформації та складання звітів за різними критеріями. Для керування та маніпулювання даними в СКБД використовують декларативну мову програмування SQL яка дозволяє створювати, модифікувати та керувати даними. Основні можливості SQL це:

- створення у базі даних таблиці;
- додавання у таблицю нових записів;
- редагування записів;
- видалення записів;
- вибірка записів з таблиць;
- редагування структури таблиць.

1.3 Порівняння SQL та NoSQL

Коли обирають СКБД головне питання це вибір структури буде це SQL або NoSQL.

Структура SQL використовують усі РСКБД за для обробки даних. SQL достатньо гнучка і дуже поширена як мова запитів SQL дозволяючи структурувати усі запити робити вибірки видаляти дані та редагувати. Однак для повної реалізації РСКБД потрібно створювати структуру документу заздолегіть. Таким чино зміна у системі може буду критичної якщо вона не була внесена як можливість для розширення.

Структура NoSQL це більш динамічний підхід оскільки дані можуть зберігатись у декілька способів: орієнтовані за колонками, документо-орінтовані, графи, пара «ключ-значення». Таким чином непотрібно будувати структуру проекту, кожен документ може мати свою структуру, у кожної БД може бути властниї синтаксис, дає можливість додавати поля під час роботи з даними. *Масштабування* – властивість системи, мережі або процесу вправлятись з підвищенням навантаження.

SQL БД дозволяє масштабування але лише вертикальне тобто має можливість підвищити навантаження на виділений сервер, нарощуючи потужність центрального процесору, об’єму ОЗУ або системи зберігання даних.

NoSQL горизонтально масштабуються тобто при підвищені кількості використовуємого трафіку є можливість розподілити його або додати більше серверів до СКБД.

Таким чином в залежності від потреб є можливим обрати ту чи іншу структуру БД.

Таблиця 1.3 – Порівняння властивостей SQL та NoSQL

	SQL	**NoSQL**
Тип	Реляційна	Не реляційна
Дані	Структуровані дані зберігаємі у таблиці	Не структурована зберігання даних у вигляді JSON
Схема	Статична	Динамічна
Масштабування	Вертикальна	Горизонтальна
Мова	Структуровані Query запити	Не структуровані Query запити
Транзакціоні системи	Рекомендовані для транзакціоних систем	Не підходять для транзакціоних систем
Гнучкість	Жорстка схема яка прив’язана до відношень таблиць	Не жорстка схема

1.4 Аналіз СКБД які використовуються у Android

Одними з най поширеніших СКБД для мобільних додатків є SQLite, MySQL, PostgreSQL.

Одної з СКБД є SQLite для Android однак поступово втрачає свою популярність у наслідок наявності появи Fairbase яка зберігає дані у хмарному сервісі.

SQLite - вбудована БД написана на C. Будучи кросплатформенною вона підтримує повний набір команд SQL. Коли додаток використовує SQLite їх зв'язок викинуться безпосереднє завдяки функціональних та прямих викликів, які містять дані. Більшість механізмів баз даних SQLite використовують жорстку статичну типізацію. При статичній типізації типу

даних значення визначається його контейнером – стовбцем у котрому зберігається значення [4]. SQLite підтримує наступні типи даних:

- NULL – порожнє значення об’єкту;
- INTEGER – ціле число, зберігається у 1,2,3,4,6 або 8 байтах.
- REAL – ціле число с плаваючим знаком , зберігається у 8-байтах;
- TEXT – текстова строка з кодуванням UTF-8, UTF-16BE, UTF-16LE.
- BLOB – тип даних зберігаючиїся у тому вигляді у якому був надана у базу даних.

SQLite є досить поширеною але придатна лише для низки випадків наприклад:

- вбудована база даних до додатків які не є портами та не розраховані на масштабування додатку – мобільні ігри та додатки одного користувача;.
- випадки коли потрібно збільшити продуктивність додатку які дуже часто використовують прямі операції запису та читання приводячи додаток на SQLite;
- випадки коли потрібно головною функцією додатку є тестування бізнес-логіки.

Переваги SQLite:

- SQLite є файловою БД а тому усі дані зберігаються у одному файлі, полегшуючи переміщення баз даних;
- SQLite використовує SQL.

Недоліки SQLite:

- відсутність управління користувачем, користувачі не мають змоги керувати зв’язками у таблиці;
- не може бути налаштоване у ручну.

Таким чином SQLite можливо використовують лише у випадках коли доступ до баз даних здійснюється лише одним користувачем та коли база

відносно мала бо зі збільшенням об’єму даних продуктивність БД зменшується.

Отже оскільки SQLite непридатна для великих об’ємів інформації використовують більш «важкі» СКБД найбільш популярними є MySQL, PostgreSQL та Fierbase але кожна з них має свої переваги і недоліки .

MySQL – реляційна СКБД з відкритим вихідним кодом з моделлю клієнт-сервер. Розробка MySQL виконується корпорацією Oracle, MySQL э рішенням для малих та середніх додатків. MySQL дозволяє обробляти великий об’єм інформації. MySQL підтримує наступні типи даних:

- TINYINT – дуже мале ціле число;
- SMALLINT – мале число;
- MEDIUMINT – ціле середнего розміру;
- INT або INTEGER – ціле нормального розміру;
- BIGINT – велике ціле;
- FLOAT – число з плаваючим знаком одинарної точності;
- DOUBLE, DOUBLE PRECISOIN, REAL – число з плаваючим знаком подвійною точності
- DECIMAL, NUMERIC – число з плаваючим знаком;
- DATE – дата;
- DATETIME – комбінація дати та часу;
- TIMESTAMP – відмітка часу;
- TIME – час;
- YEAR – рок у форматі УУ або УУУУ;
- CHAR – строка фіксованого розміру яка доповнюється пробілами;
- VARCHAR – строка змінної довжини;
- TINYBLOB,TINYTEXT – стовбці довжиною 255 символів;
- BLOB,TEXT – стовбці довжиною 65535 символів;
- MEDIUMBLOB,MEDIUMTEXT – стовбці довжиною 16777215 символів;

- LONGBLOB, LONGTEXT - стовбці довжиною 4294967295 символів;
- ENUM – перечислення;
- SET – множина;

Переваги MySQL:

- простота та наявність інструментів для візуалізації БД;
- MySQL підтримує більшість функції SQL;
- дозволяє маніпулювати великими об’ємами даних та дозволяє масштабувати додатки;
- вбудовані системи безпеки БД;
- продуктивна робота за вдяки налаштуванням.

До недоліків можна віднести наступне:

- ненадійнйність у процесах роботи з деякими даних;
- так як MySQL є має відкритий код то розробка їде гараздо повільніше ніж у комерційних БД.

Таким чином можна зробити висновок, що MySQL використовується тоді коли функції SQLite не достатьне для реалізації проекту. Також MySQL може забезпечити надійну безпеку для використання БД та використань БД у веб-ресурсах. Гнучка система налаштувань дозволяє покращити роботу БД у разі специфічних потреб. Однак MySQL не можливо використовувати у випадках коли системі потрібно одночасно читати та записувати інформацію або мігрувати з одної РСКБД до іншої.

PostgreSQL також є РСКБД як і MySQL але відокремо від MySQL PostgreSQL відповідає усім стандартам ANSI/ISO. Також PostgreSQL відокремлюється від усіх РСКБД тим що вони також мають об’єктно орінтованиї функціонал з підтримкою концепції ACID(Atomicity, Consistency, Isolation, Duration). Таким чином PostgreSQL може обробляти декілька завдань одночасно , до тогож PostgreSQL має багато додаткових бібліотек.

- Bigint – 8-байтне ціле;

- Bigserial – автоматичне інкремінтоване 8-битне ціле;
- bit [(n)] – бітова строка фіксовноної довжини;
- bit varying [(n)] – бітова строка змінної довжини;
- boolean – буліновска величина;
- box – прямокутник на плоскості ;
- bytea – бінарні дані;
- character varying [(n)] – строка символів фіксованої довжини;
- character [(n)] – строка символів змінної довжини;
- cidr – мережевий адрес IPv4 або IPv6;
- circle – коло на плоскості;
- date – календарна дата;
- double precision – число з плаваючим знаком подвійной точності;
- inet – адрес хоста IPv4 або IPv6;
- integer – 4-байтне ціле;
- interval [fields] [(p)] – часовий проміжок;
- line – нескінченна пряма на плоскості;
- lseg – відрізок на плоскості;
- macaddr – MAC-адрес;
- money – грошова величин;
- path – геометричний шлях на плоскості;
- point – геометрична точка на плоскості;
- polygon – багатокутник на плоскості;
- real – число з плаваючим знаком одинарної точності;
- smallint – 2-байтне ціле;
- serial – автоматичне інкрементування 4-бітне ціле;
- text – строка символів зміної довжини;
- time [(p)] [without time zone] – час доби (без часового пояса);
- time [(p)] with time zone: час доби (з часовим поясом);
- timestamp [(p)] [without time zone] – дата та час (без часового пояса);

- timestamp [(p)] with time zone – дата та час (з часовим поясом);
- tsquery – запит текстового пошуку;
- tsvector – документ текстового пошуку;
- txid_snapshot – снэпшот ID користувача транзакуї;
- uuid – унікальний індетефікатор;
- xml – XML-дані.

До переваг PostgreSQL можна віднести:

- повна SQL сумісність;
- не зважаючи на open sours у PostgreSQL велике комьюнеті та цілодобова підтримка ;
- підтримка сторонніми організаціями, які використовують багато функції PostgreSQL;
- за рахунок зберігаємих процедур підтримує розширення;
- об'єктно-орінтоване;

До недоліків відносимо:

- низка продуктивність на читані;
- складність використання;
- проблеми з хостингом;

Таким чином PostgreSQL забезпечує цілісність даних надійність, забезпечення стабільної роботи при обробці складних процедур та також на відміну від MySQL легше забезпечити міграцію даних на інші платформи.

Fairbase це платформа для розробки мобільних додатків яка позіціанує себе як Backend as a Service який дає можливість використовувати цю платформу як сервер, базу даних, хостінг, аунтефікацію. Додаток підключається до БД через WebSocket який відповідає за синхронізацію даних у період усего сеансу. Fairbase є NoSQL баз даних з витягаючими з цего недоліками. Таким чино Fairbase добре використовувати для додатків з незначною інформацією бо зі збільшенням об'єму даних складність обробки даних підвищується, у такому разі Fairbase складно використовувати як Backend у силу того що складність фільтрування даних ускладнюється.

Також Fairbase має низку переваг дозволяючи зберігати дані у хмаровому сервісі

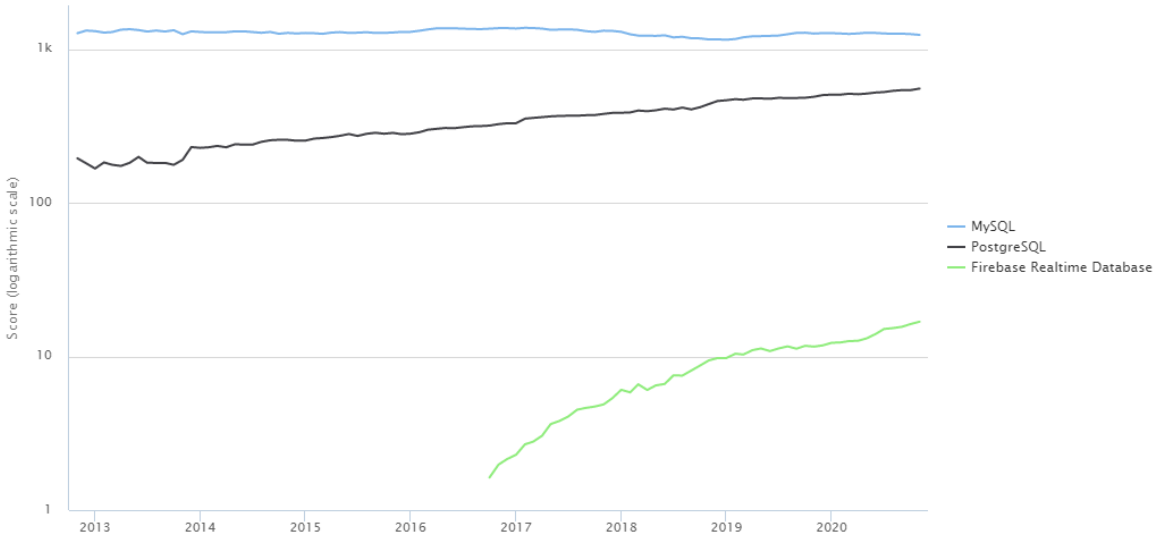

Рисунок 1.4 – Порівняння популярності СКБД

Таблиця 1.4 – Порівняння СКБД

Ім'я	**Firebase Realtime Database**	**MySQL**	**PostgreSQL**
Описання	Хмарне сховище документів у реальному часі. Клієнтами IOS, Android та JavaScript використовують один екземпляр БД автоматично отримуючи оновлення даних	Open source РСКБД	Open source РСКБД
Основна модель БД	Документа	РСКБД	РСКБД
Ліцензія	Комерційна	Open source	Open source
Хмарне сховище	Так	Ні	Ні
Влаштовані мови	-	C або C++	C

Продовження таблиці 1.4

Серверна операційна система	Хост	FreeBSD Linux OS X Solaris Windows	FreeBSD, HP-UX Linux, NetBSD OpenBSD, OS X Solaris, Unix Windows
Схема даних	Не структурована	Структурована	Структурована
Підтримка XML	ні	так	так
SQL	ні	так	так
API та інші методи доступу	Android iOS JavaScript API RESTful HTTP API	ADO.NET JDBC ODBC	ADO.NET JDBC влаштована C бібліотека ODBC streaming API для великих об’єктів
Підтримувані мови програмування	Java JavaScript Objective-C	Ada, C, C#, C++ D, Delphi, Eiffel Erlang, Haskell Java, JavaScript (Node.js), Objective-C, OCaml, Perl, PHP Python, Ruby Scheme, Tcl	.Net, C, C++, Delphi, Java, JavaScript (Node.js), Perl PHP, Python, Tcl
Серверні сценарії	Обмежена функціональність із використанням "правил"	Так	Так

Продовження таблиці 1.4

Методи розділення		горизонтальне розділення, шардування за допомогою кластера MySQL або MySQL Fabric	розділення за діапазоном, списком та (починаючи з PostgreSQL 11) за хешем
Методи реплікації		Реплікація з декількох джерел Реплікація джерела-репліки	Реплікація джерела-репліки
Зовнішні ключі	ні	так	так
Концепції транзакцій	так	ACID	ACID
Паралельність	так	так	так
Авторизація	Так, на основі правил аутентифікації та правил БД	Користувачі з чіткою концепцією авторизації	Дрібні права доступу відповідно до стандарту SQL

Таким чином для невеликих мобільних додатків добре підходить Firebase але не підходить для великої вибірки даних

2 РОЗРОБКА СТРУКТУРИ БАЗИ ДАНИХ ТА ВИБІР ТЕХНОЛОГІЙ ПРОГРАМУВАННЯ

2.1 Аналіз завдання

Завданням дипломного проекту є створення додатку для керування персоналом. Основою таких систем є БД, в якій знаходитися інформація про персонал та відносно цієї інформації вибудовується вся структура додатка, даючи можливість розширювати функціонал в залежності від потреб. Прикладом є розмір компаній, які застосовують такі системи, в залежності від навантажень можуть бути різні способи реалізації з мінімальними витратами. Отже початковим етапом проекту це створити базу даних, яка за характером організації буде структурованою – реляційною базою даних. Оскільки у програмі буде закладене можливе розширення границь тому структурування необхідно для підтримки цієї можливості.

Таким чином основні вимоги до програми наступні:

- мати БД відповідальну за зберігання інформації персоналу;
- редагування даних: змінення посади, заробітної плати або інших даних;
- видалення даних з БД;
- збір інформації для аналізу.

Обіг інформації здійснюється за запитом клієнта. Звертаючись до списку він отримує дані з БД яка відповідає на запити наданням інформації відповідної до блоку запита

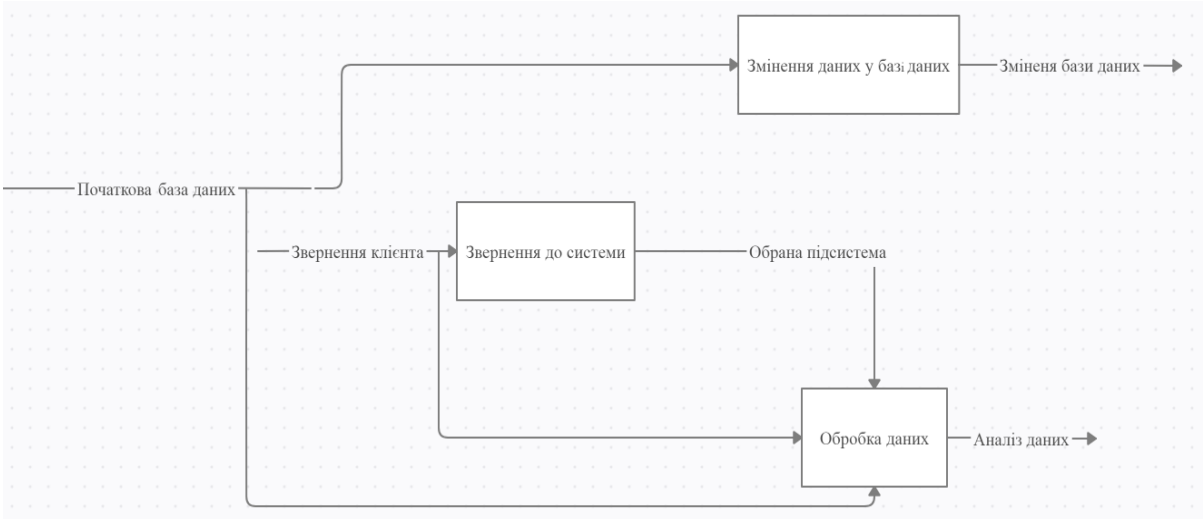

Рисунок 2.1 – Діаграма IDEF0

Рисунок 2.2 – Діаграма потоків даних

2.2 Архітектура проекту

Для реалізацій архітектури проекту були обрані бібліотеки архітектури андроїд додатків Room, ViewModel, LiveData та LifeCycle.

Room – це абстрактний шар який знаходится над SQLite, надаючи вільний доступ до БД. Room використовують для зберігання локальних данних. Наприклад, на час відсутності інтернету, данні будуть зберігатись локально до тих пір, доку не зявится з’єднання з сервером (див. рис. 2.3).

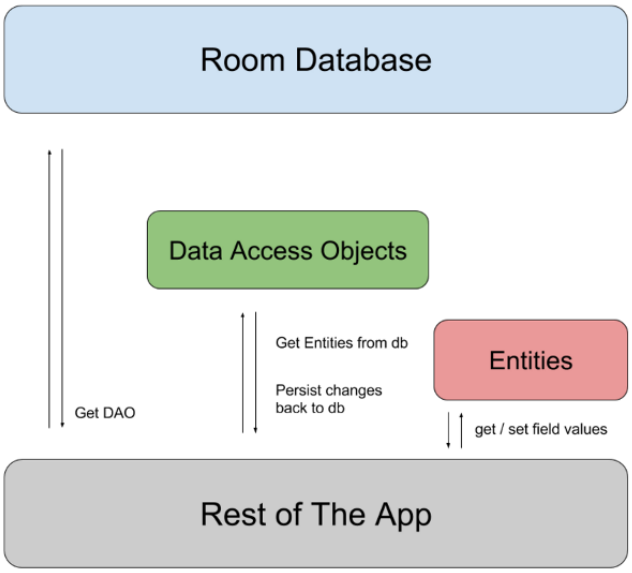

Рисунок 2.3 – Схема архетектури Room.

Дані сущності будуть зберігатись у SQLite, шар DAO відповідае за методи які будуть використовуватись для маніпуляцій з базою даних, наприклад, методи для додання та видалення якоїсь сущності за допомогою запитів Querty. Майже усі данні будуть обернені у LiveData таким чином дані activity будуть відображені у UI компоненты при зміні даних у базі даних.

SQLite – полегшена реляційна система керування базами даних. Особливістю SQLite є те, що вона не використовує парадигму клієнт-сервер, тобто рушій SQLite не є окремим процесом, з яким взаємодіє застосунок, а надає бібліотеку, з якою програма компілюється і рушій стає складовою частиною програми. Таким чином, як протокол обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому комп'ютері, на якому виконується застосунок. Простота реалізації досягається за рахунок того, що перед початком виконання транзакції весь файл, що зберігає базу даних, блокується; ACID-функції досягаються зокрема за рахунок створення файлу-журналу [7].

Кілька процесів або потоків можуть одночасно без жодних проблем читати дані з однієї бази. Запис в базу можна здійснити тільки в тому випадку, коли жодних інших запитів у цей час не обслуговується; інакше спроба запису закінчується невдачею, і в програму повертається код помилки. Іншим варіантом розвитку подій є автоматичне повторення спроб запису протягом заданого інтервалу часу [7].

У комплекті постачання йде також функціональна клієнтська частина у вигляді виконуваного файлу sqlite3, за допомогою якого демонструється реалізація функцій основної бібліотеки. Клієнтська частина працює з командного рядка, і дозволяє звертатися до файлу БД на основі типових функцій ОС [7].

Завдяки архітектурі рушія можливо використовувати SQlite як на вбудовуваних (embedded) системах, так і на виділених машинах з гігабайтними масивами даних [7].

LiveData – цє сховище даних, яке працює за принципом патерну Observer однак на відміну від Observer LiveData має уявленя про життевий цикл додатку, тобто він є відображенням житевого цикла інших компонентів додатку таких як активносі, фрагменти або сервіси. Табим чином LiveData оновлює лише ті компоненти обсервера які виконують свій житевий цкл.

Репозіторій – це клас Java, який відокремлює дані від додатку і є посередником між різними сервісами або джерелом даних наприклад веб-сервісів та інших. Оскільки Room не дозволяє робити запити до БД у головному потоці, ми використовуємо AsyncTasks для асинхроного виконання.

ViewModel використовується для з’єднання UI контролера з репозіторієм він зберігає та оброблює інформацію, яка надходить з БД у активність. ViewModel клас, який дозволяє зберігати данні при зміні конфігурації (див. рис. 2.4, 2.5).

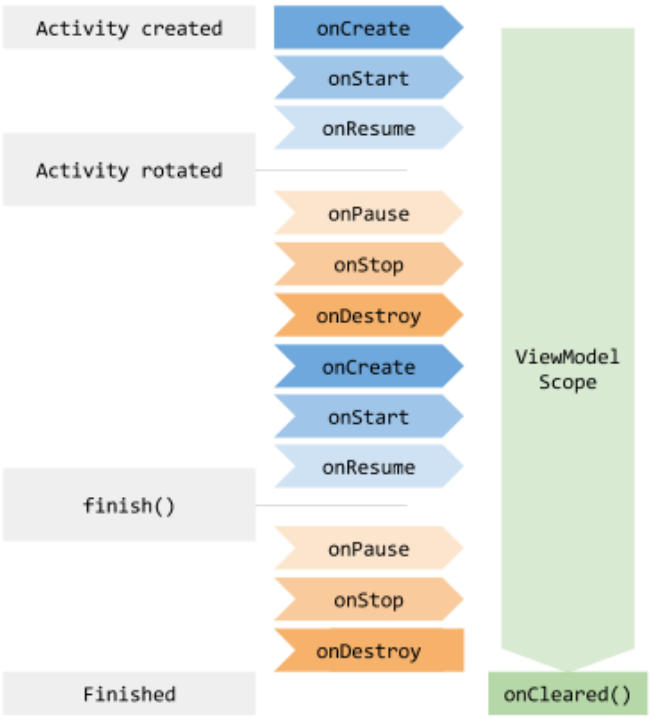

Рисунок 2.4 – Життєвий цикл ViewModel

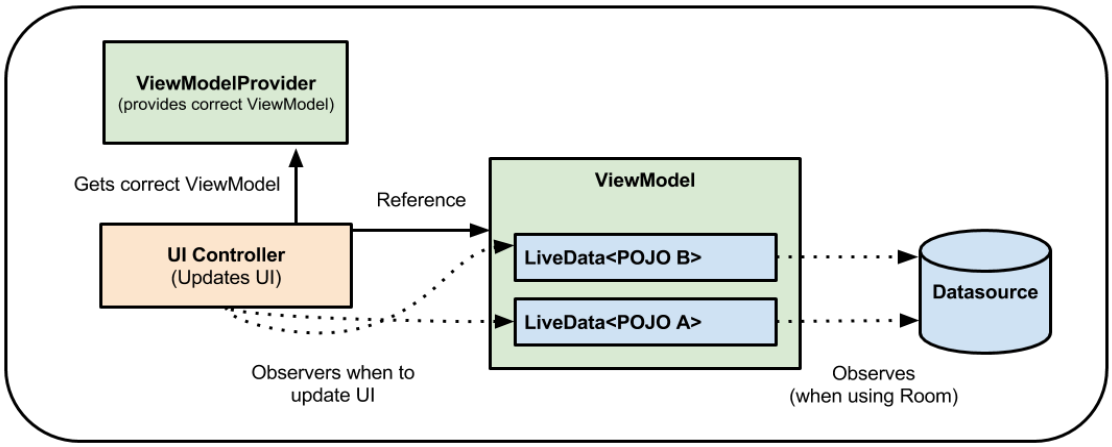

Рисунок 2.5 – Схема взаємодії ViewModel з базою даних

Архітектура проекту буде реалізована лише для локальної БД (див. рис. 2.6), але завдяки такій архітектурі є можливість добудувати проект та додати web сервіс (див. рис. 2.7). Таким чином, данні будуть зберігатись не

лише на сервері а також у локальної пам’яті у випадку відключення від мережі.

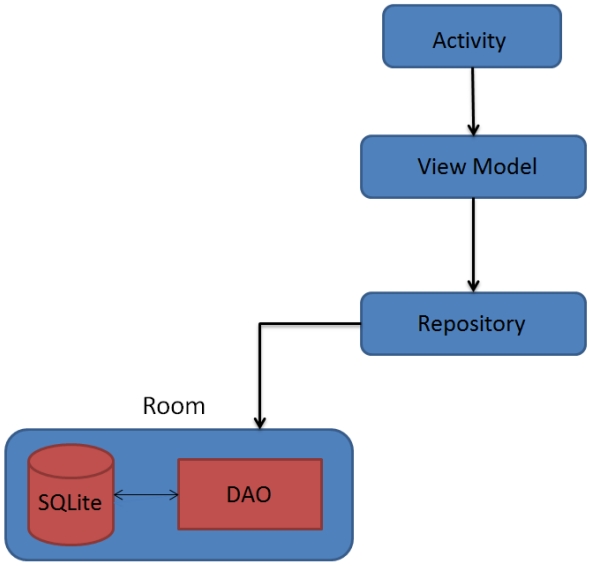

Рисунок 2.6 – Схема архітектури проекту

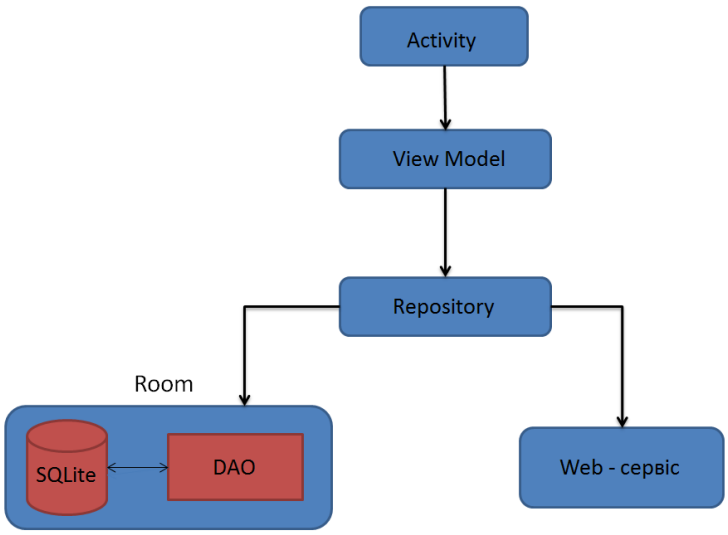

Рисунок 2.7 – Схема можливої архітектури .

2.3 Структура проекту

Оскільки це база даних реалізовує управління персоналом та розрахована на відключення від мережі.

Структура проекту (див. рис. 2.8) починається з головного меню яке об'єднує функціонал додатку. Основним класом є активність employActivity, яка реалізовує RecyclerView, за допомогою ViewModel ми можемо більш ефективно використовувати RecyclerView, адаптер якого базується на роботі з LiveData, що дозволяє відображати інформацію без необхідності перезавантажувати додаток, позбавляючи нас проблем з достовірністю даних.

Для реалізації запитів для видалення персоналу з БД ми реалізували свайп у ліву сторону, що автоматично видаляє персонал з БД та може разом з видаленням створювати документ, заповнений актуальними даними про звільнення когось з персоналу. Фунціонал додатку складеться з списку робітників, який можливо редагувати міняючи заробітну плату або посаду. Тобто, можливо буде реалізувати нарахування зарплатні за окладом та надбавкою за посаду. Реалізація зміни описана у активності addEmploy. Клас graphicActivity реалізує аналітичну роботу використовуючи графіки для більшої наочності статистичних даних, що надходять з БД та обраховуються у класі calculation. Таким чином, реалізовується узагальнене уявлення роботи внутрішніх класів.

Активність graphicsActivity відповідає за аналітичну частину проекту, відображаючи статистичні данні, які надаються у вигляді графіків. Для цього використовується бібліотека GraphView, яка дозволяє будувати графіки різного типу.

UML діаграму структури класів бази управління персоналом надано (див. рис. 2.13). Отже початкова активність mainActivity (Додаток А, MainActivity.java) є головним екраном через якиї здійснюється навігація у додатку.

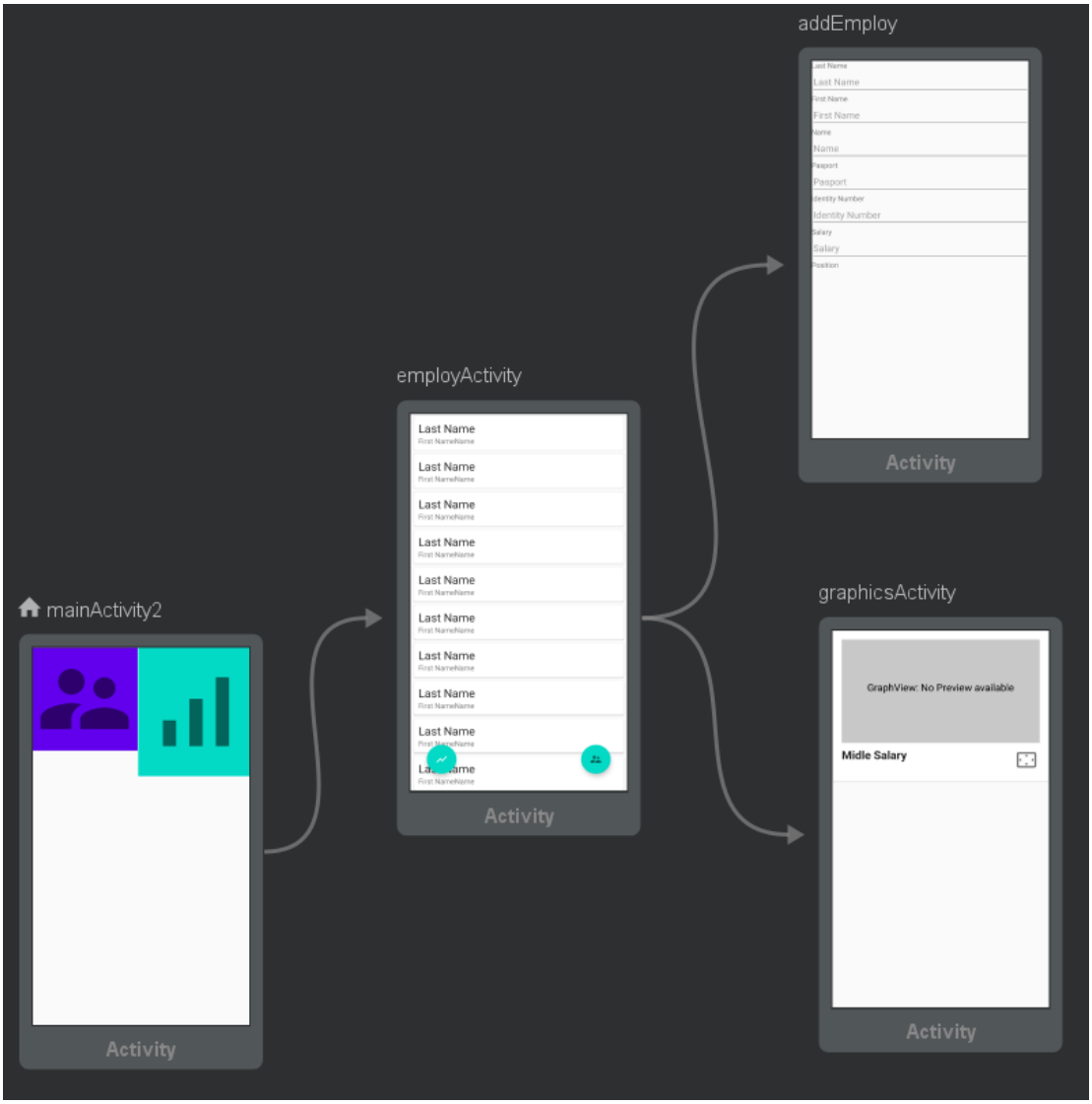

Рисунок 2.8 – Структура проекту

У головному меню здійснюється перехід до активності у employActivity (Додаток А, EmployActivity.java) якиї складається з RecyclerView дозволяє переглянути усіх робітників прокручуючи увесь персонал, також RecyclerView дає змогу маніпуляції жестами яка була реалізована для видалення робітника з БД роблячи свайп. employActivity має дві плаваючі кнопки. Кнопка з права дозволяє додати нового робітника, кнопка з права дозволяє переглянути статистику за усіма робітниками. Також данні можливо редагувати доторкнувшись до співробітника.

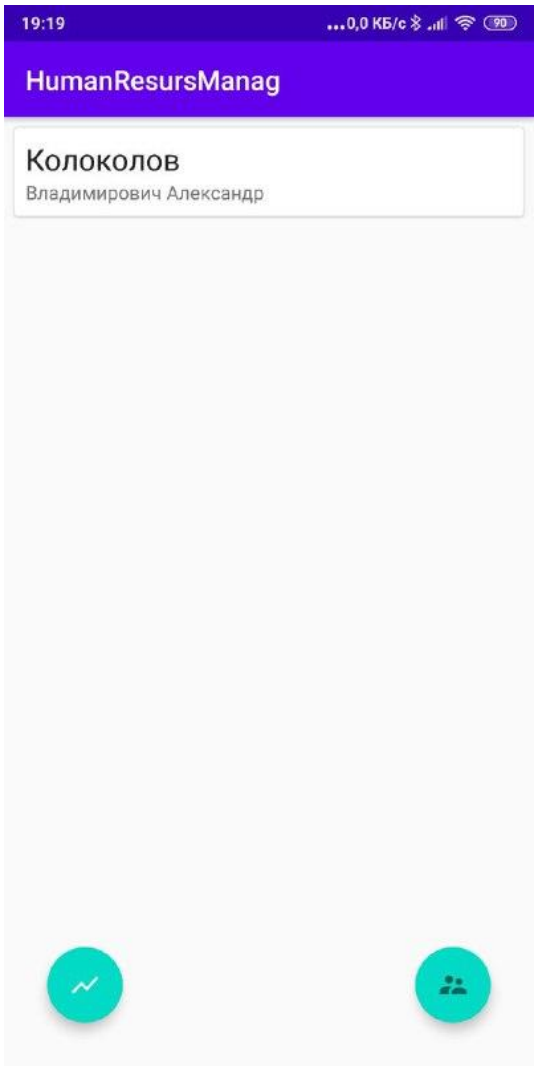

Рисунок 2.9 – employActivity

Заходячи до активності створення активності addEmploy(Додаток А , AddEmploy.java). Заповнюючи поля та натиснувши кнопку зберігти у правому верхнему куту екрану додаток зберігає дані у БД переходячи до employActivity

Для редагування потрібно натиснути на персонал. При натисненні відкриється також addEmploy але вже з заданими параметрами або повірка на то чи є ці дані новими чи вони створені проводиться за наявністю ID. За наявністю ID виконуються інші методи.

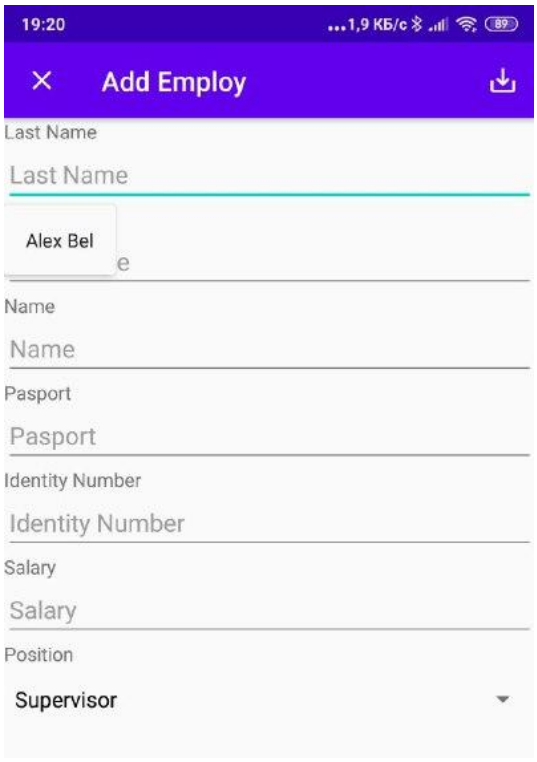

Рисунок 2.10 – addEmploy

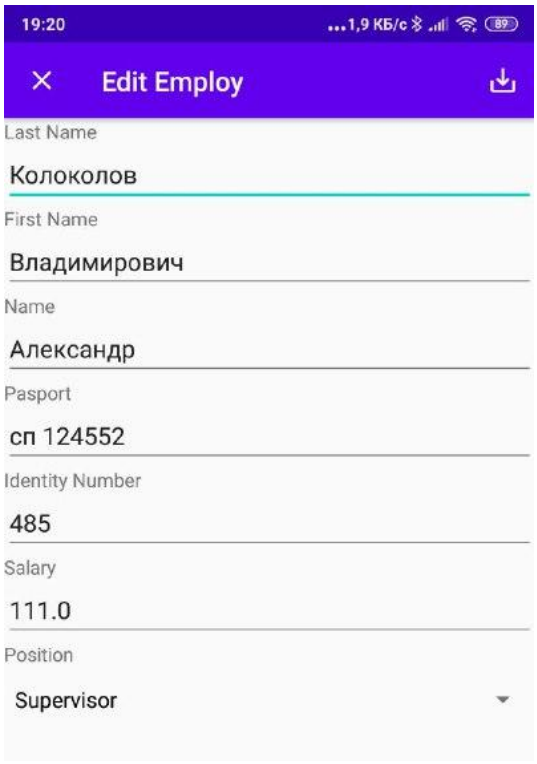

Рисунок 2.11 – addEmploy для редагування

graphicsActivity виконує розрахунок середнее заробітної плати

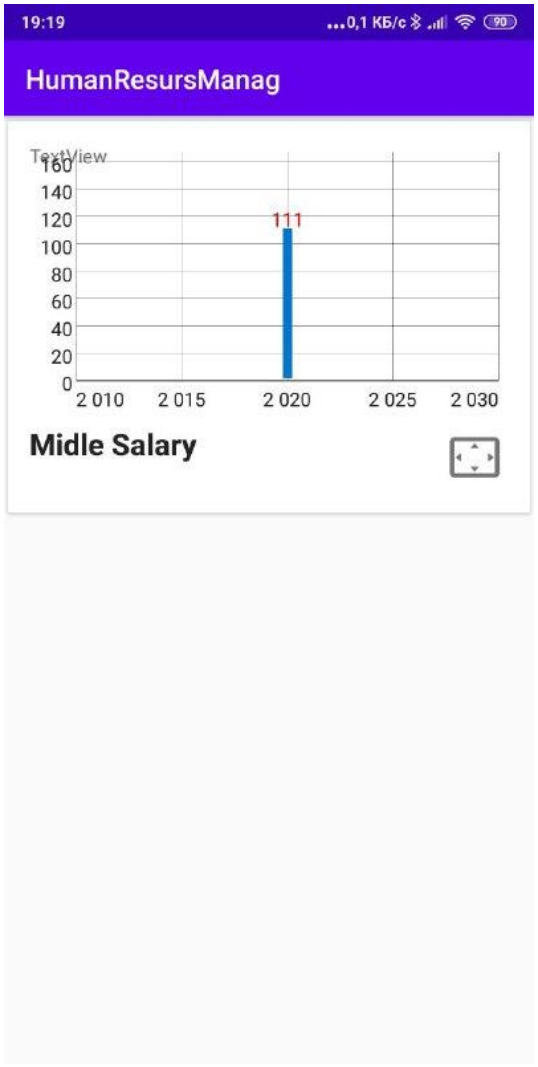

Рисунок 2.12 – graphicsActivity

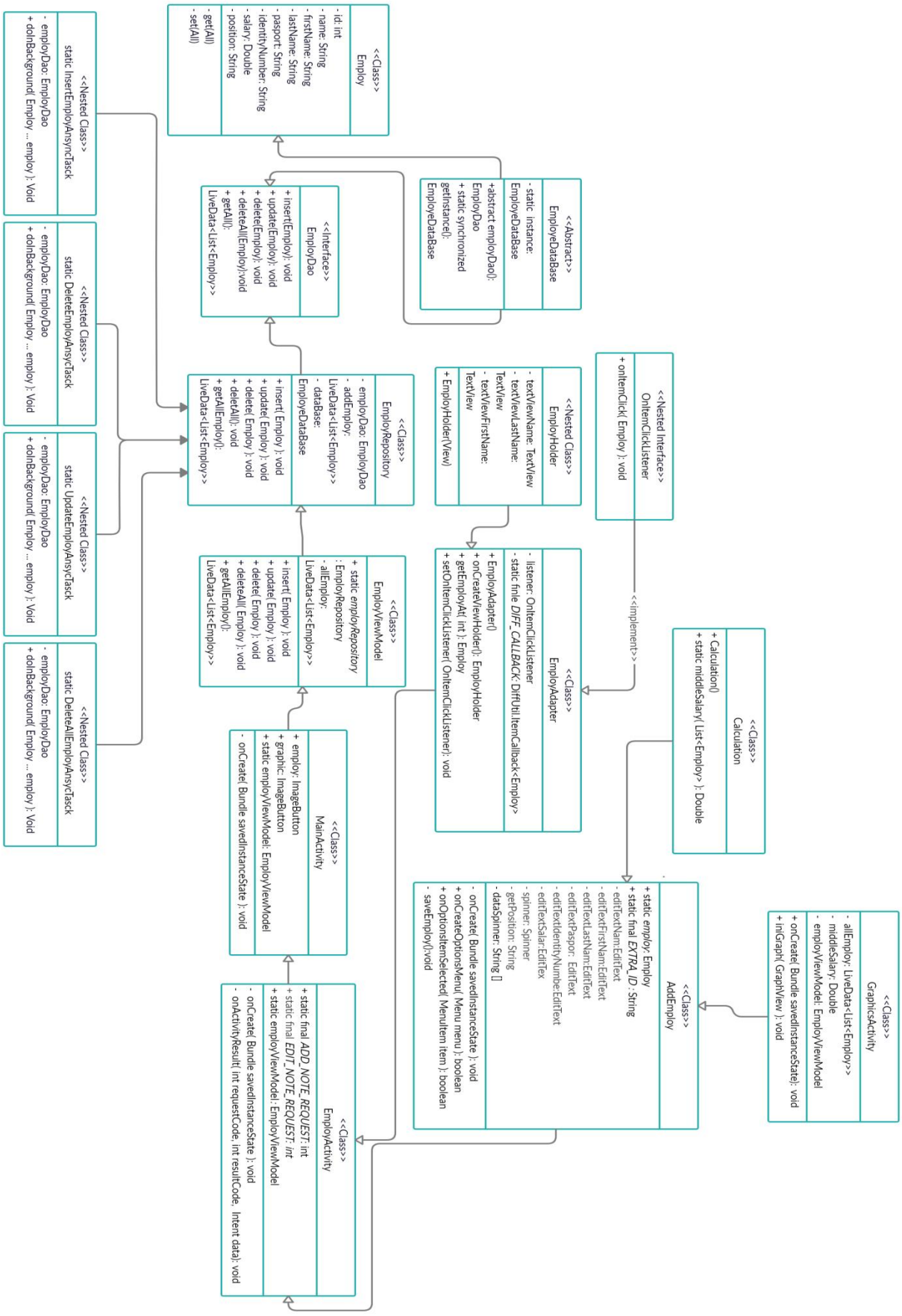

Рисунок 2.13 – UML схема структури класів

ВИСНОВКИ

Дана кваліфікаїна робота була реалізована з основними ознаками системи управління персоналу, яка дозволяє створити БД з персоналу та дозволяючи керувати цими даними змінюючи їх або видаляючи з БД за допомогою інструментів наданих SQLite. Таким чином цю систему можливо розширити додатковими модулями не руйнуючи архітектуру додатку, дозволяючи реалізувати виникаючи потреби у надані аналітичних даних: за персоналом, фінансів та інше. Додатково цю систему можливо доповнити за допомогою Firebase що дозволить зберігати інформацію у хмарному сервісі та зменшити ресурси обладненя які були би потрібні для обробки даних лише на SQLite, також зменшуючи вірогідність утрати даних у випадку якщо обладнання було загублене або зруйноване.

ПЕРЕЛІК ПОСИЛАНЬ

1. Белов В.С. Информационно-аналитические системы. Основы проектирования и применения: учебное пособие. М.: Евразийский открытый институт, 2010, 111 с.
2. Захлебин И.В., Фомичев В.А. Разработка метода семантического поиска в корпоративной базе данных. Томск: Наука плюс, 2014, 330 с.
3. Диго С.М. Проектирование и использование баз данных: учебник для студентов вузов. М.: Финансы и статистика, 2014, 364 с.
4. Дейт К. Введение в баз данных. М: Вильямс, 2001, 485с.
5. Дубенецкий Б.Я. Проектирование информационных систем. Л: ЛЭТИ, 2008, 675 с.
6. Developer.Android:[сайт]. URL: https://developer.android.com/guide.
7. Wikipedia:[сайт] URL: https://ru.wikipedia.org/wiki/SQLite
8. 1С: [сайт] URL: https://v8.1c.ru/hrm/
9. E-Staff: [сайт] URL: https://get-estaff.com.ua/

ДОДАТОК А

Код программи

Employ.java

```
package com.example.humanresursmanag.model;

import androidx.annotation.NonNull;
import androidx.room.ColumnInfo;
import androidx.room.Entity;
import androidx.room.PrimaryKey;
import java.io.Serializable;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@NoArgsConstructor
@Entity(tableName = "employ_table")
public class Employ implements Serializable {
    @NonNull
    @PrimaryKey(autoGenerate = true)
    private int id;
    private String name;
    private String firsName;
    private String lastName;
    private String pasport;
    private String identityNumber;
    @ColumnInfo(name = "salaryCol")
    private Double salary;
    private String position;
    public Employ(String name, String firsName, String lastName, String pasport, String
identityNumber
    ,Double salary, String position) {
        this.name = name;
        this.firsName = firsName;
        this.lastName = lastName;
        this.pasport = pasport;
        this.identityNumber = identityNumber;
        this.salary = salary;
        this.position = position;
    }

    public String getPosition() { return position; }
    public void setPosition(String position) { this.position = position; }
    public Double getSalary() {return salary;}
    public void setSalary(Double salary) { this.salary = salary;}
```

```
    public String getIdentityNumber() {return identityNumber;}
    public void setIdentityNumber(String identityNumber) { this.identityNumber =
identityNumber; }
    public int getId() { return id; }
    public void setId(int id) { this.id = id;}
    public String getName() { return name;}
    public void setName(String name) { this.name = name;}
    public String getFirsName() { return firsName;}
    public void setFirsName(String firsName) { this.firsName = firsName;}
    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }
    public String getPasport() { return pasport; }
    public void setPasport(String pasport) { this.pasport = pasport; }
}
```

EmployDao.java

```
package com.example.humanresursmanag.dao;

import androidx.lifecycle.LiveData;
import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;
import com.example.humanresursmanag.model.Employ;
import java.util.List;

@Dao
public interface EmployDao {
    @Insert
    void insert(Employ employ);
    @Update
    void update(Employ employ);
    @Delete
    void delete(Employ employ);
    @Query("DElETE FROM employ_table")
    void deleteAll();
    @Query("SELECT * FROM employ_table ORDER BY lastName DESC")
    LiveData<List<Employ>>getAll();
}
```

EmployDataBase.java

```
package com.example.humanresursmanag.database;

import android.content.Context;
import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;
import com.example.humanresursmanag.dao.EmployDao;
import com.example.humanresursmanag.model.Employ;

@Database(entities = {Employ.class}, version = 1)
public abstract class EmployeDataBase extends RoomDatabase {
    private static EmployeDataBase instance;
    public abstract EmployDao employDao();
    public static synchronized EmployeDataBase getInstance(Context context) {
        if (instance == null) {
            instance = Room.databaseBuilder(context.getApplicationContext(),
                    EmployeDataBase.class, "employ_table")
                    .fallbackToDestructiveMigration()
                    .build();
        }
        return instance;
    }
}
```

MainActivity.java

```
package com.example.humanresursmanag;

import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.ViewModelProviders;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;
import com.example.humanresursmanag.repository.EmployRepository;
import com.example.humanresursmanag.viewmodel.EmployViewModel;

public class MainActivity extends AppCompatActivity {

    ImageButton employ;
    ImageButton graphic;
    static EmployViewModel employViewModel;
    EmployRepository employRepository;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        employViewModel = ViewModelProviders.of(this).get(EmployViewModel.class);
        employ = (ImageButton) findViewById(R.id.imageButton);
        graphic = (ImageButton) findViewById(R.id.imageButton2);
        employ.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, EmployActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

EmployActivity.java

```
package com.example.humanresursmanag;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.Observer;
import androidx.recyclerview.widget.ItemTouchHelper;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;
import com.example.humanresursmanag.model.Employ;
import com.example.humanresursmanag.recycle.EmployAdapter;
import com.example.humanresursmanag.viewmodel.EmployViewModel;
import com.google.android.material.floatingactionbutton.FloatingActionButton;
import java.util.List;
import static com.example.humanresursmanag.AddEmploy.EXTRA_ID;

public class EmployActivity extends AppCompatActivity {
    public static final int ADD_NOTE_REQUEST = 1;
    public static final int EDIT_NOTE_REQUEST = 2;
    static EmployViewModel employViewModel = MainActivity.employViewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_employ);
        FloatingActionButton buttonAddEmploy = findViewById(R.id.button_add_employ);
```

```
        FloatingActionButton buttonGraphic = findViewById(R.id.button_graphic);

        buttonAddEmploy.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(EmployActivity.this, AddEmploy.class);
                startActivityForResult(intent, ADD_NOTE_REQUEST);
            }
        });

        buttonGraphic.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(EmployActivity.this, GraphicsActivity.class);
                startActivity(intent);
            }
        });

        RecyclerView recyclerView = findViewById(R.id.recycle_view);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        recyclerView.setHasFixedSize(true);
        final EmployAdapter adapter = new EmployAdapter();
        recyclerView.setAdapter(adapter);

        employViewModel.getAllEmploy().observe(this, new Observer<List<Employ>>() {

            @Override
            public void onChanged(List<Employ> employs) {
                adapter.submitList(employs);
            }
        });
        new ItemTouchHelper(new ItemTouchHelper.SimpleCallback(0,
                ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT) {
            @Override
            public boolean onMove(@NonNull RecyclerView recyclerView, @NonNull
RecyclerView.ViewHolder viewHolder, @NonNull RecyclerView.ViewHolder target) {
                return false; }
            @Override
            public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction)
{
                Employ employ = adapter.getEmployAt(viewHolder.getAdapterPosition());
                employViewModel.delete(adapter.getEmployAt(viewHolder.getAdapterPosition()));
                Toast.makeText(EmployActivity.this, "Employ are deleted",
Toast.LENGTH_SHORT).show();
            }
        }).attachToRecyclerView(recyclerView);
        adapter.setOnItemClickListener(new EmployAdapter.OnItemClickListener() {
            @Override
            public void onItemClick(Employ employ) {
                Intent intent = new Intent(EmployActivity.this, AddEmploy.class);
                intent.putExtra("serializbleInput", employ);
                intent.putExtra(EXTRA_ID, employ.getId());
```

```
            startActivityForResult(intent, EDIT_NOTE_REQUEST);
        }
    });
}

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == ADD_NOTE_REQUEST && resultCode == RESULT_OK) {
        employViewModel.insert((Employ) data.getSerializableExtra("serializedEmploy"));
        Toast.makeText(this, "Employ save", Toast.LENGTH_SHORT).show();
    } else if (requestCode == EDIT_NOTE_REQUEST && resultCode == RESULT_OK) {
        int id = data.getIntExtra(EXTRA_ID, -1);
        if (id == -1) {
            Toast.makeText(this, "Employ can`t be updated", Toast.LENGTH_SHORT).show();
            return;
        }
        employViewModel.update((Employ) data.getSerializableExtra("serializedEmploy"));
        Toast.makeText(this, "Employ updated", Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(this, "Employ not save", Toast.LENGTH_SHORT).show();
    }
}
}
```

AddEmploy.java

```
package com.example.humanresursmanag;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.NumberPicker;
import android.widget.Spinner;
import android.widget.Toast;
import com.example.humanresursmanag.model.Employ;

public class AddEmploy extends AppCompatActivity {
    public static Employ employ = new Employ();
    public static final String EXTRA_ID =
            "com.example.humanresursmanag.EXTRA_ID";
```

```
    private EditText editTextName;
    private EditText editTextFirstName;
    private EditText editTextLastName;
    private EditText editTextPasport;
    private EditText editTextIdentityNumber;
    private EditText editTextSalary;
    private Spinner spinner;
    private String getPosition;

    String [] dataSpinner = {"Supervisor","Worker"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_add_employ);
        ArrayAdapter<String>adapter = new
ArrayAdapter<String>(this,R.layout.support_simple_spinner_dropdown_item,dataSpinner);
        adapter.setDropDownViewResource(R.layout.support_simple_spinner_dropdown_item);

        spinner = (Spinner)findViewById(R.id.action_spinner);
        spinner.setAdapter(adapter);
        spinner.setPrompt("Position");
        editTextName = findViewById(R.id.edit_text_name);
        editTextFirstName = findViewById(R.id.edit_text_first_name);
        editTextLastName = findViewById(R.id.edit_text_last_name);
        editTextPasport = findViewById(R.id.edit_text_pasport);
        editTextIdentityNumber = findViewById(R.id.edit_text_identityNumber);
        editTextSalary = findViewById(R.id.edit_text_salary);

        spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
                getPosition = parent.getItemAtPosition(position).toString();
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {

            }
        });

        getSupportActionBar().setHomeAsUpIndicator(R.drawable.ic_close);
        Intent intent = getIntent();
        if (intent.hasExtra(EXTRA_ID)) {
            setTitle("Edit Employ");
            employ = (Employ)intent.getSerializableExtra("serializbleInput");
            editTextName.setText(employ.getName());
            editTextFirstName.setText(employ.getFirsName());
            editTextLastName.setText(employ.getLastName());
            editTextPasport.setText(employ.getPasport());
            editTextIdentityNumber.setText(employ.getIdentityNumber());
            editTextSalary.setText(String.valueOf(employ.getSalary()));
```

```
            spinner.setSelection(adapter.getPosition(employ.getPosition()));
        } else {
            setTitle("Add Employ");
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater menuInflater = getMenuInflater();
        menuInflater.inflate(R.menu.add_employ_menu, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item) {
        switch (item.getItemId()) {
            case R.id.save_employ:
                saveEmploy();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
    private void saveEmploy() {
        employ.setName(editTextName.getText().toString());
        employ.setFirsName(editTextFirstName.getText().toString());
        employ.setLastName(editTextLastName.getText().toString());
        employ.setPasport(editTextPasport.getText().toString());
        employ.setIdentityNumber(editTextIdentityNumber.getText().toString());
        employ.setSalary(Double.valueOf(editTextSalary.getText().toString()));
        employ.setPosition(getPosition);

        if (employ.getName().trim().isEmpty() || employ.getFirsName().isEmpty() ||
employ.getLastName().isEmpty()) {
            Toast.makeText(this, "Please insert a name, first name, last name",
Toast.LENGTH_SHORT).show();
            return;
        }
        Intent data = new Intent();
        data.putExtra("serializedEmploy", employ);
        int id = getIntent().getIntExtra(EXTRA_ID, -1);
        if (id != -1) {
            data.putExtra(EXTRA_ID, id);
        }
        setResult(RESULT_OK, data);
        finish();

    }
}
```

GraphicsActivity.java

```
package com.example.humanresursmanag;

import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.LiveData;
import android.graphics.Color;
import android.icu.util.Calendar;
import android.os.Build;
import android.os.Bundle;
import com.example.humanresursmanag.calculation.Calculation;
import com.example.humanresursmanag.model.Employ;
import com.example.humanresursmanag.viewmodel.EmployViewModel;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.series.BarGraphSeries;
import com.jjoe64.graphview.series.DataPoint;
import java.util.List;


public class GraphicsActivity extends AppCompatActivity {

    private LiveData<List<Employ>> allEmploy;
    private Double middleSalary;
    EmployViewModel employViewModel = EmployActivity.employViewModel;
    @RequiresApi(api = Build.VERSION_CODES.N)
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_graphics);
        GraphView graph = (GraphView) findViewById(R.id.graph);
        allEmploy = employViewModel.getAllEmploy();
        middleSalary = Calculation.middleSalary(allEmploy.getValue());
        iniGraph(graph);
    }

    @RequiresApi(api = Build.VERSION_CODES.N)
    public void iniGraph(GraphView graph) {
        BarGraphSeries<DataPoint> barGraphSeries = new BarGraphSeries<>(new DataPoint[]{
                new DataPoint(Calendar.getInstance().get(Calendar.YEAR), middleSalary)
        });
        barGraphSeries.setSpacing(98);
        barGraphSeries.setAnimated(true);
        barGraphSeries.setDrawValuesOnTop(true);
        barGraphSeries.setValuesOnTopColor(Color.RED);
        graph.addSeries(barGraphSeries);
        graph.getViewport().setMinX(2010);
        graph.getViewport().setMaxX((Calendar.getInstance()
                .get(Calendar.YEAR))+10);
        graph.getViewport().setMinY(0);
        graph.getViewport().setMaxY(middleSalary+middleSalary/2);
```

```
        graph.getViewport().setXAxisBoundsManual(true);
        graph.getViewport().setYAxisBoundsManual(true);
    }
}
```

EmployViewModel.java

```
package com.example.humanresursmanag.viewmodel;

import android.app.Application;
import androidx.annotation.NonNull;
import androidx.lifecycle.AndroidViewModel;
import androidx.lifecycle.LiveData;
import com.example.humanresursmanag.model.Employ;
import com.example.humanresursmanag.repository.EmployRepository;
import java.util.List;

public class EmployViewModel extends AndroidViewModel {

    static EmployRepository employRepository;
    private LiveData<List<Employ>> allEmploy;

    public EmployViewModel(@NonNull Application application) {
        super(application);
        employRepository = new EmployRepository(application);
        allEmploy = employRepository.getAllEmploy();
    }

    public void insert(Employ employ) { employRepository.insert(employ);}

    public void update(Employ employ) { employRepository.update(employ);}

    public void delete(Employ employ) { employRepository.delete(employ);}

    public void deleteAll() { employRepository.deleteAll();}

    public LiveData<List<Employ>> getAllEmploy() { return allEmploy;}

}
```

EmployRepository.java

```
package com.example.humanresursmanag.repository;

import android.app.Application;
```

```
import android.os.AsyncTask;
import androidx.lifecycle.LiveData;
import com.example.humanresursmanag.dao.EmployDao;
import com.example.humanresursmanag.database.EmployeDataBase;
import com.example.humanresursmanag.model.Employ;
rt java.util.List;
import lombok.AllArgsConstructor;

public class EmployRepository extends Application {
    private EmployDao employDao;
    private LiveData<List<Employ>> allEmploy;
    private EmployeDataBase dataBase;

    public EmployRepository(Application application) {
        dataBase = EmployeDataBase.getInstance(application);
        employDao = dataBase.employDao();
        allEmploy = employDao.getAll();
    }
    public void insert(Employ employ) {
        new InsertEmployAnsyncTasck(employDao).execute(employ);
    }
    public void update(Employ employ) {
        new UpdateEmployAnsycTasck(employDao).execute(employ);
    }
    public void delete(Employ employ) {
        new DeleteEmployAnsycTasck(employDao).execute(employ);
    }
    public void deleteAll() {
        new DeleteAllEmployAnsycTasck(employDao).execute();
    }
    public LiveData<List<Employ>> getAllEmploy() {
        return allEmploy;
    }

    @AllArgsConstructor
    private static class InsertEmployAnsyncTasck extends AsyncTask<Employ, Void, Void> {
        private EmployDao employDao;

        @Override
        protected Void doInBackground(Employ... employs) {
            employDao.insert(employs[0]);
            return null;
        }
    }

    @AllArgsConstructor
    private static class UpdateEmployAnsycTasck extends AsyncTask<Employ, Void, Void> {
        private EmployDao employDao;

        @Override
        protected Void doInBackground(Employ... employs) {
            employDao.update(employs[0]);
```

```
            return null;
        }
    }
    @AllArgsConstructor
    private static class DeleteEmployAnsycTasck extends AsyncTask<Employ, Void, Void> {
        private EmployDao employDao;

        @Override
        protected Void doInBackground(Employ... employs) {
            employDao.delete(employs[0]);
            return null;
        }
    }

    @AllArgsConstructor
    private static class DeleteAllEmployAnsycTasck extends AsyncTask<Void, Void, Void> {
        private EmployDao noteDao;

        @Override
        protected Void doInBackground(Void... voids) {
            noteDao.deleteAll();
            return null;
        }
    }
}
```

EmployAdapter.java

```
package com.example.humanresursmanag.recycle;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.DiffUtil;
import androidx.recyclerview.widget.ListAdapter;
import androidx.recyclerview.widget.RecyclerView;
import com.example.humanresursmanag.R;
import com.example.humanresursmanag.model.Employ;

public class EmployAdapter extends ListAdapter<Employ, EmployAdapter.EmployHolder> {

    private OnItemClickListener listener;

    public EmployAdapter() {
        super(DIFF_CALLBACK);
    }
```

```
    private static final DiffUtil.ItemCallback<Employ> DIFF_CALLBACK = new
DiffUtil.ItemCallback<Employ>() {
        @Override
        public boolean areItemsTheSame(@NonNull Employ oldItem, @NonNull Employ
newItem) {

            return oldItem.getId() == newItem.getId();
        }

        @Override
        public boolean areContentsTheSame(@NonNull Employ oldItem, @NonNull Employ
newItem) {
            return oldItem.getName().equals(newItem.getName()) &&
                    oldItem.getFirsName().equals(newItem.getFirsName()) &&
                    oldItem.getLastName().equals(newItem.getLastName());
        }
    };

    @NonNull
    @Override
    public EmployHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View itemView = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_employ,
parent, false);
        return new EmployHolder(itemView);
    }

    @Override
    public void onBindViewHolder(@NonNull EmployHolder holder, int position) {
        Employ currentEmploy = getItem(position);
        holder.textViewName.setText(currentEmploy.getName());
        holder.textViewFirstName.setText(currentEmploy.getFirsName());
        holder.textViewLastName.setText(currentEmploy.getLastName());
    }
    public Employ getEmployAt(int position) {
        return getItem(position);
    }
    class EmployHolder extends RecyclerView.ViewHolder {
        private TextView textViewName;
        private TextView textViewLastName;
        private TextView textViewFirstName;

        public EmployHolder(@NonNull View itemView) {
            super(itemView);
            textViewName = itemView.findViewById(R.id.text_view_name);
            textViewFirstName = itemView.findViewById(R.id.text_view_firstName);
            textViewLastName = itemView.findViewById(R.id.text_view_lastName);

            itemView.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    int position = getAdapterPosition();
                    if (listener != null && position != RecyclerView.NO_POSITION) {
```

```
                listener.onItemClick(getItem(position));
            }
        }
    });
  }
}

public interface OnItemClickListener { void onItemClick(Employ employ);}

public void setOnItemClickListener(OnItemClickListener listener) {
  this.listener = listener;
}
}
```

Calculation.java

```
package com.example.humanresursmanag.calculation;

import com.example.humanresursmanag.model.Employ;
import java.util.ArrayList;
import java.util.List;

public class Calculation {
  private Calculation() {}
  public static Double middleSalary(List<Employ> list) {
    List<Double>middle = new ArrayList<Double>();
    for (int i = 0; i < list.size(); i++) {
      middle.add(list.get(i).getSalary());
    }
    Double sum = 0.0;
    for (int i = 0; i < middle.size(); i++) {
      sum += middle.get(i);
    }
    return sum/middle.size();
  }
}
```