МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПРИРОДНИЧО-ГУМАНІТАРНИЙ ФАХОВИЙ КОЛЕДЖ

ДВНЗ «Ужгородський національний університет»

Циклова комісія програмування та інформаційних технологій

***Реєстраційний №***

***Дата реєстрації***

**Росоха Максим Валентинович**

студент 3-го курсу

групи КН-33

денної форми навчання

залікова книжка №253096 МБД

**Android To-Do list з нагадуванням про невиконані завдання**

**Курсовий проект з дисципліни**

«Об’єктно-орієнтоване програмування»

|  |  |
| --- | --- |
| Рекомендовано до захисту: | Науковий керівник: |
| Підпис наукового керівника  Дата «  »   2022 р. | Викл. Матоляк І.В. |

Робота захищена «  »   2022 р.

з оцінкою «  /  /  »

Підписи членів комісії

Ужгород 2022

**ЗМІСТ**

ВСТУП 3

[1. ТЕОРЕТИЧНА ЧАСТИНА 4](#_Toc103984905)

[1.1 Загальні відомості про предметну область 5](#_Toc103984906)

[1.2 Загальні відомості про використані технології 5](#_Toc103984910)

1.2.1 Основні поняття баз даних 7

[1.2.2 Концептуальна схема](#_Toc103984913) 8

[1.2.3 Логічна та фізична схеми](#_Toc103984914) 8

[1.3 Опис предметної області та постановка задачі](#_Toc103984915) 9

[1.4 Огляд аналогів 1](#_Toc103984916)0

[1.5 Обґрунтування вибору методу розв’язку задачі 1](#_Toc103984917)3

[2. ПРАКТИЧНА ЧАСТИНА 1](#_Toc103984918)4

[2.1 Визначення сутностей та їх атрибутів](#_Toc103984919) 14

[2.2 Інформаційно-логічна модель 16](#_Toc103984920)

[2.3 Перевірка таблиць на відповідність нормальним формам](#_Toc103984921) 19

[2.4 Функціональні частини програми](#_Toc103984922) 19

[2.5 Фізична структура програми 3](#_Toc103984923)1

[2.6 Тестування проекту 3](#_Toc103984924)1

[2.7 Інтерфейс та керівництво користувача 3](#_Toc103984925)4

[ВИСНОВКИ](#_Toc103984926) 39

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 4](#_Toc103984927)0

[Додаток 1. Концептуальна схема](#_Toc103984928)

[Додаток 2. Фізична схема](#_Toc103984929)

Додаток 3. Схема класів

[Додаток 4. Лістинг програми](#_Toc103984932)

Життя змушує нас завжди залишатися в тонусі. Щоб нічого не забувати і все постійно встигати, ви змушені записувати справи в списки. Для цих цілей ви напевно використовуєте не тільки паперові носії, а й спеціальні додатки для смартфонів.

Метою роботи є застосування та покращення набутих навичок з дисциплін «Об’єктно-орієнтоване програмування» та створення програми, основне завдання якої — дати користувачам змогу створювати та виконувати завдання з нагадуванням про них.

Завдання — створення якісного мобільного додатка, з максимально інформативним, простим та зручним у використанні графічним інтерфейсом. Створити реляційну базу даних на основі SQL. Мова програмування Java. Мобільний додаток повинен відповідати таким критеріям:

* Мова програмування Java;
* ООП підхід;
* Робота з базою даних;
* Коректна робота додатка.

Актуальність цього програмного продукту полягає у тому, щоб користувач зміг створювати або виконувати завдання, не забуваючи про них. Головна задача даного програмного продукту — нагадувати користувачеві про невиконані завдання, щоб нічого не забувати і все завжди встигати.

Предметом дослідження стали користувачі, які створюють та виконують завдання, і саме завдання.

Об’єктом дослідження виступає реляційна база даних SQLite, середовище керування базами даних SQLiteStudio, мова програмування Java та середовище розробки Android Studio.

Зважаючи на тему роботи, вік цільової аудиторії є необмеженим, тому що в будь-якому віці дозволено створювати та виконувати завдання з нагадуванням про них.

# **ТЕОРЕТИЧНА ЧАСТИНА**

У даному розділі пояснюються терміни і визначення, які допомагають зрозуміти практичну частину звіту.

Проект характеризується наступними термінами, технологіями і інструментальними засобами:

* Керівник;
* Працівник;
* Завдання;
* To Do List;
* Android Studio;
* База даних;
* СКБД;
* SQL;
* SQLite;
* DDL і DML;
* Java;
* ООП;
* Принципи ООП;
* DAO;
* Модель «сутність-зв'язок» ;
* Концептуальна схема;
* Логічна схема;
* Фізична схема;
* Нормалізація бази даних;
* Реляційна модель даних;
* ER діаграма;
* Цілісність даних;
* Нормальна форма.

## **Загальні відомості про предметну область**

«To Do List» або Список TO DO — це простий список справ або список завдань. У додатку є дві ролі «керівник» і «працівник»:

* Керівник — це користувач, який може створювати, задавати дату закінчення та редагувати виконавців завдання;
* Працівник — це користувач, який може тільки виконувати завдання.

Завдання — це завдання, яке створив керівник. Завдання може виконувати як працівник, так і керівник. Завдання характеризується назвою, змістом, керівником який створив його, датою публікації, датою закінчення, датою виконання і коментарями до нього.

Коментар — це текст, який користувач може додати до завдання в цілях уточнення завдання. Коментар характеризується змістом, датою написання та автором.

Додаток потрібен для того, щоб не забувати про справи та не завантажувати пам'ять. Список справ дозволить вам ефективно організувати свій робочий час. До нього можна записати все: від списку покупок до важливих ділових зустрічей.

## **Загальні відомості про використані технології**

DDL або мова визначення даних фактично складається з команд SQL, які можна використовувати визначення схеми бази даних. Він просто має справу з описами схеми бази даних і використовується для створення та зміни структури об'єктів бази даних у базі даних. DDL є набором команд SQL, що використовуються для створення, зміни та видалення структур бази даних, але не даних.

DML (мова обробки даних) — команди SQL, які мають справу з маніпулюванням даними, присутніми в базі даних, відносяться до DML або мови маніпулювання даними, і це включає більшість операторів SQL. Це компонент оператора SQL, який управляє доступом до даних та бази даних [1].

Базові принципи ООП:

* Абстракція - відокремлення концепції від її екземпляра;
* Поліморфізм - реалізація завдань однієї і тієї ж ідеї різними способами;
* Наслідування — здатність об'єкта чи класу базуватися іншому об'єкті чи класі. Це головний механізм повторного використання коду. Спадкове ставлення класів чітко визначає їхню ієрархію;
* Інкапсуляція – розміщення одного об'єкта або класу всередині іншого для розмежування доступу до них.

Не повторяйся (Don’t repeat yourself — DRY) — уникання повторного написання коду, виносячи в абстракції завдання і дані, що часто використовуються. Кожна частина коду або інформації повинна знаходитися в однині в доступному місці. Це один з принципів коду, що читається.

Принцип єдиного обов'язку — для кожного класу має бути визначено єдине призначення. Всі ресурси, необхідні для його здійснення, повинні бути інкапсульовані в цей клас і підпорядковані тільки цьому завданню.

Принцип відкритості/закритості — програмні сутності мають бути відкриті розширення, але закриті змін [2].

DAO — об'єкт доступу до даних. Це клас, який дозволяє завантажувати, додавати, видаляти та змінювати дані. Цей об’єкт не обмежується зберіганням будь-якого конкретного типу даних. З його використанням можна оперувати багатьма типами даних. Це забезпечує високу гнучкість. Однак, завдяки підтримці коду хорошої якості, кожному об’єкту призначається свій DAO, наприклад, User/UserDao, Animal/AnimalDao.

Java — об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems».



### Основні поняття баз даних

База даних — це певний набір даних, які пов'язані між собою спільною ознакою або властивістю, та впорядковані, наприклад, за алфавітом.

Об'єднання великої кількості даних в єдину базу дає змогу для формування безлічі варіації групування інформації — особисті дані клієнта, історія замовлень, каталог товарів та будь-що інше.

Головною перевагою БД є швидкість внесення та використання потрібної інформації. Завдяки спеціальним алгоритмам, які використовуються для баз даних, можна легко знаходити необхідні дані всього за декілька секунд. Також в базі даних існує певний взаємозв'язок інформації: зміна в одному рядку може спричинити зміни в інших рядках — це допомагає працювати з інформацією простіше і швидше [3].

Реляційна модель даних — це модель даних, де текстова чи числова інформація, що зображається за допомогою таблиць. Кожна таблиця, яка називається відношенням, складається з рядків, які називаються кортежами, та стовпчиків, які називаються атрибутами. Реляційна модель визначає представлення даних (структура), захищеність від некоректних змін (цілісність) та операції, що можуть бути виконані з даними (операції з даними) [4].

Цілісність даних — підтримка та забезпечення точності та цілісності даних протягом всього життєвого циклу. Нормалізація схеми бази даних — покроковий процес розбиття одного відношення відповідно до алгоритму нормалізації на декілька відношень на базі функціональних залежностей.

Нормальна форма - це така форма чого-небудь, що не допускає подальших спрощень.

Модель «сутність-зв'язок» — модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків.

Типи зв’язків між сутностями за множинністю:

* Один — до одного 1 : 1. Одному екземпляру однієї сутності відповідає один екземпляр іншої.
* Один — до багатьох 1 : ∞ або 1 : М. Одному екземпляру однієї сутності може відповідати кілька екземплярів іншої сутності
* Багато — до одного ∞ : 1 або М : 1. Багатьом екземплярам однієї сутності відповідає один екземпляр іншої сутності.
* Багато — до багатьох ∞ : ∞ або М : М. Багатьом екземплярам однієї сутності може відповідати кілька екземплярів іншої сутності.

Діаграма прив'язки сутності ER - це блок-схеми, які ілюструють, як «сутності» (люди, об'єкти або концепції) ставляться один до одного в системі. ER-діаграма - це та модель, яка найчастіше використовуються для розробки або налагодження реляційних баз даних в областях, бізнес-інформаційних систем і досліджень. Вона використовує набір геометричних символів, таких як прямокутник, ромб, овал і лінії, для відображення взаємозв'язку об'єктів, відносин і їх атрибутів [5].

### Концептуальна схема

Концептуальна схема — це опис високого рівня інформаційних потреб, що лежать в основі дизайну бази даних. Зазвичай вона включає лише основні поняття та зв'язки між ними. Як правило, це початкова модель з недостатньою деталізацією для створення фактичної бази даних.

Нотація Пітера Чена. Сутності відображуються у вигляді прямокутнків, зв'язки у вигляді ромбів. Якщо сутність бере участь у відносинах, вони пов'язані лінією. Якщо відносини не є обов'язковими, то лінія пунктирна. Атрибути позначаються в вигляді овалів і пов'язані з однією сутністю або зв'язком. Овал похідних атрибутів зображується пунктирним контуром.

### Логічна та фізична схеми

Логічна схема — функційна схема, в якій використовуються формальні методи опису.

Фізична схема — це опис того, як дані представлені та зберігаються (файли, індекси та ін.) у другорядному сховищі, використовуючи конкретну систему керування базами даних (СКБД).

## **Опис предметної області та постановка задачі**

У додатку повинна бути можливість авторизуватися як «керівник» або «працівник». Керівник створює завдання і час за скільки воно має бути виконане, працівник виконує його. Працівник і керівник можуть бачити активні, виконані і створені завдання (якщо це керівник).

Завдання повинно складатися з назви, змісту, дати публікації, дати закінчення, дати виконання, перегляду виконавців. У завданні повинна бути можливість створення коментарів.

Додаток повинен повідомляти виконавця завдання про невиконане завдання.

Завдання — створення якісного додатка, з максимально інформативним, простим та зручним інтерфейсом. Створити реляційну базу даних на основі SQL. Мова програмування Java.

Порядок виконання завдання можна схарактеризувати наступними кроками:

* Для виконання завдання було обрано реляційну модель бази даних. Базуючись на тому, що додаток для операційної системи Android середовищем розробки було вибрано Android Studio. Реляційною СКБД стала SQLite оскільки вона не є серверною.
* Створити концептуальну схему.
* Створити фізичну схему.
* Створити DDL і DML.
* Написати класи для роботи з базою даних.
* Написати класи з бізнес-логікою.
* Написати інтерфейс.
* Зробити тестування.
* Написати звіт.

## Огляд аналогів

Першим аналогом став мобільний додаток «Список завдань» (див. рисунок 1.1–1.2) [6].

Переваги мобільного додатка:

* зручний та інтуїтивно зрозумілий інтерфейс додатка;
* багато різного типу налаштувань;
* можливість зациклити виконання завдань.

Недоліки додатка:

* присутня реклама;
* відсутній зміст завдання;
* завдання призначаються тільки для самого себе.

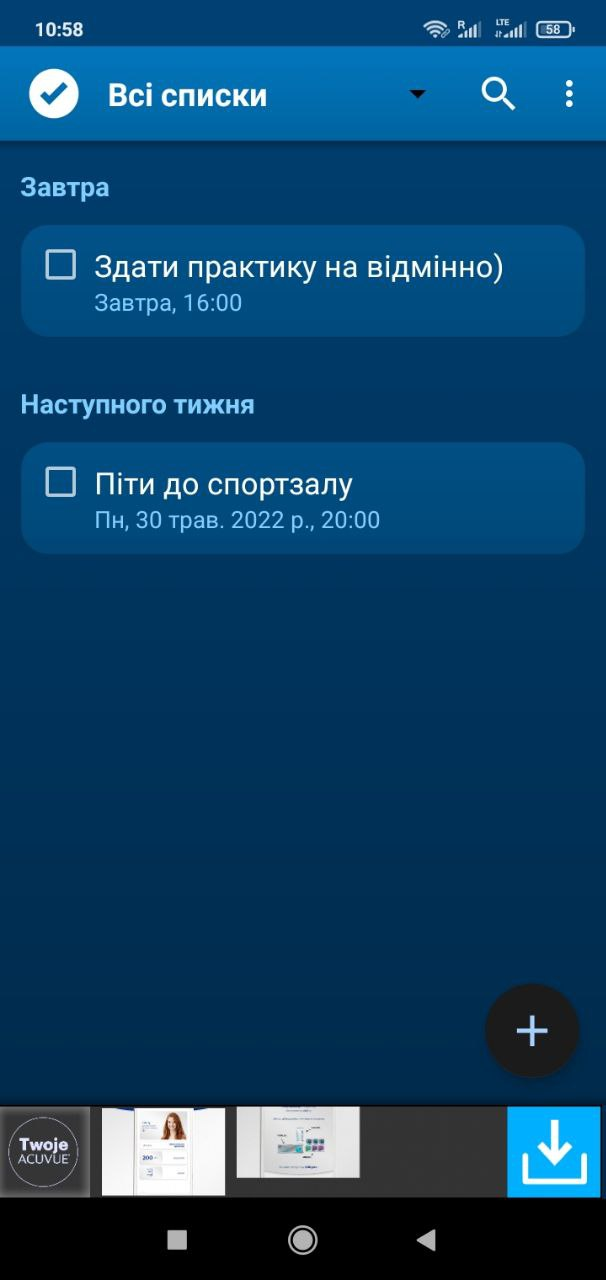


Рисунок 1.1 Додаток «Список завдань» перегляд завдань

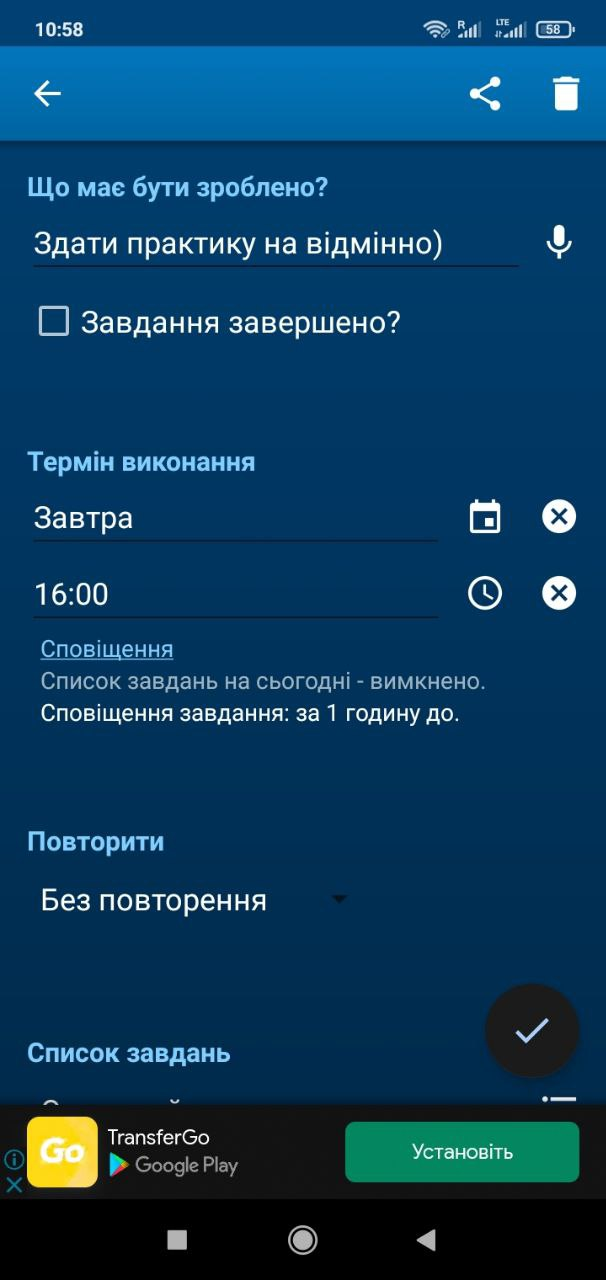


Рисунок 1.2 Додаток «Список завдань» перегляд вмісту завдання

Другим аналогом став мобільний додаток «Google Classroom» (див. рисунок 1.3–1.5) [7].

Переваги мобільного додатка:

* зручний та інтуїтивно зрозумілий інтерфейс додатка;
* багато різного типу налаштувань;
* можливість відправляти різні файли;
* розбиття на класи;
* можливість публікувати не лише завдання.

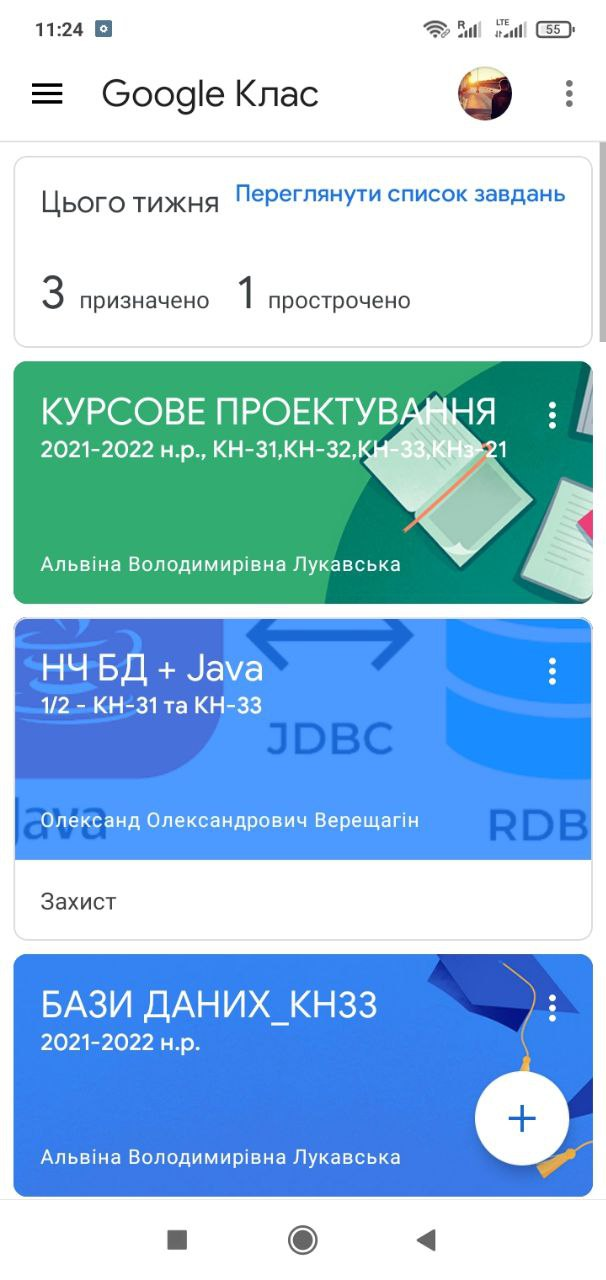


Рисунок 1.3 Додаток «Google Classroom» перегляд класів

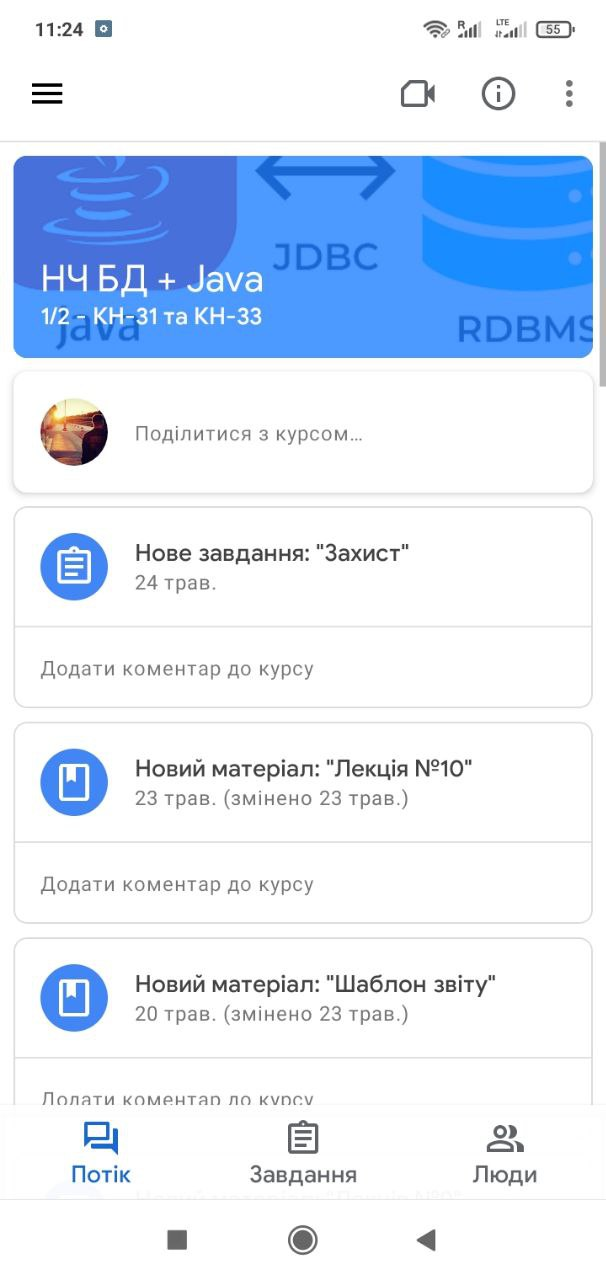


Рисунок 1.4 Додаток «Google Classroom» перегляд вмісту класу

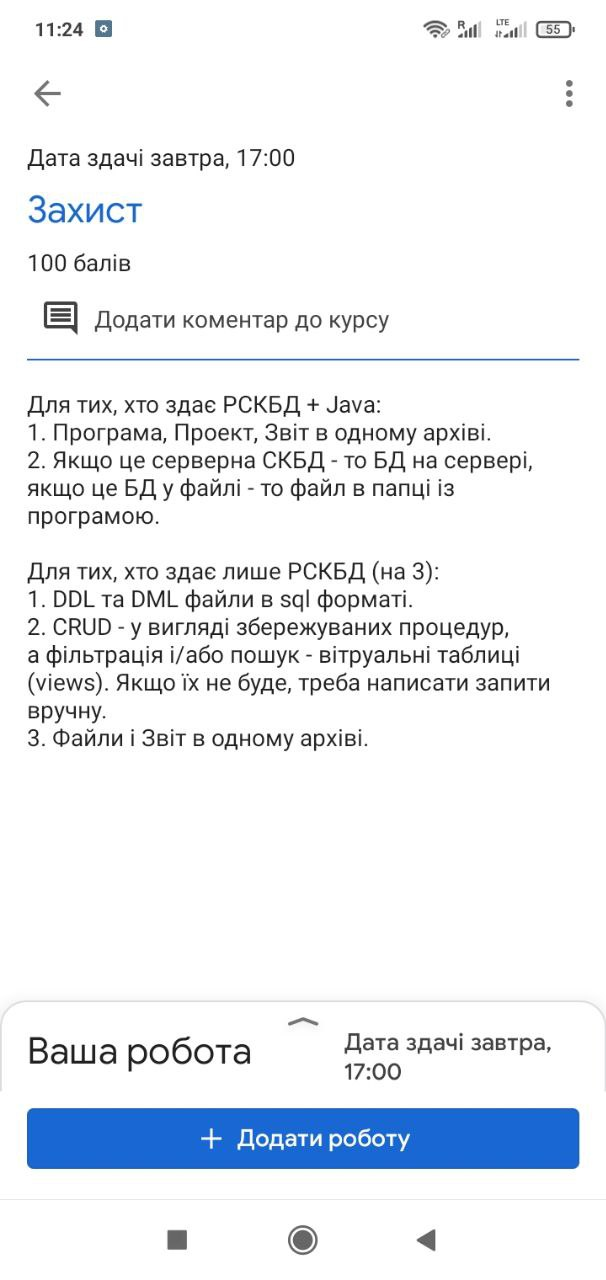


Рисунок 1.5 Додаток «Google Classroom» перегляд завдань

## Обґрунтування вибору методу розв’язку задачі

* Модель даних обрана «сутність-зв'язок» тому, що це модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків.
* Реляційна база даних обрана тому, що вона найбільш інтуїтивно зрозуміла і під час навчання ми практикували тільки її.
* СКБД SQLite обрана тому, бо для її використання не потрібне підключення до сервера.
* Алгоритмом хешування паролів був обраний SHA-1, тому що він надійний і досить легкий у використанні.
* Клієнт для управління СКБД був обраний SQLiteStudio оскільки у ньому простий і зрозумілий інтерфейс, а також є все, що потрібно для реалізації поставленого завдання.
* Середовище розробки було обрано Android Studio, оскільки воно як найкраще підходить для реалізації Android додатків.
* UI фреймворк був обраний Android, оскільки додаток розробляється під операційну систему Android.

# **ПРАКТИЧНА ЧАСТИНА**



## **Визначення сутностей та їх атрибутів**

У концептуальній схемі описані такі сутності: користувач, вибраний користувач, виконавець, завдання і коментар (див. таблиця 2.1).

*Таблиця 2.1 Сутності і їх атрибути*

|  |  |
| --- | --- |
| Користувач | Це людина, яка користується додатком. Користувач поділяється на керівника і працівника. |
| Ідентифікатор (ключове поле, обов’язковий кортеж), ПІБ (обов’язковий кортеж), логін (обов’язковий кортеж), роль (обов’язковий кортеж), пароль (обов’язковий кортеж). |
| Пов’язаний з сутністю виконавець типом 1:багатьох;  пов’язаний з сутністю вибраний користувач типом 1: багатьох;  пов’язаний з сутністю завдання типом 1: багатьох;  пов’язаний з сутністю коментар типом 1: багатьох. |
| Вибраний користувач | Це користувач, якого обрав інший користувач в цілях скорочення часу пошуку його вдруге. |
| Ідентифікатор поточного користувача (обов’язковий кортеж), ідентифікатор вибраного користувача (обов’язковий кортеж). |
| Пов’язаний з сутністю користувач типом багато:1. |
| Виконавець | Це користувач, для якого було створено завдання. |
| Ідентифікатор завдання (обов’язковий кортеж), ідентифікатор користувача (обов’язковий кортеж), повідомлення про створення завдання (обов’язковий кортеж), повідомлення з попередженням (обов’язковий кортеж), повідомлення про не виконання завдання (обов’язковий кортеж). |

*Продовження таблиця 2.1*

|  |  |
| --- | --- |
|  | Пов’язаний з сутністю користувач типом багато:1;  Пов’язаний з сутністю коментар типом 1:багато;  Пов’язаний з сутністю завдання типом багато:багато. |
| Завдання | Це завдання, яке створив керівник і виконує працівник. |
| Ідентифікатор(ключове поле, обов’язковий кортеж), ідентифікатор керівника(обов’язковий кортеж), назва(обов’язковий кортеж), зміст, дата публікації(обов’язковий кортеж), дата закінчення(обов’язковий кортеж), дата виконання, виконане? (обов’язковий кортеж). |
| Пов’язане з сутністю користувач типом багато:1;  Пов’язане з сутністю виконавець типом багато:багато;  Пов’язане з сутністю коментар типом 1:багато. |
| Коментар | Це коментар до завдання. Використовується у цілях уточнення деякої інформації по завданню. |
| Ідентифікатор(ключове поле, обов’язковий кортеж), ідентифікатор завдання(обов’язковий кортеж), ідентифікатор користувача(обов’язковий кортеж), зміст(обов’язковий кортеж), дата написання(обов’язковий кортеж). |
| Пов’язаний з сутністю користувач типом багато:1;  Пов’язаний з сутністю виконавець типом багато:1;  Пов’язаний з сутністю завдання типом багато:1. |

## **Інформаційно-логічна модель**

Таблиця «users» (див рисунок 2.1):

* батьківська таблиця;
* статичний тип;
* ключове поле «id»;
* унікальне поле «login»;
* CHECK поле «role».

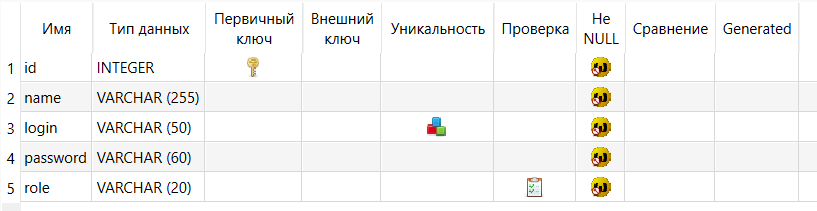


Рисунок 2.1 Структура таблиці «users»

Таблиця «executors» (див. рисунок 2.2):

* залежна таблиця;
* динамічний тип;
* CHECK поля «about\_creating\_task», «task\_completion\_warning», «late\_completion\_of\_task».

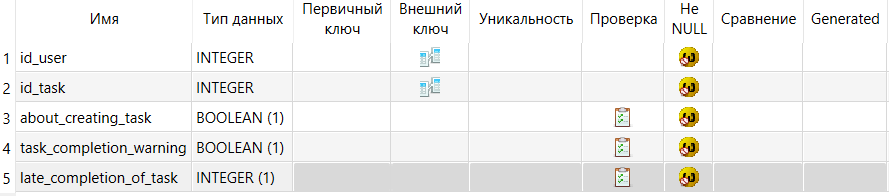


Рисунок 2.2 Структура таблиці «executors»

Таблиця «favorite\_users» (див. рисунок 2.3):

* залежна таблиця;
* динамічний тип;
* CHECK поле «id\_favorite\_user».

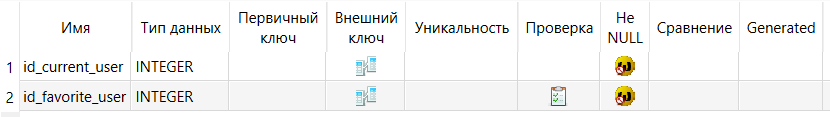


Рисунок 2.3 Структура таблиці «favorite\_users»

Таблиця «tasks» (див. рисунок 2.4):

* залежна таблиця;
* динамічний тип;
* ключове поле «id»;
* CHECK поле «done».



Рисунок 2.4 Структура таблиці «tasks»

Таблиця «comments» (див. рисунок 2.5):

* залежна таблиця;
* динамічний тип;
* ключове поле «id».

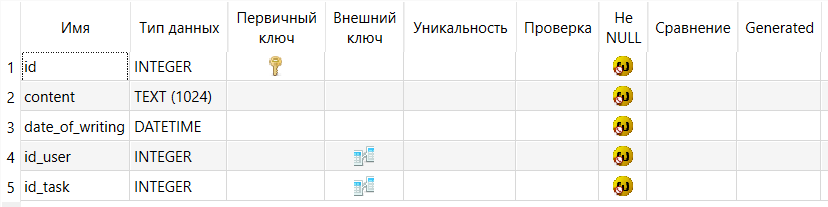


Рисунок 2.5 Структура таблиці «comments»

Приклад заповнених даних показаний у таблицях 2.2-2.6:

Таблиця 2.2 «users»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ідентифікатор | ПІБ | Логін | Пароль | Роль |
| 1 | Росоха Максим Валентинович | rosokha | 7C222FB2927D828AF22F592134E8932480637C0D | Leader |
| 2 | Павло Валерійович Дуров | durov | D36DA3E6884F6D1E9E7983FF13E99CF5C8F5745A | Worker |
| 3 | Ліонель Андрес Мессі Куччиттіні | messi | 7D8F4B4B4613DC7E15333E6449692AD4AF502D1D | Leader |
| …. | | | | |
| 20 | Маск Ілон Рів | mask | 614E00A6CF5E0A27838EC055FF89E945F681054F | Worker |

Таблиця 2.3 «executors»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ідентифікатор користувача | Ідентифікатор завдання | повідомлення про створення завдання | повідомлення з попередженням | повідомлення про не виконання завдання |
| 1 | 4 | 0 | 0 | 0 |
| 3 | 21 | 1 | 0 | 0 |
| … | | | | |
| 7 | 3 | 1 | 1 | 1 |

Таблиця 2.4 «favorite\_users»

|  |  |
| --- | --- |
| Ідентифікатор поточного користувача | Ідентифікатор вибраного користувача |
| 2 | 4 |
| 1 | 7 |
| … | |
| 9 | 4 |

Таблиця 2.5 «tasks»

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Ідентифікатор | Ідентифікатор виконавця | Назва | Зміст | Дата публікації | Дата закінчення | Дата виконання | Виконане? |
| 1 | 2 | Здати практику | Здай практику бо будеш без стипендії | 2022-05-13 11:26 | 2022-05-27 12:00 |  | 0 |
| … | | | | | | | |
| 23 | 7 | Піти у спотрзал |  | 2022-04-26 13:33 | 2022-06-10 22:00 | 2022-06-09 16:32 | 1 |

Таблиця 2.6 «comments»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ідентифікатор | Зміст | Дата написання | Ідентифікатор користувача | Ідентифікатор завдання |
| 1 | Я не знаю, що написати. Допоможіть | 2022-05-13 12:13 | 1 | 3 |
| … | | | | |
| 9 | У кого взяти постановку задачі? | 2022-04-03 18:00 | 4 | 2 |

Наповнення бази даних здійснює керівник.

У базі даних підтримуються такі функції як: (CRUD), пошук та фільтри.

Всі мої таблиці та кортежі пишуться в нижньому регістрі з заміною пробілів на нижнє підкреслення. Назви таблиць записані в множині, а кортежів в однині.

## **Перевірка таблиць на відповідність нормальним формам**

Таблиця «users» знаходиться в 3 нормальній формі, тому що неключові атрибути не залежать один від одного, але залежать від ключового атрибута, це відповідає третій нормальній формі.

Таблиця «executors» знаходиться у 2 нормальній формі, тому що у таблиці відсутні часткові залежності.

Таблиця «favorite\_users» знаходиться у 2 нормальній формі, тому що у таблиці відсутні часткові залежності.

Таблиця «tasks» знаходиться в 3 нормальній формі, тому що неключові атрибути не залежать один від одного, але залежать від ключового атрибуту, це відповідає третій нормальній формі.

Таблиця «comments» знаходиться в 3 нормальній формі, тому що неключові атрибути не залежать один від одного, але залежать від ключового атрибута, це відповідає третій нормальній формі.

Всі таблиці бази даних відповідають нормальним формам. Більшість відповідає третій нормальній формі, тому у мене не можуть бути аномалії в даних і структура даних не буде порушена.

## Функціональні частини програми

У файлі DDL зберігаються SQL запити для створення таблиць у базі даних. У файлі DML зберігаються SQL запити для заповнення таблиць. Ці файли я використав для створення і заповнення бази даних у цілях тестування і подальшого використання.

База даних зберігається у внутрішніх файлах додатка.

Повний цикл спілкування з базою даних:

* перевірка на наявність бази даних (якщо бази даних немає, тоді відбувається її створення);
* відкриття підключення;
* запит на вибірку;
* запит на створення;
* запит на оновлення;
* запит на видалення;
* закриття підключення.

В запитах використовував анонімні параметри. Великою перевагою їх використання є те, що за допомогою них не можна зробити SQL ін’єкцію.

Data Access Object — об'єкт що надає абстрактний інтерфейс до деяких видів баз даних чи механізмів персистентності реалізуючи певні операції без розкриття деталей бази даних. Він надає відображення від програмних викликів до рівня персистентності.

Використав такі принципи ООП як інкапсуляція, наслідування, абстракції, інтерфейси, принцип не повторяйся (Don’t repeat yourself — DRY), принцип єдиного обов'язку, принцип відкритості/закритості, KISS.

Структура проекту складається з чотирьох директорій (див. рисунок 2.6):

* авторизація;
* dao;
* завдання;
* користувачі.

У директорії авторизація зберігаються класи для авторизації користувача у додатку.

У директорії dao зберігаються класи для роботи з базою даних.

У директорії завдання зберігаються класи для роботи із завданнями.

У директорії користувачі зберігаються класи для роботи з користувачами.



Рисунок 2.6 Структура проекту

Клас «MainActivity» — це клас з якого починається запуск програми. Являє собою графічний користувацький інтерфейс з вибором авторизації (див. рисунок 2.7).

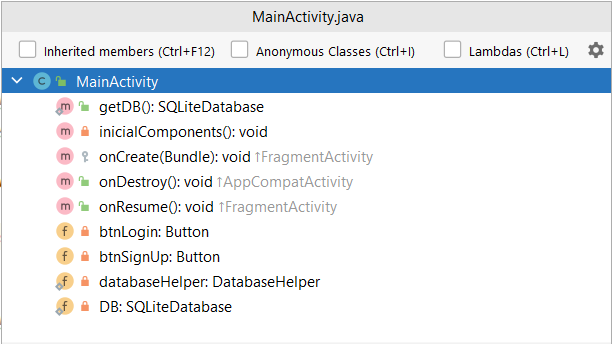


Рисунок 2.7 Внутрішня структура класу «MainActivity»

Клас «SignUpActivity» являє собою графічний користувацький інтерфейс реєстрації (див. рисунок 2.8).



Рисунок 2.8 Внутрішня структура класу «SignUpActivity»

Клас «LogInActivity» являє собою графічний користувацький інтерфейс входу в програму (див. рисунок 2.9).

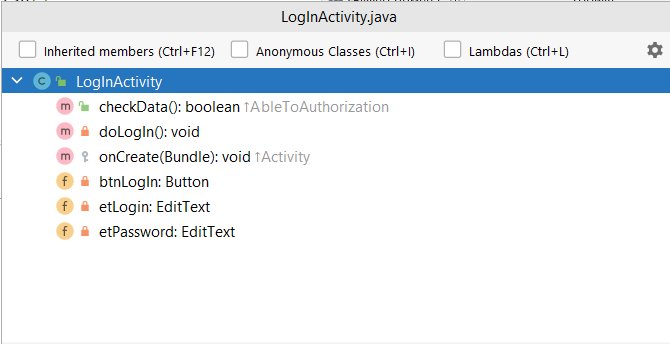


Рисунок 2.9 Внутрішня структура класу «LogInActivity»

Клас авторизації «AuthorizationDao», який працює з базою даних (див. рисунок 2.10).

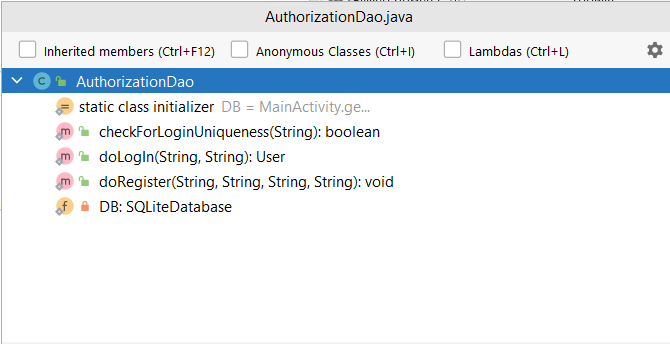


Рисунок 2.10 Внутрішня структура класу «AuthorizationDao»

«CommentDao» клас коментаря, який працює з базою даних (див. рисунок 2.11).

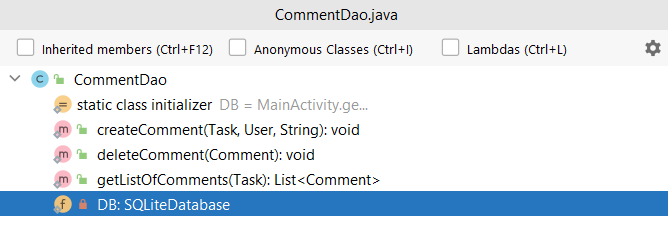


Рисунок 2.11 Внутрішня структура класу «CommentDao»

«DatabaseHelper» допоміжний клас для роботи з базою даних (див. рисунок 2.12).

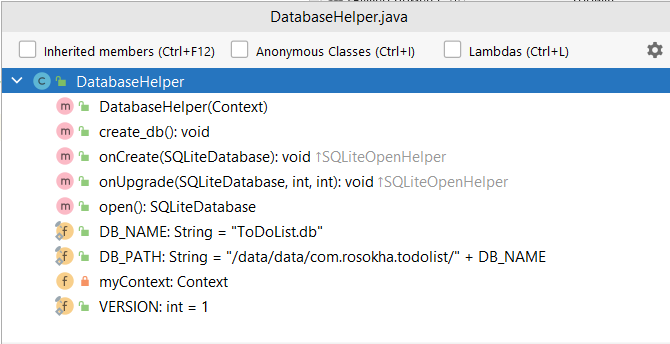


Рисунок 2.12 Внутрішня структура класу «DatabaseHelper»

«LoaderOfFavoriteUsersDao» клас, що працює з вибраними користувачами у базі даних (див. рисунок 2.13).

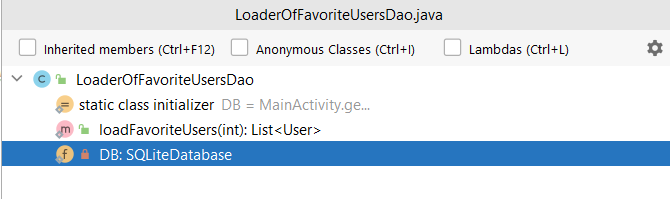


Рисунок 2.13 Внутрішня структура класу «LoaderOfFavoriteUsersDao»

«LoaderOfTasksDao» клас який завантажує завдання з бази даних (див. рисунок 2.1 4).

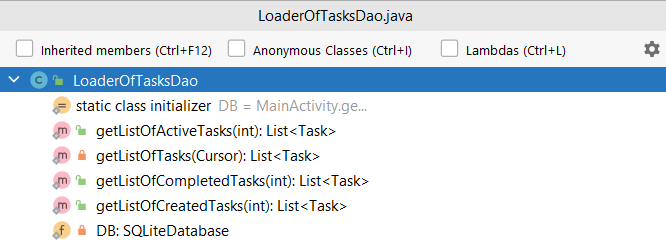


Рисунок 2.14 Внутрішня структура класу «LoaderOfTasksDao»

«SearchDao» клас пошуку (див. рисунок 2.15).

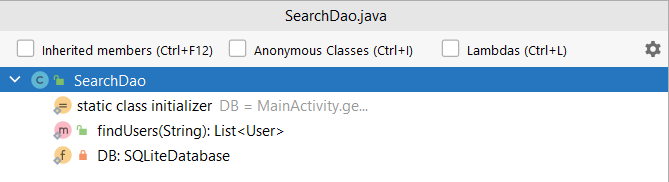


Рисунок 2.15 Внутрішня структура класу «SearchDao»

«TaskDao» являє собою сутність, яка створює завдання і додає виконавців до нього (див. рисунок 2.16).

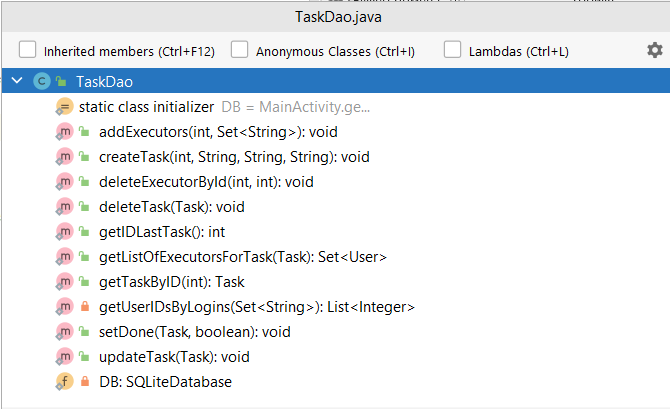


Рисунок 2.16 Внутрішня структура класу «TaskDao»

«UserDao» користувач, що взаємодє з базою даних (див. рисунок 2.17).

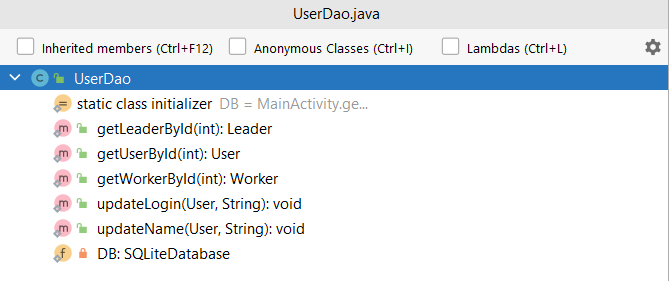


Рисунок 2.17 Внутрішня структура класу «UserDao»

«ViewProfileDao» клас перегляду профілю, який працює з базою даних (див. рисунок 2.18).

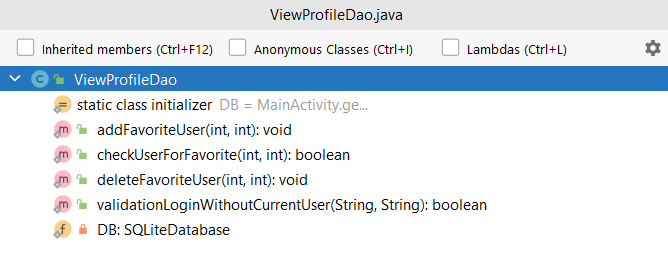


Рисунок 2.18 Внутрішня структура класу «ViewProfileDao»

«Comment» являє собою коментар до завдання (див. рисунок 2.19).

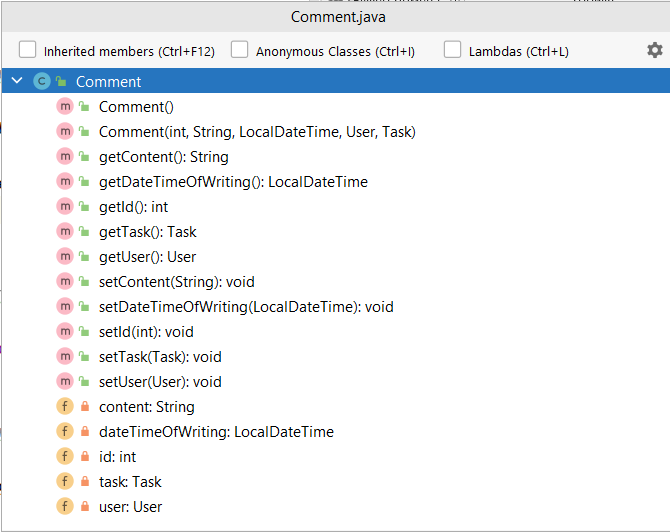


Рисунок 2.19 Внутрішня структура класу «Comment»

«FinderExecutorsActivity» являє собою графічний користувацький інтерфейс пошуку виконавців до завдання (див. рисунок 2.20).

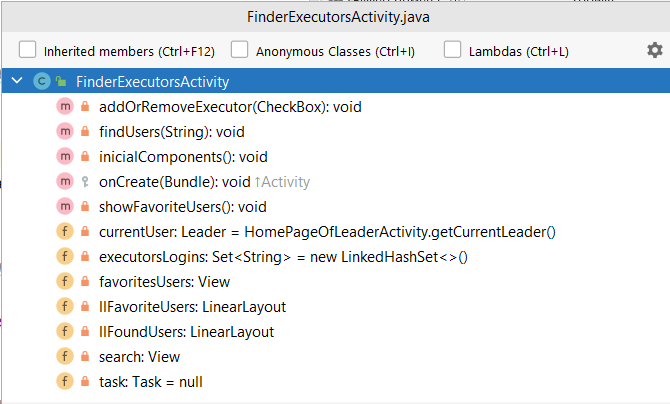


Рисунок 2.20 Внутрішня структура класу «FinderExecutorsActivity»

«Task» являє собою об’єкт завдання (див. рисунок 2.20).

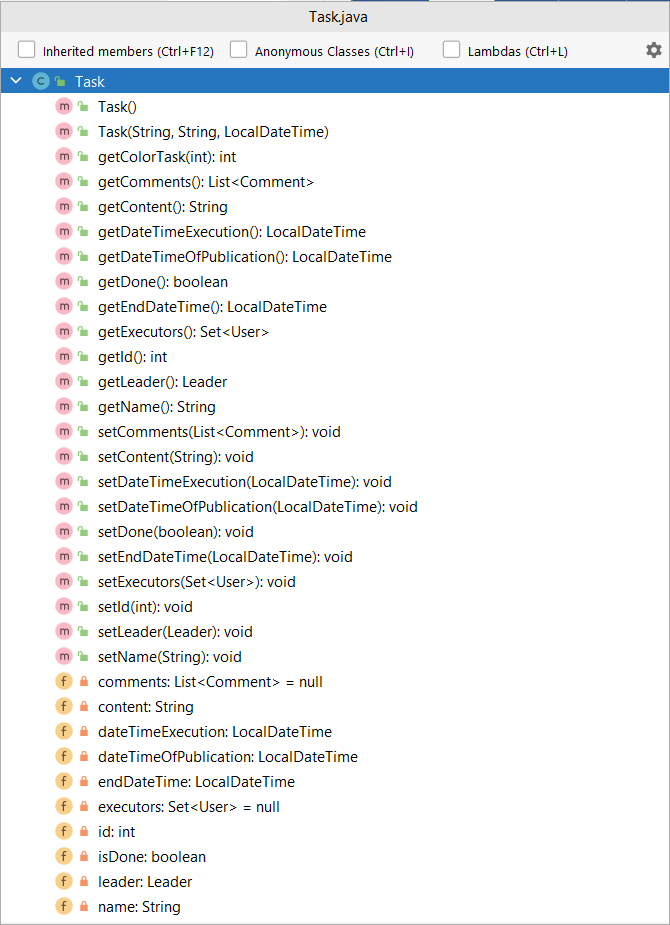


Рисунок 2.20 Внутрішня структура класу «Task»

«TaskCreatorActivity» являє собою графічний користувацький інтерфейс створення завдання (див. рисунок 2.21).

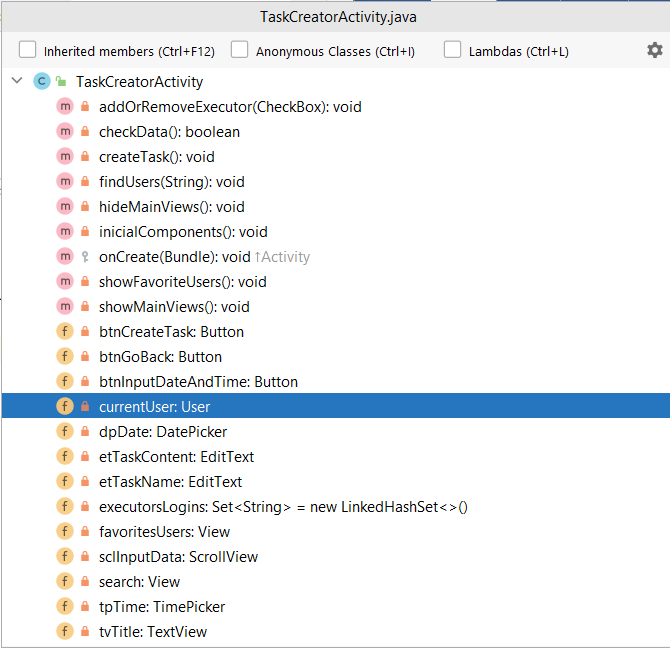


Рисунок 2.21 Внутрішня структура класу «TaskCreatorActivity»

«TasksViewerActivity» являє собою графічний користувацький інтерфейс перегляду завдань (див. рисунок 2.22).

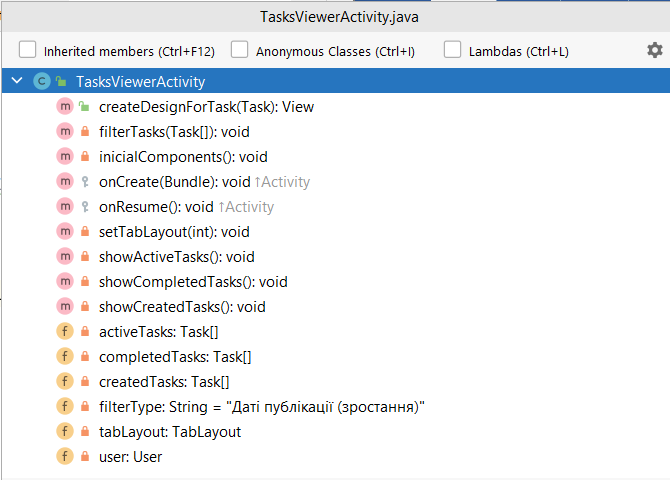


Рисунок 2.22 Внутрішня структура класу «TasksViewerActivity»

«TaskViewerActivity» являє собою графічний користувацький інтерфейс перегляду завдання (див. рисунок 2.23).

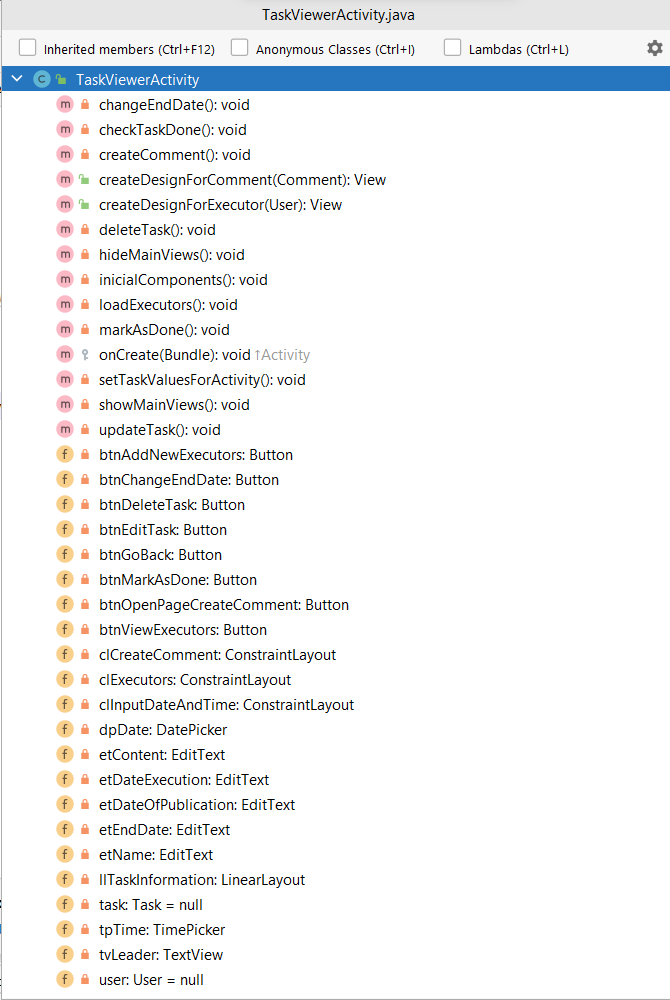


Рисунок 2.23 Внутрішня структура класу «TaskViewerActivity»

«FavoriteUsersViewerActivity» являє собою графічний користувацький інтерфейс перегляду вибраних користувачів (див. рисунок 2.24).

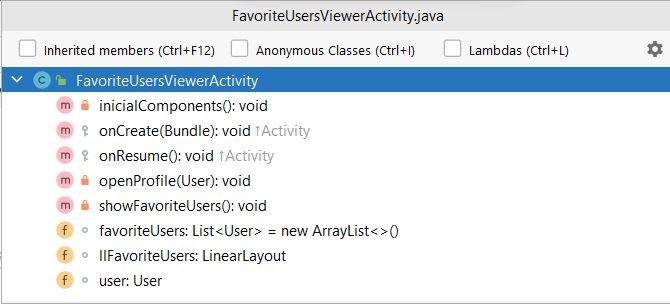


Рисунок 2.24 Внутрішня структура класу «FavoriteUsersViewerActivity»

«HomePageOfLeaderActivity» являє собою графічний користувацький інтерфейс кабінету керівника (див. рисунок 2.25).

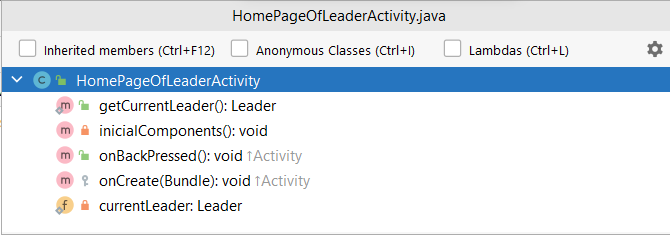


Рисунок 2.25 Внутрішня структура класу «HomePageOfLeaderActivity»

«HomePageOfWorkerActivity» являє собою графічний користувацький інтерфейс кабінету працівника (див. рисунок 2.26).

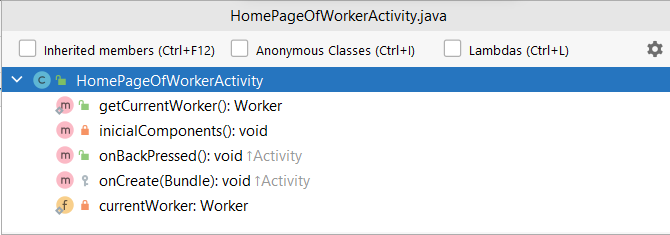


Рисунок 2.26 Внутрішня структура класу «HomePageOfWorkerActivity»

«Leader» являє собою керівника, який створює завдання (див. рисунок 2.27).

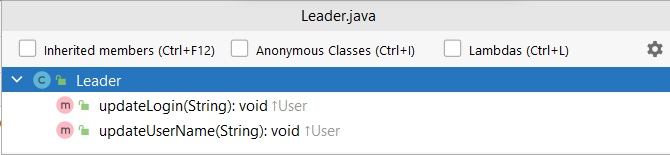


Рисунок 2.27 Внутрішня структура класу «Leader»

«ProfileViewerActivity» представляє собою графічний користувацький інтерфейс перегляду профілю користувача (див. рисунок 2.28).

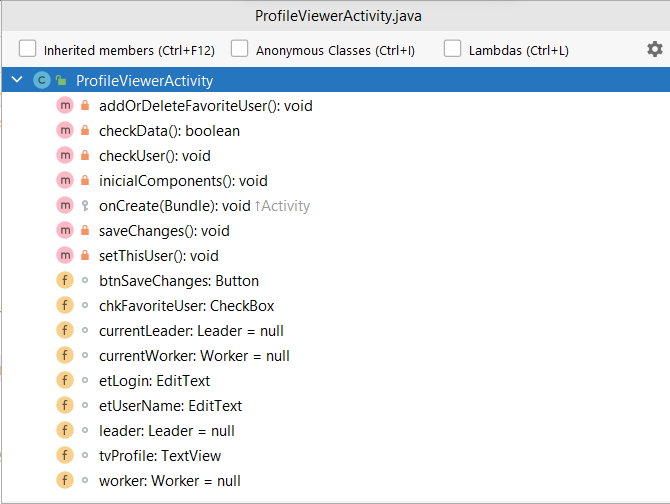


Рисунок 2.28 Внутрішня структура класу «ProfileViewerActivity»

«SearchUsersActivity» являє собою графічний користувацький інтерфейс пошуку користувачів (див. рисунок 2.29).

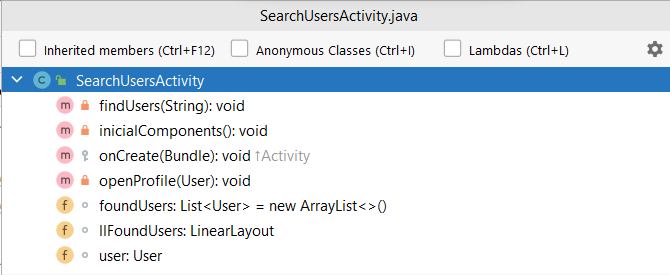


Рисунок 2.29 Внутрішня структура класу «SearchUsersActivity»

«User» являє собою користувача (див. рисунок 2.30).

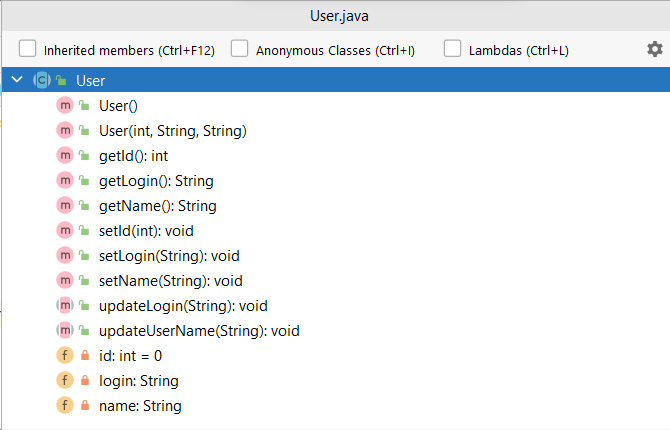


Рисунок 2.30 Внутрішня структура класу «User»

«Worker» являє собою працівника, який отримує від керівника завдання і виконує їх (див. рисунок 2.31).

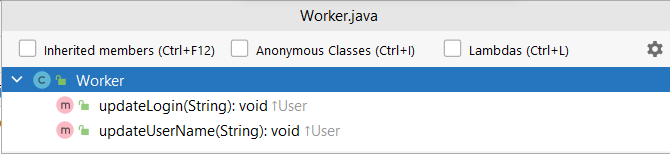


Рисунок 2.31 Внутрішня структура класу «Worker»

«NotificationService» являє собою сервіс перевірки повідомлень для користувача (див. рисунок 2.32).

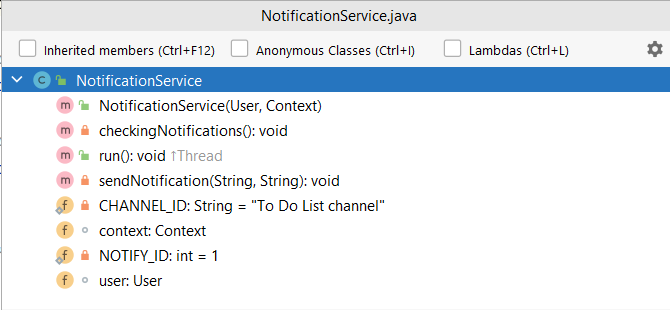


Рисунок 2.32 Внутрішня структура класу «NotificationService»

Список використаних елементів управління зображений на рисунку 2.33.



Рисунок 2.33 Список використаних елементів управління

У додатку є такі значні функції як:

* Увійти у програму з обліковим записом;
* Зареєструвати у програмі обліковий запис;
* Вибрати тип облікового запису як керівник або працівник;
* Можливість переглядати як свій профіль, так і чужий;
* Можливість редагувати свій профіль;
* Можливість шукати за допомогою пошуку користувачів;
* Можливість додавати та видаляти користувачів з вибраних;
* Можливість переглядати, виконувати, створювати та видаляти завдання;
* Можливість переглядати, додавати та видаляти виконавців завдання;
* Можливість писати та видаляти коментарі до завдань;
* Можливість фільтрувати завдання по датах.

## Фізична структура програми

Для того, щоб користувач зміг почати користуватися програмою потрібно всього один завантажувальний файл «app-debug.apk» (див. рисунок 2.34).

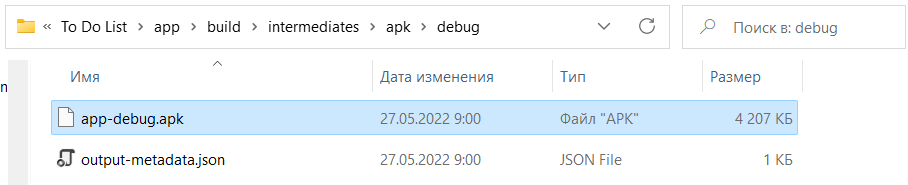


Рисунок 2.34 Завантажувальний файл програми

## Тестування проекту

Тестування відбувається на ОС Android, на трьох пристроях: Xiaomi Redmi 8 2019, Samsung Galaxy J6 2018, Xiaomi Mi Play 2019.

Тестування БД відбувається на пристроях з ОС Android.

Для тестування БД використав файли DDL та DML. DDL файл створює БД з таблицями, а DML файл заповнює таблиці даними. Тестові дані наближені до даних, які мають бути у програмі.

Тестування відбувалося двома методами: через елемент «Toast», використовував налагоджувач.

Налагоджувач використовував в цілях знаходження в якому місці додатка виникають помилки.

Результати функціонування додатка показані на рисунках 2.35–2.38

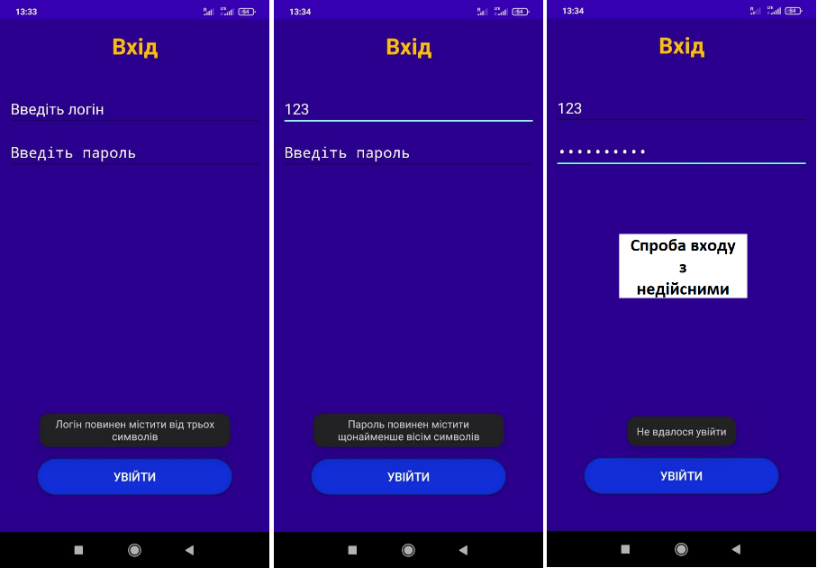


Рисунок 2.35 Тестування входу в програму

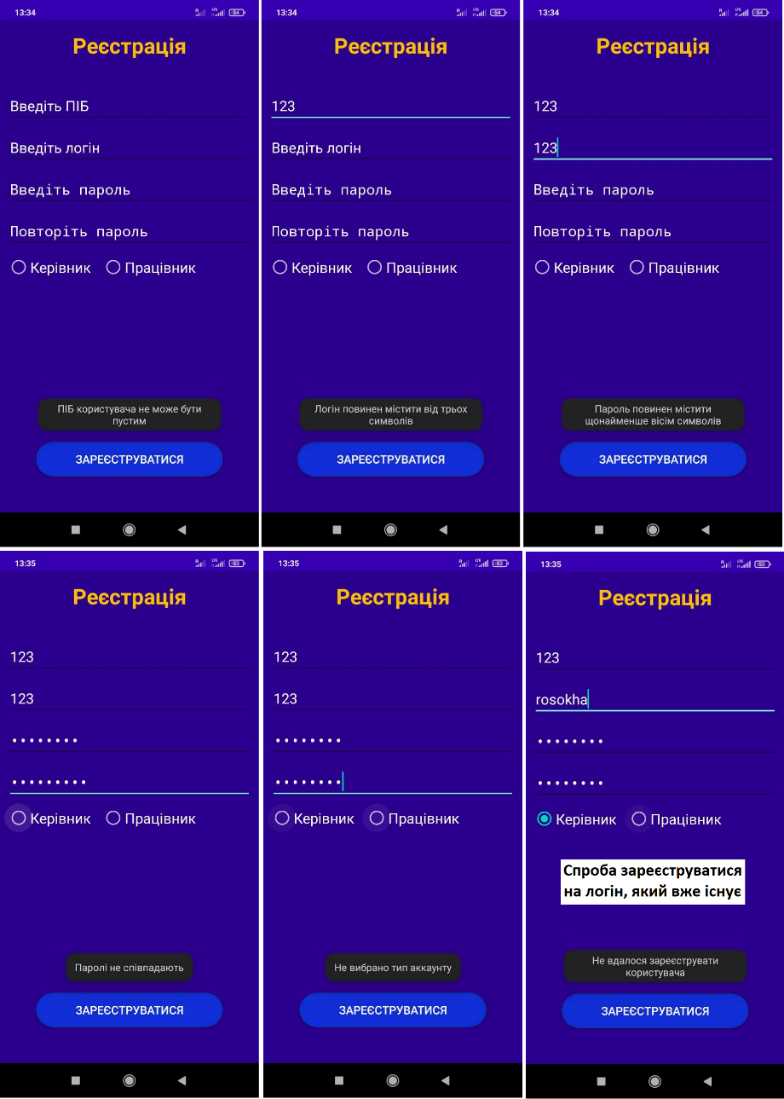


Рисунок 2.36 Тестування реєстрації в програму



Рисунок 2.37 Тестування зміни логіну

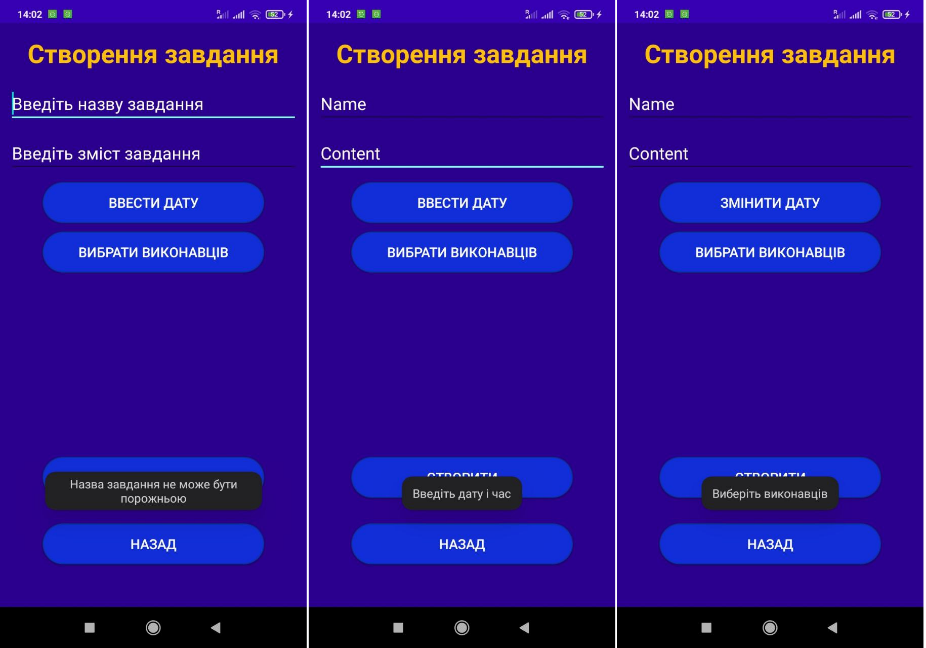


Рисунок 2.38 Тестування створення завдання

Валідація присутня у всіх полях введення. Для поля введення використав елемент «EditText», в якому можна задати максимальну довжину введення символі, та тип введення. Наприклад максимальну довжину введення символів використав для обмеження введення логіну на 50 символів, в такому випадку елемент «EditText» не дозволить ввести більше за 50 символів. Тип введення використав для унеможливлення при введенні тексту переходити на інший рядок.

Мінімальна довжина введення символі перевіряється за допомогою умовних операторів if та else. Якщо користувач введе менше символів чим потрібно, то на екрані з’явиться елемент «Toast» з текстом помилки.

Валідація дати та часу не може бути не коректною оскільки для введення дати та часу використовуються елементи календар та годинник.

Програма повинна виключатись тільки в єдиному випадку, коли відсутнє підключення до бази даних. Програма інформує про помилку при спробі ввести не коректні дані.

## Інтерфейс та керівництво користувача

Для того, щоб користувач зміг почати користуватися програмою потрібно тільки запустити один завантажувальний файл «app-debug.apk» і підтвердити завантаження. Після цього у користувача на робочому столі з’явиться повноцінний додаток (див. рисунок 2.39).



Рисунок 2.39 Вигляд додатка на робочому столі

У додатку наявно 9 вікон:

* головне вікно з вибором авторизації (див. рисунок 2.40);
* вікно входу (див. рисунок 2.35);
* вікно реєстрації (див. рисунок 2.36);
* кабінети керівника та працівника, це домашні сторінки користувачів (див. рисунок 2.41);
* вікно перегляду завдань, де можна переглядати: активні, виконані, і створені (якщо це керівник) завдання (див. рисунок 2.42);
* вікно перегляду і редагування профілю (див. рисунок 2.37);
* вікно створення завдання (див. рисунок 2.38);
* вікно перегляду вибраних користувачів (див. рисунок 2.43);
* вікно пошуку (див. рисунок 2.44);
* вікно перегляду завдання і коментарів до нього (див. рисунок 2.45);
* вікно перегляду виконавців (див. рисунок 2.46);
* вікно додавання виконавців до завдання (див. рисунок 2.47);
* вікно створення коментарів до завдання (див. рисунок 2.48);
* дизайн повідомлень (див. рисунок 2.49).

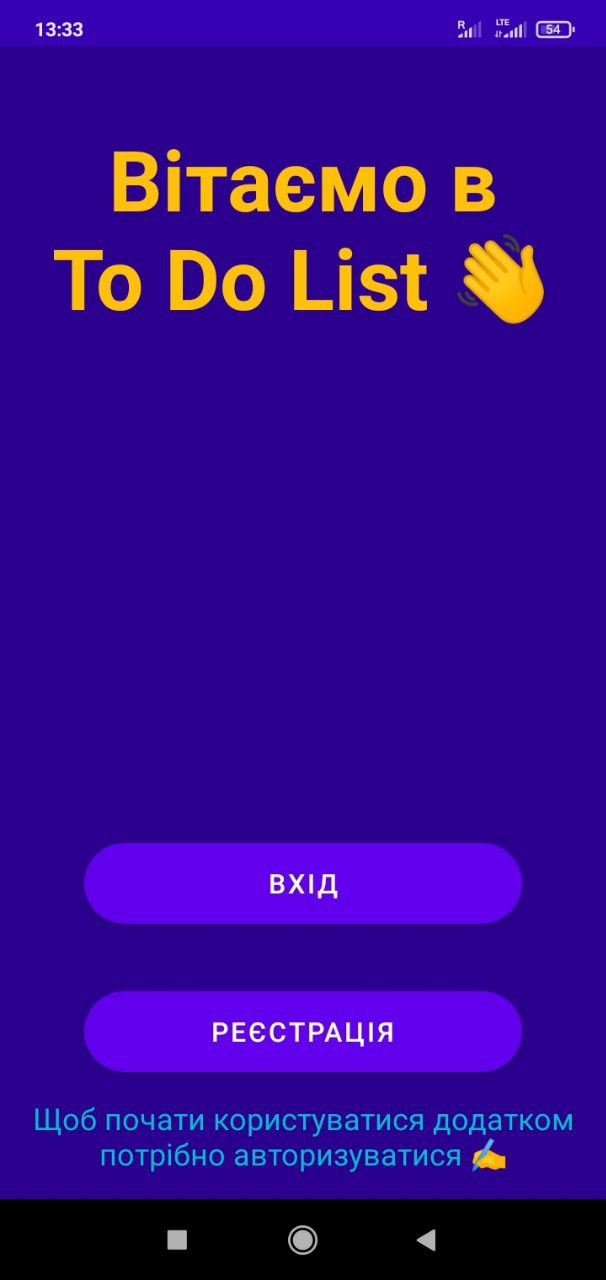


Рисунок 2.40 Головне вікно програми

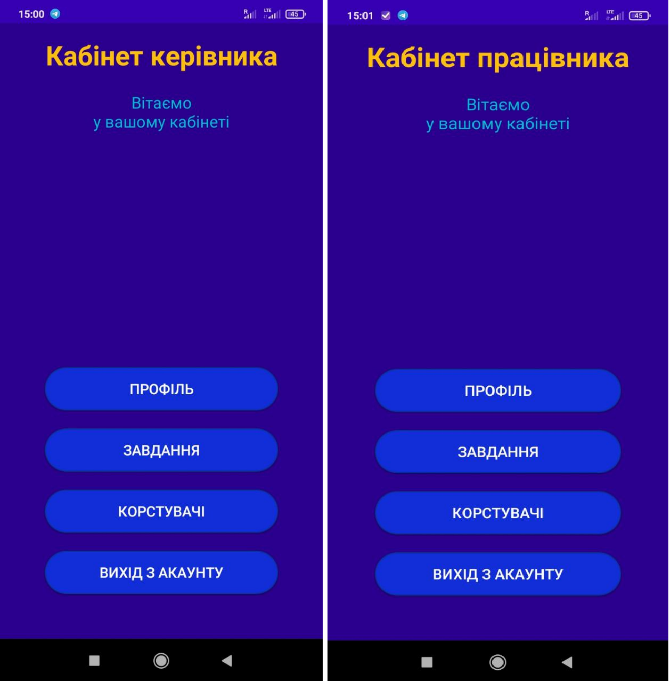


Рисунок 2.41 Вікна кабінетів працівника і керівника

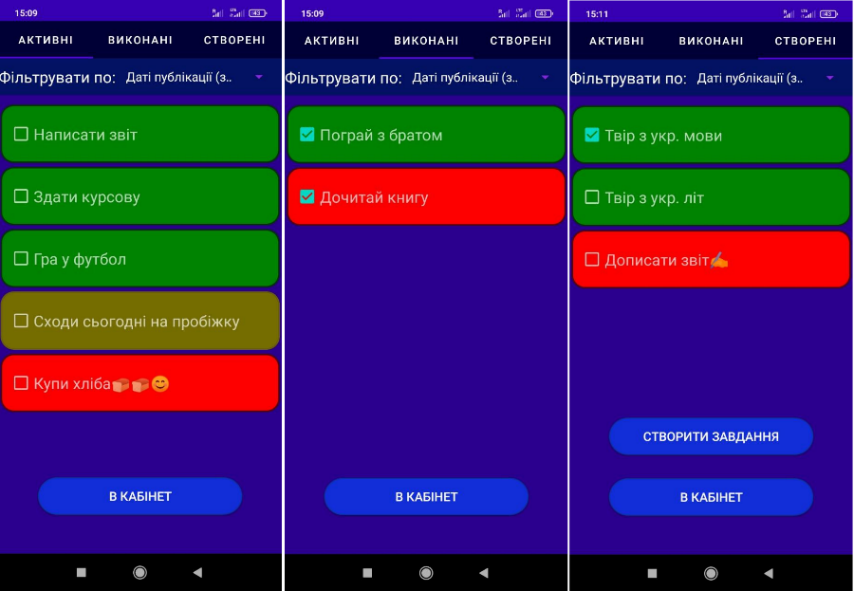


Рисунок 2.42 Вікно перегляду завдань

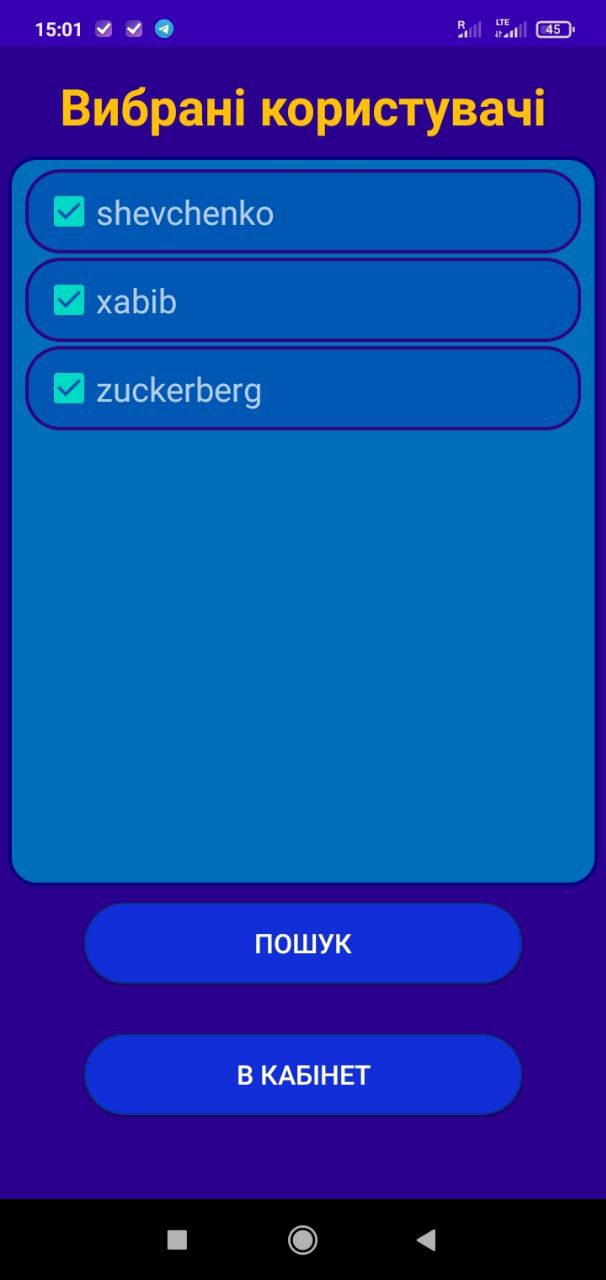


Рисунок 2.43 Вікно перегляду вибраних користувачів

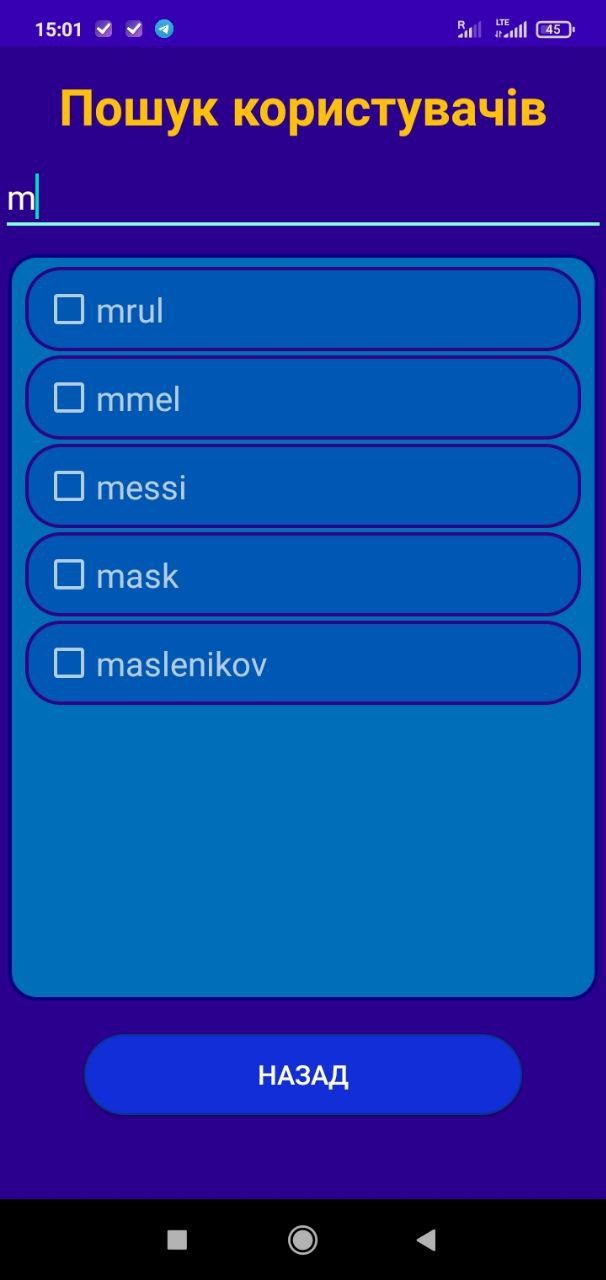


Рисунок 2.44 Вікно пошуку користувачів

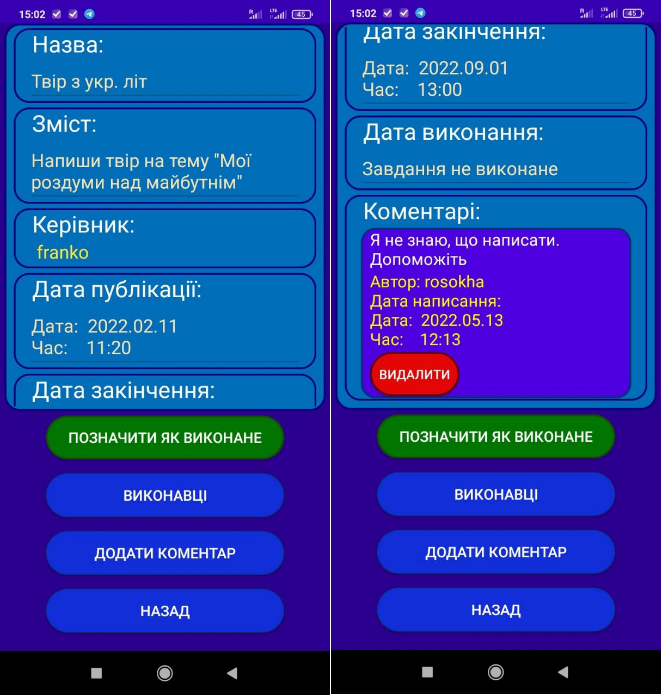


Рисунок 2.45 Вікно перегляду завдання і коментарів до нього

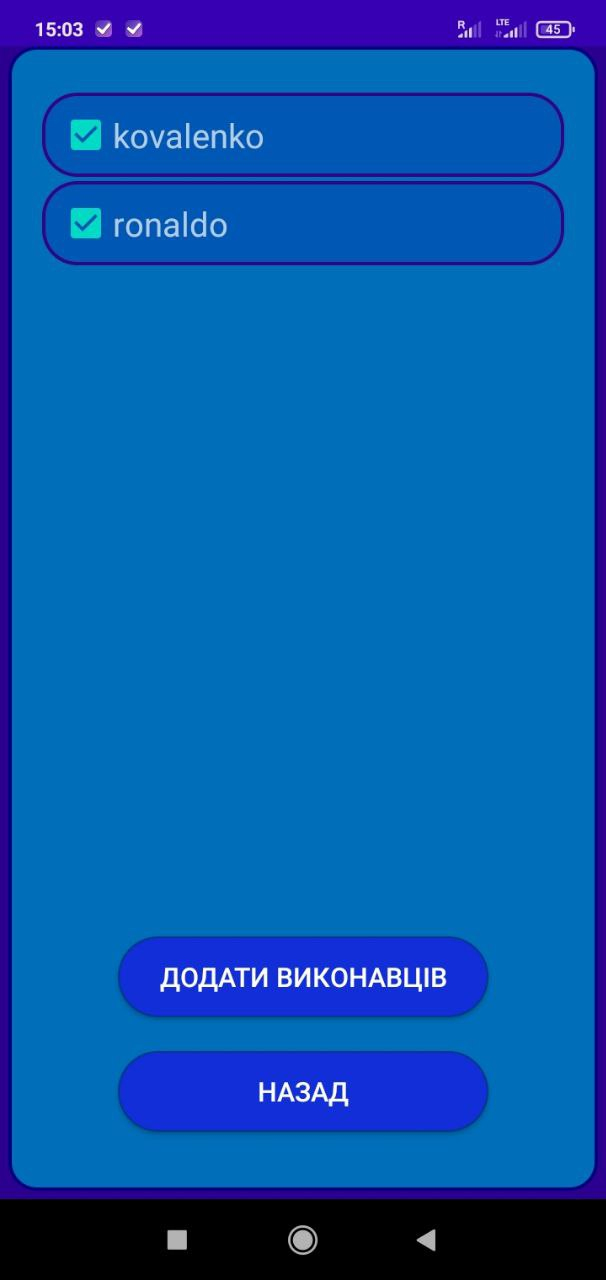


Рисунок 2.46 Вікно перегляду виконавців завдання



Рисунок 2.47 Вікно додавання виконавців до завдання

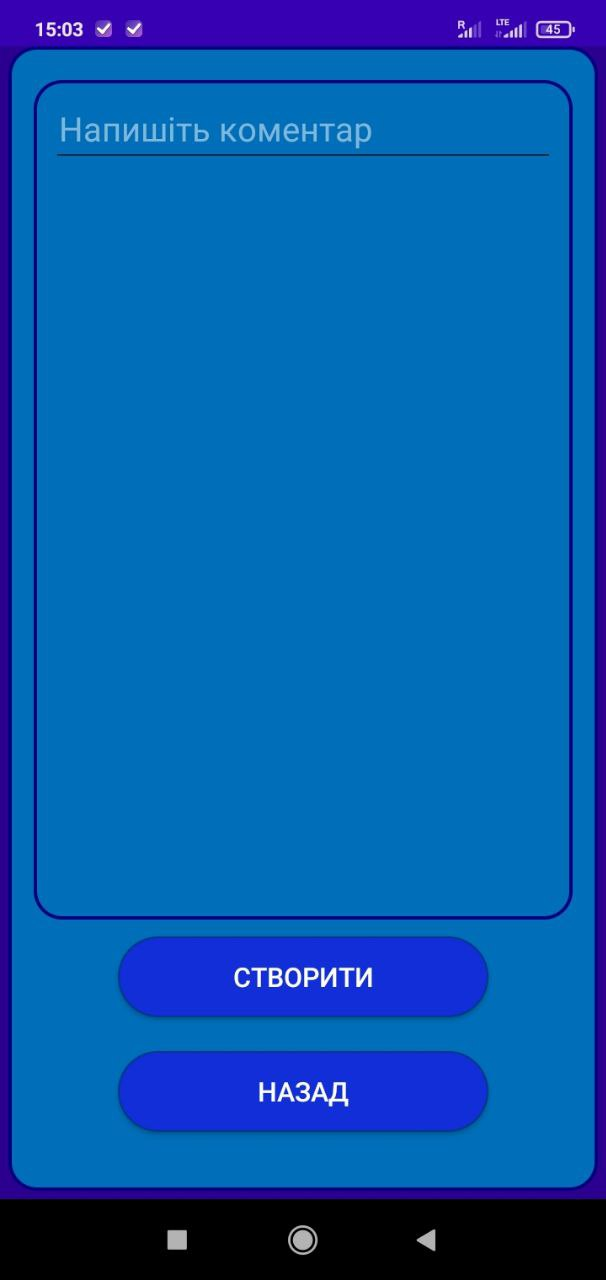


Рисунок 2.48 Вікно створення коментаря до завдання

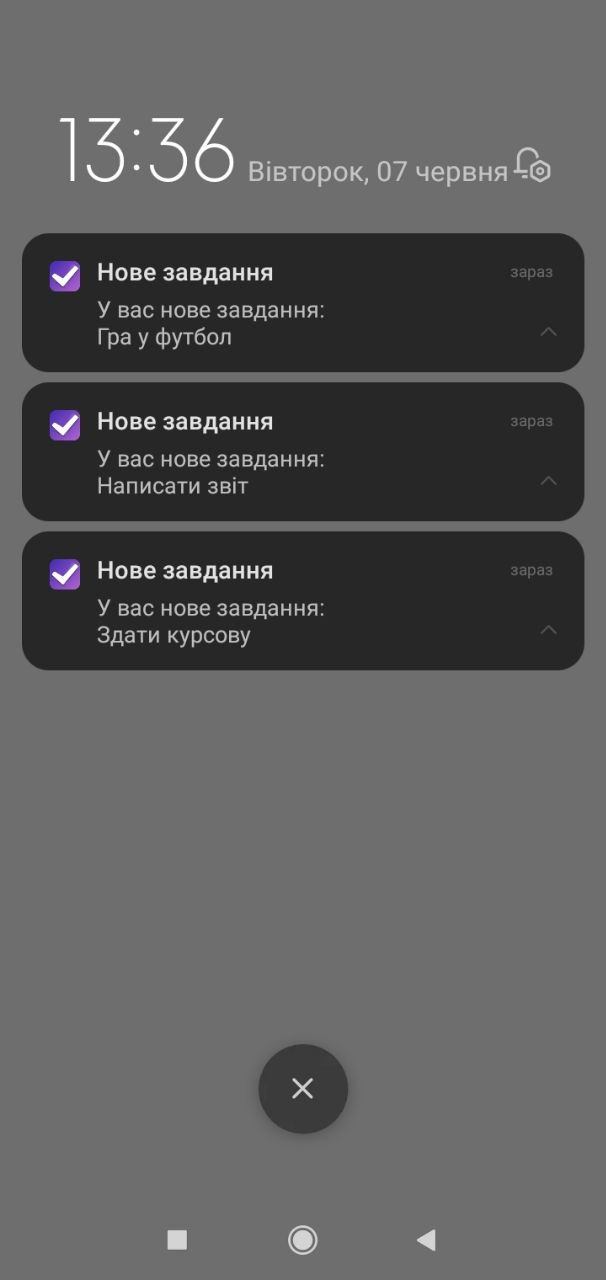


Рисунок 2.49 Дизайн повідомлень

ВИСНОВКИ

Під час виконання роботи було застосовано, вдосконалено та закріплено набуті навички з дисципліни «Об’єктно-орієнтоване програмування». Протягом виконання роботи було розроблено додаток для операційної системи Android: «Android To-Do list з нагадуванням про невиконані завдання». Основне завдання створеного додатка, яку він і виконує — дати користувачам змогу створювати та виконувати завдання з нагадуванням про них. Додаток був написаний на об’єктно-орієнтованій мові програмування Java з використанням реляційної бази даних SQLite.

Створений додаток відповідає таким критеріям:

* ООП підхід;
* Робота з базою даних;
* Коректна робота додатка.

Актуальність цього програмного продукту полягає у тому, щоб користувач зміг створювати або виконувати завдання, не забуваючи про них. Головне завдання даного програмного продукту — нагадувати користувачеві про невиконані завдання, щоб нічого не забувати і все завжди встигати.

Під час створення додатка були вдосконалені знання баз даних та навички роботи з ними. Навчився правильно робити нормалізацію таблиць у базі даних, щоб не виникали аномалії, та не порушувалася структура даних. Вдосконалив навички програмування об’єктно-орієнтованою мовою програмування Java. Отримав та закріпив нові навички роботи з повним життєвим циклом Android додатків. Отримав навички володіння середовищем програмування Android Studio.

Протягом створення мобільного додатка було виявлено багато помилок, які заважали коректній роботі програми, і були успішно усунені. Додаток має інформативний, зручний та простий для сприйняття інтерфейс.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Shawrohit602, Bhaluram18, Varshachoudhary. 2021. SQL | DDL, DQL, DML, DCL and TCL Commands. URL: <https://www.geeksforgeeks.org

/sql-ddl-dql-dml-dcl-tcl-commands/>. (дата звернення:12.04.2022);

1. Tproger. 2015. Шпаргалка за принципами ООП. URL: <https://tproger.ru

/translations/oop-principles-cheatsheet/>. (дата звернення: 25.02.2022);

1. Герасимик Іван. 2016. Що таке база даних? URL: <http://apeps.kpi.ua

/shco-take-basa-danykh>. (дата звернення: 29.04.2022);

1. Bogomol. 2015. Реляційна модель даних. URL: <іт.словник.укр

/index.php/Реляційна\_модель\_даних>. (дата звернення: 23.03.2022);

1. Раміс Ганієв. 2018. ER-діаграма - це Опис, види, правила побудови. URL: <http://hi-news.pp.ua/kompyuteri

/14668-er-dagrama-ce-opis-vidi-pravila-pobudovi.html>. (дата звернення: 09.05.2022);

1. Splend Apps. 14.11.2013. Список завдань. URL: <https://play.google.com/store/apps/details?id=com.splendapps.splendo> (дата звернення: 08.02.2022);
2. Google LLC. 14.01.2015. Google Classroom. URL: <https://play.google.com/store/apps

/details?id=com.google.android.apps.classroom>. (дата звернення: 08.02.2022);

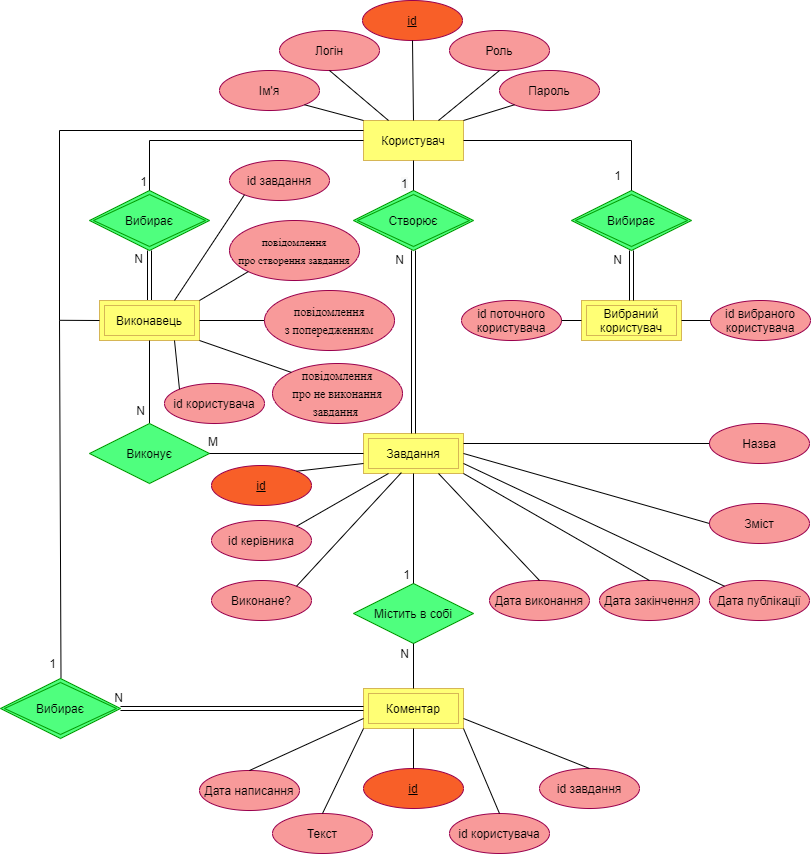
1. Metanit. 2012. Android. URL: < https://metanit.com/java/android/>. (дата звернення: 21.02.2022);
2. R class Tech. 27.04.2020. Бази даних. URL: <https://www.youtube.com

/playlist?list=PLf30vI0hEi1v435cBmZSHkr1QAJdOk9mb>. (дата звернення: 10.05.2022);

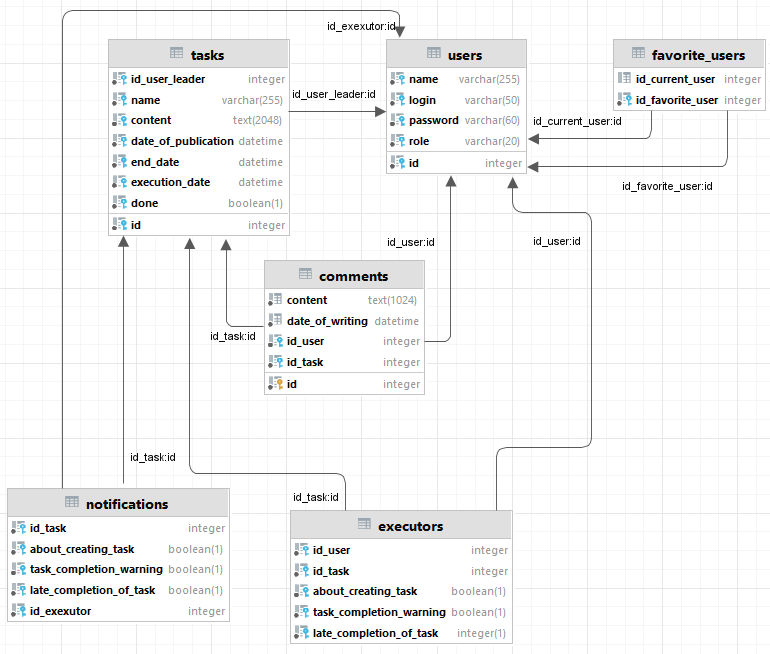
1. Neco Ru. 12.06.2019. Розробка додатків на Android Studio. URL: <https://www.youtube.com

/playlist?list=PLmjT2NFTgg1c-CC0l6GuvpH7\_2JZBxqzf>. (дата звернення: 14.03.2022).

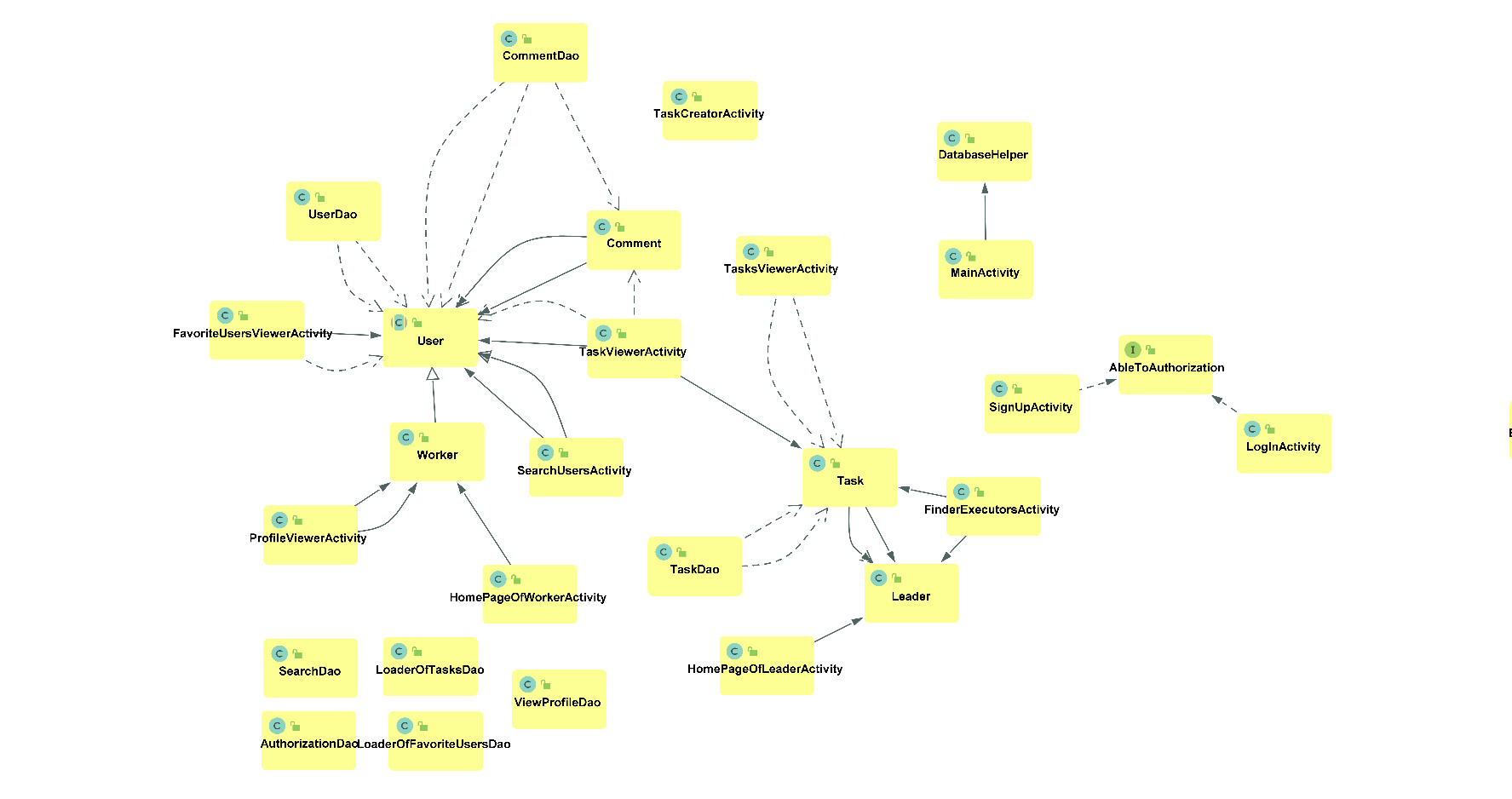
Додаток 1. Концептуальна схема



Додаток 2. Фізична схема



Додаток 3. Схема класів



Додаток 4. Лістинг програми

**MainActivity**

package com.rosokha.todolist.authorization;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.app.NotificationChannel;  
import android.app.NotificationManager;  
import android.content.Intent;  
import android.content.pm.ActivityInfo;  
import android.database.sqlite.SQLiteDatabase;  
import android.os.Build;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import android.widget.Toast;  
  
import com.rosokha.todolist.R;  
import com.rosokha.todolist.dao.AuthorizationDao;  
import com.rosokha.todolist.dao.DatabaseHelper;  
import com.rosokha.todolist.users.HomePageOfLeaderActivity;  
import com.rosokha.todolist.users.HomePageOfWorkerActivity;  
import com.rosokha.todolist.users.Leader;  
import com.rosokha.todolist.users.User;  
import com.rosokha.todolist.users.Worker;  
  
import java.io.FileInputStream;  
  
  
/\*\*  
 \* Клас з якого починається запуск програми.  
 \* Представляє собою графічний користувацький інтерфейс з вибором авторизації  
 \*  
 \* @see AppCompatActivity  
 \*/  
public class MainActivity extends AppCompatActivity {  
  
 /\*\*  
 \* Об'єкт представляючий собою допоміжний клас для бази даних  
 \*/  
 private static DatabaseHelper databaseHelper;  
  
 /\*\*  
 \* Об'єкт, що представляє собою базу даних  
 \*/  
 private static SQLiteDatabase DB;

/\*\*

\* Кнопка, при клацанні на яку здійснюється перехід до вікна входу у програму

\*/

private Button btnLogin;

/\*\*

\* Кнопка, при клацанні на яку здійснюється перехід до вікна реєстрації у програму

\*/

private Button btnSignUp;

/\*\*

\* Перевіряє чи вже був здійснений вхід з файлу

\*/

private static boolean isLoginFromFile = false;

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see AppCompatActivity#onCreate(Bundle)

\* @see AppCompatActivity#setContentView(View)

\* @see AppCompatActivity#setRequestedOrientation(int)

\* @see AppCompatActivity#getSupportActionBar()

\* @see MainActivity#inicialComponents()

\* @see DatabaseHelper

\* @see DatabaseHelper#create\_db()

\* @see AppCompatActivity#getApplicationContext()

\*/

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

getSupportActionBar().hide();

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);

inicialComponents();

createNotificationChannel();

databaseHelper = new DatabaseHelper(getApplicationContext());

databaseHelper.create\_db();

}

/\*\*

\* Викликається при поверненні користувача в дане activity

\*

\* @see AppCompatActivity#onResume()

\* @see MainActivity#readUserFromFileAndLogIn()

\*/

@Override

protected void onResume() {

super.onResume();

try {

DB = databaseHelper.open();

readUserFromFileAndLogIn();

isLoginFromFile = false;

} catch (Exception exception) {

Toast.makeText(this, "Не вдалося підключитися до бази даних", Toast.LENGTH\_LONG).show();

}

}

/\*\*

\* Метод, який ініціалізує компоненти файлу activity\_main.xml

\*

\* @see Intent

\* @see AppCompatActivity#startActivity(Intent)

\*/

private void inicialComponents() {

btnLogin = findViewById(R.id.btnLogin);

btnSignUp = findViewById(R.id.btnSignUp);

btnLogin.setOnClickListener((View v) -> {

Intent intent = new Intent(MainActivity.this, LogInActivity.class);

startActivity(intent);

});

btnSignUp.setOnClickListener((View v) -> {

Intent intent = new Intent(MainActivity.this, SignUpActivity.class);

startActivity(intent);

});

}

/\*\*

\* Читає файл з даними користувача, і здійснює вхід у програму

\*/

private void readUserFromFileAndLogIn() {

try {

FileInputStream fin = openFileInput("user.txt");

byte[] bytes = new byte[fin.available()];

fin.read(bytes);

String text = new String(bytes);

if (text.equals("")) {

return;

}

String[] userData = text.split(AbleToAuthorization.separator);

if (fin != null) {

fin.close();

}

User user = AuthorizationDao.doLogIn(userData[0], userData[1]);

if (!isLoginFromFile) {

isLoginFromFile = true;

if (user instanceof Leader) {

Intent intent = new Intent(MainActivity.this, HomePageOfLeaderActivity.class);

intent.putExtra(Leader.class.getSimpleName(), user);

startActivity(intent);

} else {

Intent intent = new Intent(MainActivity.this, HomePageOfWorkerActivity.class);

intent.putExtra(Worker.class.getSimpleName(), user);

startActivity(intent);

}

}

} catch (Exception ex) {

}

}

/\*\*

\* Створю канал для повідомлень

\*/

private void createNotificationChannel() {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

CharSequence name = "rosokha";

String description = "To do list channel";

int importance = NotificationManager.IMPORTANCE\_DEFAULT;

NotificationChannel channel = new NotificationChannel("To Do List channel", name, importance);

channel.setDescription(description);

NotificationManager notificationManager = getSystemService(NotificationManager.class);

notificationManager.createNotificationChannel(channel);

}

}

/\*\*

\* Setter для об'єкта бази даних

\*

\* @return об 'єкт бази даних

\* @see DatabaseHelper

\*/

public static SQLiteDatabase getDB() {

return DB;

}

/\*\*

\* Викликається при знищенні activity

\*

\* @see AppCompatActivity#onDestroy()

\* @see DatabaseHelper

\* @see DatabaseHelper#close()

\*/

@Override

public void onDestroy() {

super.onDestroy();

DB.close();

}

}

**LogInActivity**

package com.rosokha.todolist.authorization;

import android.annotation.SuppressLint;

import android.app.Activity;

import android.content.Context;

import android.content.Intent;

import android.content.pm.ActivityInfo;

import android.database.SQLException;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.Toast;

import androidx.annotation.Nullable;

import com.rosokha.todolist.R;

import com.rosokha.todolist.dao.AuthorizationDao;

import com.rosokha.todolist.users.HomePageOfLeaderActivity;

import com.rosokha.todolist.users.HomePageOfWorkerActivity;

import com.rosokha.todolist.users.Leader;

import com.rosokha.todolist.users.User;

import com.rosokha.todolist.users.Worker;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

/\*\*

\* Представляє собою графічний користувацький інтерфейс входу в програму

\*

\* @see Activity

\* @see AbleToAuthorization

\*/

public class LogInActivity extends Activity implements AbleToAuthorization {

/\*\*

\* Кнопка, після натиску якої здійснюється вхід

\*/

private Button btnLogIn;

/\*\*

\* Логін користувача

\*/

private EditText etLogin;

/\*\*

\* Пароль користуача

\*/

private EditText etPassword;

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see Nullable

\* @see Activity#onCreate(Bundle)

\* @see Activity#setContentView(View)

\* @see Activity#setRequestedOrientation(int)

\*/

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_login);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);//блокує поворот екрану

btnLogIn = findViewById(R.id.btnLogin);

etLogin = findViewById(R.id.etLogin);

etPassword = findViewById(R.id.etPassword);

btnLogIn.setOnClickListener((View view) -> doLogIn(etLogin.getText().toString(), hashPassword(etPassword.getText().toString())));

}

/\*\*

\* Здійснює вхід користувача

\*

\* @see AuthorizationDao

\* @see AuthorizationDao#doLogIn(String, String)

\* @see LogInActivity#checkData()

\* @see Toast

\* @see Toast#makeText(Context, int, int)

\* @see Toast#show()

\* @see Intent

\* @see Intent#putExtra(String, int)

\* @see Activity#startActivity(Intent)

\* @see Activity#finish()

\* @see User

\* @see Leader

\* @see Worker

\*/

@SuppressLint("Range")

private void doLogIn(String login, String password) {

if (checkData() == false) {

return;

}

User user = null;

try {

user = AuthorizationDao.doLogIn(login, password);

writeUserToFile(login, password);

} catch (SQLException ex) {

Toast.makeText(this, "Не вдалося увійти", Toast.LENGTH\_LONG).show();

return;

}

if (user instanceof Leader) {

Intent intent = new Intent(this, HomePageOfLeaderActivity.class);

intent.putExtra(Leader.class.getSimpleName(), user);

startActivity(intent);

} else {

Intent intent = new Intent(this, HomePageOfWorkerActivity.class);

intent.putExtra(Worker.class.getSimpleName(), user);

startActivity(intent);

}

Toast.makeText(this, "Вхід успішно здійснений", Toast.LENGTH\_LONG).show();

finish();

}

/\*\*

\* Перевіряє на коректність введених даних

\*

\* @return повертає true якщо дані введені коректно. Плвертає false якщо дані введені не коректно

\* @see Toast#makeText(Context, int, int)

\* @see Toast#show()

\*/

@Override

public boolean checkData() {

if (etLogin.getText().length() < 3) {

Toast.makeText(this, "Логін повинен містити від трьох символів", Toast.LENGTH\_LONG).show();

return false;

}

if (etPassword.getText().length() < 8) {

Toast.makeText(this, "Пароль повинен містити щонайменше вісім символів", Toast.LENGTH\_LONG).show();

return false;

}

return true;

}

/\*\*

\* Записує дані користувача у файл, щоб повторно не авторизуватися

\*

\* @param login логін

\* @param password пароль

\*/

private void writeUserToFile(String login, String password) {

String text = login + separator + password;

FileOutputStream fos = null;

try {

fos = openFileOutput(userFile, MODE\_PRIVATE);

} catch (FileNotFoundException e) {

return;

}

try {

fos.write(text.getBytes());

} catch (IOException e) {

return;

}

if (fos != null) {

try {

fos.close();

} catch (IOException e) {

return;

}

}

}

}

**SignUpActivity**

package com.rosokha.todolist.authorization;

import android.app.Activity;

import android.content.Context;

import android.content.Intent;

import android.content.pm.ActivityInfo;

import android.database.SQLException;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.RadioButton;

import android.widget.Toast;

import androidx.annotation.Nullable;

import com.rosokha.todolist.R;

import com.rosokha.todolist.dao.AuthorizationDao;

import com.rosokha.todolist.users.HomePageOfLeaderActivity;

import com.rosokha.todolist.users.HomePageOfWorkerActivity;

import com.rosokha.todolist.users.Leader;

import com.rosokha.todolist.users.Worker;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

/\*\*

\* Представляє собою графічний користувацький інтерфейс реєстрації

\*

\* @see Activity

\* @see AbleToAuthorization

\*/

public class SignUpActivity extends Activity implements AbleToAuthorization {

/\*\*

\* Кнопка, після натиску якої здійснюється реєстрація

\*/

private Button btnSignUp;

/\*\*

\* ПІБ користувача

\*/

private EditText etUserName;

/\*\*

\* Логін користувача

\*/

private EditText etLogin;

/\*\*

\* Пароль користуача

\*/

private EditText etPassword;

/\*\*

\* Повторений пароль користувача

\*/

private EditText etRepeatedPassword;

/\*\*

\* Тип аккаунту керівник

\*/

private RadioButton rbLeader;

/\*\*

\* Тип аккаунту працівник

\*/

private RadioButton rbWorker;

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see Nullable

\* @see Activity#onCreate(Bundle)

\* @see Activity#setContentView(View)

\* @see Activity#setRequestedOrientation(int)

\*/

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_sign\_up);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);//блокує поворот екрану

btnSignUp = findViewById(R.id.btnSignUp);

etUserName = findViewById(R.id.etUserName);

etLogin = findViewById(R.id.etLogin);

etPassword = findViewById(R.id.etPassword);

etRepeatedPassword = findViewById(R.id.etRepeatedPassword);

rbLeader = findViewById(R.id.rbLeader);

rbWorker = findViewById(R.id.rbWorker);

btnSignUp.setOnClickListener((View view) -> toRegister());

}

/\*\*

\* Реєструє користувача

\*

\* @see SignUpActivity#checkData()

\* @see AuthorizationDao

\* @see AuthorizationDao#doRegister(String, String, String, String)

\* @see AuthorizationDao#doLogIn(String, String)

\* @see Intent

\* @see Intent#putExtra(String, int)

\* @see Activity#startActivity(Intent)

\* @see Activity#finish()

\*/

private void toRegister() {

if (checkData() == false) {

return;

}

try {

AuthorizationDao.doRegister(etUserName.getText().toString(), etLogin.getText().toString(),

hashPassword(etPassword.getText().toString()), rbLeader.isChecked() ? "Leader" : "Worker");

writeUserToFile(etLogin.getText().toString(), hashPassword(etPassword.getText().toString()));

} catch (SQLException ex) {

Toast.makeText(this, "Не вдалося зареєструвати користувача", Toast.LENGTH\_LONG).show();

return;

}

if (rbLeader.isChecked()) {

try {

Intent intent = new Intent(SignUpActivity.this, HomePageOfLeaderActivity.class);

intent.putExtra(Leader.class.getSimpleName(), AuthorizationDao.doLogIn(etLogin.getText().toString(),

hashPassword(etPassword.getText().toString())));

startActivity(intent);

} catch (Exception ex) {

Toast.makeText(this, "Не вдалося увійти", Toast.LENGTH\_LONG).show();

}

} else {

try {

Intent intent = new Intent(SignUpActivity.this, HomePageOfWorkerActivity.class);

intent.putExtra(Worker.class.getSimpleName(), AuthorizationDao.doLogIn(etLogin.getText().toString(),

hashPassword(etPassword.getText().toString())));

startActivity(intent);

} catch (Exception ex) {

Toast.makeText(this, "Не вдалося увійти", Toast.LENGTH\_LONG).show();

}

}

Toast.makeText(this, "Користувач успішно зареєстрований", Toast.LENGTH\_LONG).show();

finish();

}

/\*\*

\* Перевіряє на коректність введених даних

\*

\* @return повертає true якщо дані введені коректно. Плвертає false якщо дані введені не коректно

\* @see AuthorizationDao

\* @see AuthorizationDao#checkForLoginUniqueness(String)

\* @see Toast#makeText(Context, int, int)

\* @see Toast#show()

\*/

@Override

public boolean checkData() {

if (etUserName.getText().length() == 0) {

Toast.makeText(this, "ПІБ користувача не може бути пустим", Toast.LENGTH\_LONG).show();

return false;

}

if (etLogin.getText().length() < 3) {

Toast.makeText(this, "Логін повинен містити від трьох символів", Toast.LENGTH\_LONG).show();

return false;

}

if (etPassword.getText().length() < 8) {

Toast.makeText(this, "Пароль повинен містити щонайменше вісім символів", Toast.LENGTH\_LONG).show();

return false;

}

if (!etPassword.getText().toString().equals(etRepeatedPassword.getText().toString())) {

Toast.makeText(this, "Паролі не співпадають", Toast.LENGTH\_LONG).show();

return false;

}

if (!rbLeader.isChecked() && !rbWorker.isChecked()) {

Toast.makeText(this, "Не вибрано тип аккаунту", Toast.LENGTH\_LONG).show();

return false;

}

try {

if (AuthorizationDao.checkForLoginUniqueness(etLogin.getText().toString())) { return true; }

} catch (SQLException ex){

Toast.makeText(this, "Користувач з таким логіном вже існує", Toast.LENGTH\_LONG).show();

return false;

}

return true;

}

/\*\*

\* Записує дані користувача у файл, щоб повторно не авторизуватися

\* @param login логін

\* @param password пароль

\*/

private void writeUserToFile(String login, String password){

String text = login + separator + password;

FileOutputStream fos = null;

try {

fos = openFileOutput(userFile, MODE\_PRIVATE);

} catch (FileNotFoundException e) {

Toast.makeText(this, "Не вдалося знайти файл", Toast.LENGTH\_LONG).show();

}

try {

fos.write(text.getBytes());

} catch (IOException e) {

Toast.makeText(this, "Не вдалося записати у файл", Toast.LENGTH\_LONG).show();

}

if (fos != null) {

try {

fos.close();

} catch (IOException e) {

Toast.makeText(this, "Не вдалося закрити FileOutputStream", Toast.LENGTH\_LONG).show();

}

}

}

}

**AbleToAuthorization**

package com.rosokha.todolist.authorization;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

/\*\*

\* Інтерфейс, що представлює собою здатність авторизовуватися

\*/

interface AbleToAuthorization {

String separator = "⋱Ꙟ⋰";

String userFile = "user.txt";

/\*\*

\* Перевірка на коректність введених даних

\*

\* @return повертає true якщо дані введені коректно. Плвертає false якщо дані введені не коректно

\*/

boolean checkData();

/\*\*

\* Хешує пароль користувача методом "SHA-1"

\*

\* @param password пароль, який потрібно захешувати

\* @return захешований пароль

\* @see MessageDigest

\* @see MessageDigest#digest(byte[]) MessageDigest#digest(byte[])

\* @see StringBuilder

\* @see StringBuilder#append(int) StringBuilder#append(int)

\*/

default String hashPassword(String password) {

MessageDigest sha1 = null;

try {

sha1 = MessageDigest.getInstance("SHA-1");

} catch (NoSuchAlgorithmException ex) {

return password;

}

byte[] bytes = sha1.digest(password.getBytes());

StringBuilder stringBuilder = new StringBuilder();

for (byte b : bytes) {

stringBuilder.append(String.format("%02X", b));

}

return stringBuilder.toString();

}

}

**AuthorizationDao**

package com.rosokha.todolist.dao;

import android.annotation.SuppressLint;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import com.rosokha.todolist.authorization.LogInActivity;

import com.rosokha.todolist.authorization.MainActivity;

import com.rosokha.todolist.users.Leader;

import com.rosokha.todolist.users.User;

import com.rosokha.todolist.users.Worker;

/\*\*

\* Клас авторизації, який працює з базою даних

\*/

public class AuthorizationDao {

/\*\*

\* Об'єкт бази даних

\*/

private static SQLiteDatabase DB;

static {

DB = MainActivity.getDB();

}

/\*\*

\* Реєструє користувача вносячи дані у базу даних

\*

\* @param name ПІБ користувача

\* @param login логін користувача

\* @param password пароль користувача

\* @param role роль користувача

\* @throws SQLException помилка під час реєстрації

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String)

\* @see SQLException

\*/

public static void doRegister(String name, String login, String password, String role) throws SQLException {

DB.execSQL("INSERT INTO users (name, login, password, role) " +

" VALUES (?,?,?,?)",

new String[]{name, login, password, role});

}

/\*\*

\* Перевіряє унікальність логіну користувача у базі даних

\*

\* @param login the login

\* @return поверта true, якщо такого логіну ще не має. Повертає false якщо такий логін вже існує

\* @throws SQLException the sql exception

\* @see Cursor

\* @see Cursor#moveToFirst() Cursor#moveToFirst()

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see SQLException

\*/

public static boolean checkForLoginUniqueness(String login) throws SQLException {

Cursor query = DB.rawQuery("SELECT login" +

" FROM users" +

" WHERE login = ?;",

new String[]{login});

return query.moveToFirst() ? false : true;

}

/\*\*

\* Здійснює вхід користувача

\*

\* @param login логін користувача

\* @param password пароль користувача

\* @return користувача user

\* @throws SQLException помилка під час входу

\* @see LogInActivity#checkData() LogInActivity#checkData()

\* @see Cursor

\* @see Cursor#moveToFirst() Cursor#moveToFirst()

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see SQLException

\*/

@SuppressLint("Range")

public static User doLogIn(String login, String password) throws SQLException {

Cursor query = DB.rawQuery("SELECT \*" +

" FROM users" +

" WHERE login = ?" +

" AND password = ?" +

" LIMIT 1;",

new String[]{login, password});

if (query.moveToFirst()) {

if (query.getString(query.getColumnIndex("role")).equals("Leader")) {

Leader leader = new Leader();

leader.setId(query.getInt(query.getColumnIndex("id")));

leader.setName(query.getString(query.getColumnIndex("name")));

leader.setLogin(query.getString(query.getColumnIndex("login")));

query.close();

return leader;

}

if (query.getString(query.getColumnIndex("role")).equals("Worker")){

Worker worker = new Worker();

worker.setId(query.getInt(query.getColumnIndex("id")));

worker.setName(query.getString(query.getColumnIndex("name")));

worker.setLogin(query.getString(query.getColumnIndex("login")));

query.close();

return worker;

}

} else {

throw new SQLException();

}

return null;

}

}

**CommentDao**

package com.rosokha.todolist.dao;

import android.annotation.SuppressLint;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import android.os.Build;

import com.rosokha.todolist.authorization.MainActivity;

import com.rosokha.todolist.task.Comment;

import com.rosokha.todolist.task.Task;

import com.rosokha.todolist.users.User;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Клас коментаря, який працює з базою даних

\*/

public class CommentDao {

/\*\*

\* Об'єкт бази даних

\*/

private static SQLiteDatabase DB;

static {

DB = MainActivity.getDB();

}

/\*\*

\* Створює коментар

\*

\* @param task завдання, до якого створюється коментар

\* @param user користувач, який пише коментар

\* @param content зміст коментаря

\* @throws SQLException помилка при створенні коментаря

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\* @see Task

\* @see User

\*/

public static void createComment(Task task, User user, String content) throws SQLException {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

DB.execSQL(" INSERT INTO comments (content, date\_of\_writing, id\_user, id\_task)" +

"VALUES (?,?,?,?);",

new Object[]{content, LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")),

user.getId(), task.getId()});

}

}

/\*\*

\* Повертає список коментарів до завдання

\*

\* @param task завдання до якого потрібно завантажити коментарі

\* @return список коментарів

\* @throws SQLException помилка при створенні списку коментарів

\* @see SQLException

\* @see Comment

\* @see Cursor

\* @see Cursor#moveToNext() Cursor#moveToNext()

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\* @see Task

\*/

@SuppressLint("Range")

public static List<Comment> getListOfComments(Task task) throws SQLException {

List<Comment> commentss = new ArrayList<>();

Cursor query = DB.rawQuery("SELECT \* " +

" FROM comments " +

" WHERE id\_task = ?;",

new String[]{task.getId() + ""});

while (query.moveToNext()) {

Comment comment = new Comment();

comment.setId(query.getInt(query.getColumnIndex("id")));

comment.setContent(query.getString(query.getColumnIndex("content")));

comment.setTask(task);

comment.setUser(UserDao.getUserById(query.getInt(query.getColumnIndex("id\_user"))));

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

comment.setDateTimeOfWriting(LocalDateTime.parse(query.getString(query.getColumnIndex("date\_of\_writing")),

DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

}

commentss.add(comment);

}

return commentss;

}

/\*\*

\* Видаляє коментар

\*

\* @param comment коментар, який потрібно видалити

\* @throws SQLException помилка під час видалення коментаря

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\* @see Comment

\*/

public static void deleteComment(Comment comment) throws SQLException {

DB.execSQL("DELETE FROM comments " +

" WHERE id = ?;",

new Integer[]{comment.getId()});

}

}

**DatabaseHelper**

package com.rosokha.todolist.dao;

import android.content.Context;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.util.Log;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.sql.SQLException;

/\*\*

\* Допоміжний клас для роботи з базою даних

\*

\* @see SQLiteOpenHelper

\*/

public class DatabaseHelper extends SQLiteOpenHelper {

/\*\*

\* Вікно програми

\*/

private Context myContext;

/\*\*

\* Назва бази даних

\*/

public static final String DB\_NAME = "ToDoList.db";

/\*\*

\* Повний шлях до внутрішнього файлу бази даних

\*/

public static final String DB\_PATH = "/data/data/com.rosokha.todolist/" + DB\_NAME;

/\*\*

\* Версія бази даних

\*/

public static final int VERSION = 1;

/\*\*

\* Конструктор класу DatabaseHelper

\*

\* @param context вікно програми

\*/

public DatabaseHelper(Context context) {

super(context, DB\_NAME, null, VERSION);

this.myContext = context;

}

/\*\*

\* Створює бд при необхідності

\*

\* @param db база даних

\*/

@Override

public void onCreate(SQLiteDatabase db) {

}

/\*\*

\* Оновлює бд

\*

\* @param db база даних

\* @param oldVersion стара версія

\* @param newVersion нова версія

\*/

@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

}

/\*\*

\* Створює внутрішній файл бази даних

\*/

public void create\_db() {

File file = new File(DB\_PATH);

if (!file.exists()) {

//Підключення локальної бд як потік

try (InputStream myInput = myContext.getAssets().open(DB\_NAME);

// Відкриваємо порожню бд

OutputStream myOutput = new FileOutputStream(DB\_PATH)) {

// Побайтово копіюються данні

byte[] buffer = new byte[1024];

int length;

while ((length = myInput.read(buffer)) > 0) {

myOutput.write(buffer, 0, length);

}

myOutput.flush();

} catch (IOException ex) {

Log.d("DatabaseHelper", ex.getMessage());

}

}

}

/\*\*

\* Відкриває внутрішню базу даних

\*

\* @return sq lite database

\* @throws SQLException the sql exception

\* @throws IllegalStateException the illegal state exception

\*/

public SQLiteDatabase open() throws SQLException, IllegalStateException {

return SQLiteDatabase.openDatabase(DB\_PATH, null, SQLiteDatabase.OPEN\_READWRITE);

}

}

**LoaderOfFavoriteUsersDao**

package com.rosokha.todolist.dao;

import android.annotation.SuppressLint;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import com.rosokha.todolist.authorization.MainActivity;

import com.rosokha.todolist.users.Leader;

import com.rosokha.todolist.users.User;

import com.rosokha.todolist.users.Worker;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Клас працюючий з вибраними користувачами у базі даних

\*/

public class LoaderOfFavoriteUsersDao {

/\*\*

\* Об'єкт бази даних

\*/

private static SQLiteDatabase DB;

static {

DB = MainActivity.getDB();

}

/\*\*

\* Завантажує вибраних користувачів

\*

\* @param idCurrentUser id поточного користувача, для якого потрібно загрузити вибраних користувачів

\* @return список вибраних користувачів

\* @throws SQLException the sql exception

\* @see User

\* @see Leader

\* @see Worker

\* @see SQLException

\* @see Cursor

\* @see Cursor#moveToNext() Cursor#moveToNext()Cursor#moveToNext()Cursor#moveToNext()

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])SQLiteDatabase#rawQuery(String, String[])SQLiteDatabase#rawQuery(String, String[])

\*/

@SuppressLint("Range")

public static List<User> loadFavoriteUsers(int idCurrentUser) throws SQLException{

List<User> favoriteUsers = new ArrayList<>();

try {

Cursor query = DB.rawQuery("SELECT favorite\_users.\*, users.\*" +

" FROM favorite\_users " +

" JOIN users" +

" ON favorite\_users.id\_favorite\_user = users.id" +

" WHERE favorite\_users.id\_current\_user = " + idCurrentUser +

" GROUP BY favorite\_users.id\_favorite\_user;",

null);

while (query.moveToNext()) {

if (query.getString(query.getColumnIndex("role")).equals("Leader")) {

Leader tempLeader = new Leader();

tempLeader.setId(query.getInt(query.getColumnIndex("id")));

tempLeader.setName(query.getString(query.getColumnIndex("name")));

tempLeader.setLogin(query.getString(query.getColumnIndex("login")));

favoriteUsers.add(tempLeader);

} else {

Worker tempWorker = new Worker();

tempWorker.setId(query.getInt(query.getColumnIndex("id")));

tempWorker.setName(query.getString(query.getColumnIndex("name")));

tempWorker.setLogin(query.getString(query.getColumnIndex("login")));

favoriteUsers.add(tempWorker);

}

}

query.close();

} catch (SQLException ex){

throw new SQLException();

}

return favoriteUsers;

}

}

**LoaderOfTasksDao**

package com.rosokha.todolist.dao;

import android.annotation.SuppressLint;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import android.os.Build;

import com.rosokha.todolist.authorization.MainActivity;

import com.rosokha.todolist.task.Task;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Клас який завантажує завдання з бази даних

\*/

public class LoaderOfTasksDao {

/\*\*

\* Об'єкт бази даних

\*/

private static SQLiteDatabase DB;

static {

DB = MainActivity.getDB();

}

/\*\*

\* Повертає список активних завдань

\*

\* @param idCurrentUser ідентифікатор користувача, для якого потрібно створити список завдань

\* @return список активних завдань

\* @throws SQLException помилка при створенні списку активних завдань

\* @see SQLException

\* @see Task

\* @see Cursor

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see LoaderOfTasksDao#getListOfTasks(Cursor) LoaderOfTasksDao#getListOfTasks(Cursor)

\*/

@SuppressLint("Range")

public static List<Task> getListOfActiveTasks(int idCurrentUser) throws SQLException {

List<Task> tasks = new ArrayList<>();

Cursor query = DB.rawQuery("SELECT tasks.\*, executors.\*" +

" FROM tasks " +

" JOIN executors" +

" ON tasks.id = executors.id\_task" +

" WHERE executors.id\_user = ?" +

" AND tasks.done = 0;",

new String[]{idCurrentUser + ""});

while (query.moveToNext()) {

Task task = new Task();

task.setId(query.getInt(query.getColumnIndex("id")));

task.setName(query.getString(query.getColumnIndex("name")));

task.setContent(query.getString(query.getColumnIndex("content")));

task.setDone(query.getInt(query.getColumnIndex("done")) == 1);

task.setLeader(UserDao.getLeaderById(query.getInt(query.getColumnIndex("id\_user\_leader"))));

task.setExecutors(TaskDao.getListOfExecutorsForTask(task));

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

task.setDateTimeOfPublication(LocalDateTime.parse(query.getString(query.getColumnIndex("date\_of\_publication")),

DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

task.setEndDateTime(LocalDateTime.parse(query.getString(query.getColumnIndex("end\_date")),

DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

if (query.getString(query.getColumnIndex("execution\_date")) != null) {

task.setDateTimeExecution(LocalDateTime.parse(query.getString(query.getColumnIndex("execution\_date")),

DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

} else {

task.setDateTimeExecution(null);

}

}

boolean f = query.getInt(query.getColumnIndex("about\_creating\_task")) == 1;

boolean s = query.getInt(query.getColumnIndex("task\_completion\_warning")) == 1;

boolean t = query.getInt(query.getColumnIndex("late\_completion\_of\_task")) == 1;

task.setNotificationAboutCreatingTask(f);

task.setNotificationTaskCompletionWarning(s);

task.setNotificationLateCompletionOfTask(t);

tasks.add(task);

}

return tasks;

}

/\*\*

\* Повертає список виконаних завдань

\*

\* @param idCurrentUser ідентифікатор користувача, для якого потрібно створити список завдань

\* @return список виконаних завдань

\* @throws SQLException помилка при створенні списку виконаних завдань

\* @see SQLException

\* @see Task

\* @see Cursor

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see LoaderOfTasksDao#getListOfTasks(Cursor) LoaderOfTasksDao#getListOfTasks(Cursor)

\*/

public static List<Task> getListOfСompletedTasks(int idCurrentUser) throws SQLException {

List<Task> tasks = null;

Cursor query = DB.rawQuery("SELECT tasks.\*" +

" FROM tasks " +

" JOIN executors" +

" ON tasks.id = executors.id\_task" +

" WHERE executors.id\_user = ?" +

" AND tasks.done = 1;",

new String[]{idCurrentUser + ""});

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

tasks = getListOfTasks(query);

}

return tasks;

}

/\*\*

\* Повертає список власних завдань

\*

\* @param idCurrentUser ідентифікатор користувача, для якого потрібно створити список завдань

\* @return список створених завдань

\* @throws SQLException помилка при створенні списку завдань

\* @see SQLException

\* @see Task

\* @see Cursor

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see LoaderOfTasksDao#getListOfTasks(Cursor) LoaderOfTasksDao#getListOfTasks(Cursor)

\*/

public static List<Task> getListOfСreatedTasks(int idCurrentUser) throws SQLException {

List<Task> tasks = null;

Cursor query = DB.rawQuery("SELECT tasks.\*" +

" FROM tasks " +

" JOIN users" +

" ON tasks.id\_user\_leader = users.id" +

" WHERE users.id = ?" +

" AND users.role = 'Leader';",

new String[]{idCurrentUser + ""});

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

tasks = getListOfTasks(query);

}

return tasks;

}

/\*\*

\* Створює і повертає список завдання із запиту

\*

\* @param query запит до бази дпних

\* @return список завдань

\* @throws SQLException помилка під час створення списку завдань

\* @see SQLException

\* @see Task

\* @see Cursor

\* @see Cursor#moveToNext()

\*/

@SuppressLint("Range")

private static List<Task> getListOfTasks(Cursor query) throws SQLException {

List<Task> tasks = new ArrayList<>();

while (query.moveToNext()) {

Task task = new Task();

task.setId(query.getInt(query.getColumnIndex("id")));

task.setName(query.getString(query.getColumnIndex("name")));

task.setContent(query.getString(query.getColumnIndex("content")));

task.setDone(query.getInt(query.getColumnIndex("done")) == 1);

task.setLeader(UserDao.getLeaderById(query.getInt(query.getColumnIndex("id\_user\_leader"))));

task.setExecutors(TaskDao.getListOfExecutorsForTask(task));

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

task.setDateTimeOfPublication(LocalDateTime.parse(query.getString(query.getColumnIndex("date\_of\_publication")),

DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

task.setEndDateTime(LocalDateTime.parse(query.getString(query.getColumnIndex("end\_date")),

DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

if (query.getString(query.getColumnIndex("execution\_date")) != null) {

task.setDateTimeExecution(LocalDateTime.parse(query.getString(query.getColumnIndex("execution\_date")),

DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

} else {

task.setDateTimeExecution(null);

}

}

tasks.add(task);

}

return tasks;

}

}

**SearchDao**

package com.rosokha.todolist.dao;

import android.annotation.SuppressLint;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import com.rosokha.todolist.authorization.MainActivity;

import com.rosokha.todolist.users.Leader;

import com.rosokha.todolist.users.User;

import com.rosokha.todolist.users.Worker;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Клас пошуку

\*/

public class SearchDao {

/\*\*

\* Об'єкт бази баних

\*/

private static SQLiteDatabase DB;

static {

DB = MainActivity.getDB();

}

/\*\*

\* Шукає користувачів

\*

\* @param login логін, по якому здійснюється пошук

\* @return список знайдених користувачів

\* @throws SQLException the sql exception

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see Cursor

\* @see Cursor#moveToNext() Cursor#moveToNext()

\*/

@SuppressLint("Range")

public static List<User> findUsers(String login) throws SQLException {

List<User> foundUsers = new ArrayList<>();

Cursor query = DB.rawQuery("SELECT \* " +

" FROM users " +

" WHERE login " +

" LIKE ? " +

" AND ? != '';",

new String[]{login + "%", login});

while (query.moveToNext()) {

if (query.getString(query.getColumnIndex("role")).equals("Leader")) {

Leader tempLeader = new Leader();

tempLeader.setId(query.getInt(query.getColumnIndex("id")));

tempLeader.setName(query.getString(query.getColumnIndex("name")));

tempLeader.setLogin(query.getString(query.getColumnIndex("login")));

foundUsers.add(tempLeader);

} else {

Worker tempWorker = new Worker();

tempWorker.setId(query.getInt(query.getColumnIndex("id")));

tempWorker.setName(query.getString(query.getColumnIndex("name")));

tempWorker.setLogin(query.getString(query.getColumnIndex("login")));

foundUsers.add(tempWorker);

}

}

query.close();

return foundUsers;

}

}

**TaskDao**

package com.rosokha.todolist.dao;

import android.annotation.SuppressLint;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import android.os.Build;

import androidx.annotation.RequiresApi;

import com.rosokha.todolist.authorization.MainActivity;

import com.rosokha.todolist.task.Task;

import com.rosokha.todolist.users.Leader;

import com.rosokha.todolist.users.User;

import com.rosokha.todolist.users.Worker;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

import java.util.LinkedHashSet;

import java.util.List;

import java.util.Set;

/\*\*

\* Представляє собою сутність, яка створює завдання і додає виконавців до нього

\*/

public class TaskDao {

/\*\*

\* Об'єкт бази даних

\*/

private static SQLiteDatabase DB;

static {

DB = MainActivity.getDB();

}

/\*\*

\* Створює завдання

\*

\* @param idCurrentUser ідентифікатор керівника, який створив завдання

\* @param nameTask назва завдання

\* @param contentTask зміст завдання

\* @param endDateTime дата і час закінчення завдання

\* @throws SQLException помилка при створенні завдання

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\*/

@RequiresApi(api = Build.VERSION\_CODES.O)

public static void createTask(int idCurrentUser, String nameTask, String contentTask, String endDateTime) throws SQLException {

DB.execSQL(" INSERT INTO tasks (id\_user\_leader, name, content, date\_of\_publication, end\_date, done) " +

"VALUES (?,?,?,?,?,?);",

new Object[]{idCurrentUser, nameTask, contentTask,

LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")), endDateTime, 0});

}

/\*\*

\* Повертає ідентифікатор останнього створеного завдання

\*

\* @return ідентифікатор завдання

\* @throws SQLException помилка під час отримання ідентифікатору останнього завдання

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see Cursor

\* @see Cursor#moveToFirst() Cursor#moveToFirst()

\*/

@SuppressLint("Range")

public static int getIDLastTask() throws SQLException {

Cursor query = DB.rawQuery("SELECT id" +

" FROM tasks " +

" ORDER BY id DESC LIMIT 1;",

null);

if (query.moveToFirst()) {

return query.getInt(query.getColumnIndex("id"));

} else {

throw new SQLException();

}

}

/\*\*

\* Додає виконавців до завдання

\*

\* @param idTask ідентифікатор завдання, до якого потрібно додати виконавців

\* @param executorsLogins логіни виконавців завдання

\* @throws SQLException помилка при додаванні виконавців до завдання

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\* @see TaskDao#getUserIDsByLogins(Set) TaskDao#getUserIDsByLogins(Set)

\*/

public static void addExecutors(int idTask, Set<String> executorsLogins) throws SQLException {

List<Integer> idsExecutors = getUserIDsByLogins(executorsLogins);

for (int idExecutor : idsExecutors) {

DB.execSQL("INSERT INTO executors(id\_user, id\_task)" +

" VALUES (?,?);",

new Object[]{idExecutor, idTask});

}

}

/\*\*

\* Знаходить завдання по ідентифікатору

\*

\* @param id ідентифікатор завдання

\* @return завдання task by id

\* @throws SQLException помилка під час отримання завдання по ідентифікатору

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see Cursor

\* @see Cursor#moveToFirst() Cursor#moveToFirst()

\* @see Task

\*/

@SuppressLint("Range")

public static Task getTaskByID(int id) throws SQLException {

Task task = new Task();

Cursor query = DB.rawQuery("SELECT \* " +

" FROM tasks " +

" WHERE id = ?;",

new String[]{id + ""});

if (query.moveToFirst()) {

task.setId(id);

task.setName(query.getString(query.getColumnIndex("name")));

task.setContent(query.getString(query.getColumnIndex("content")));

task.setDone(query.getInt(query.getColumnIndex("done")) == 1);

task.setLeader(UserDao.getLeaderById(query.getInt(query.getColumnIndex("id\_user\_leader"))));

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

task.setDateTimeOfPublication(LocalDateTime.parse(query.getString(query.getColumnIndex("date\_of\_publication")),

DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

task.setEndDateTime(LocalDateTime.parse(query.getString(query.getColumnIndex("end\_date")),

DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

if (query.getString(query.getColumnIndex("execution\_date")) != null) {

task.setDateTimeExecution(LocalDateTime.parse(query.getString(query.getColumnIndex("execution\_date")),

DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

} else {

task.setDateTimeExecution(null);

}

}

}

return task;

}

/\*\*

\* Оновлює дані завдання у бд

\*

\* @param task завдання яке потрібно оновити

\* @throws SQLException помилка під час оновлення завдання

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\* @see Task

\*/

public static void updateTask(Task task) throws SQLException {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

DB.execSQL("UPDATE tasks " +

" SET name = ?, content = ?, end\_date = ? " +

" WHERE id = ?",

new Object[]{task.getName(), task.getContent(),

task.getEndDateTime().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")), task.getId()});

} else {

throw new SQLException();

}

}

/\*\*

\* Видаляє завдання

\*

\* @param task завдання яке потрібно видалити

\* @throws SQLException помилка під час видалення завдання

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\* @see Task

\*/

public static void deleteTask(Task task) throws SQLException {

DB.execSQL("DELETE FROM tasks " +

" WHERE id = ?;",

new Object[]{task.getId()});

}

/\*\*

\* Створює список ідентифікаторів виконавців завдання

\*

\* @param executorsLogins логіни, по яким створюєсться список з id виконавців

\* @return список з ідентифікаторами виконавців

\* @throws SQLException помилка при отриманні id

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[])

\* @see Cursor

\* @see Cursor#moveToFirst()

\* @see Task

\*/

@SuppressLint("Range")

private static List<Integer> getUserIDsByLogins(Set<String> executorsLogins) throws SQLException {

List<Integer> ids = new ArrayList<>();

for (String login : executorsLogins) {

Cursor query = DB.rawQuery("SELECT id" +

" FROM users " +

" WHERE login = ?;",

new String[]{login});

if (query.moveToFirst()) {

ids.add(query.getInt(query.getColumnIndex("id")));

}

}

return ids;

}

/\*\*

\* Встановлює статус виконання для завдання

\*

\* @param task завдання

\* @param isDone виконане?

\* @throws SQLException помилка при зміні статусу виконання завдання

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\* @see Task

\*/

public static void setDone(Task task, boolean isDone) throws SQLException {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

if (isDone) {

DB.execSQL("UPDATE tasks " +

" SET done = ?, execution\_date = ? " +

" WHERE id = ?;",

new Object[]{1,

LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")),

task.getId()});

} else {

DB.execSQL("UPDATE tasks " +

" SET done = ?, execution\_date = ? " +

" WHERE id = ?;",

new Object[]{0, null, task.getId()});

}

} else {

throw new SQLException();

}

}

/\*\*

\* Створює список виконавців завдання

\*

\* @param task завдання

\* @return список виконавців

\* @throws SQLException промилка при створенні списку виконавців

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see Cursor

\* @see Cursor#moveToNext() Cursor#moveToNext()

\* @see Task

\* @see User

\* @see Leader

\* @see Worker

\*/

@SuppressLint("Range")

public static Set<User> getListOfExecutorsForTask(Task task) throws SQLException {

Set<User> executors = new LinkedHashSet<>();

Cursor query = DB.rawQuery("SELECT users.\* " +

" FROM users " +

" JOIN executors " +

" ON users.id = executors.id\_user " +

" " +

" JOIN tasks " +

" ON tasks.id = executors.id\_task " +

" WHERE tasks.id = ?;",

new String[]{task.getId() + ""});

while (query.moveToNext()) {

User user;

if (query.getString(query.getColumnIndex("role")).equals("Leader")) {

user = new Leader();

user.setId(query.getInt(query.getColumnIndex("id")));

user.setName(query.getString(query.getColumnIndex("name")));

user.setLogin(query.getString(query.getColumnIndex("login")));

} else {

user = new Worker();

user.setId(query.getInt(query.getColumnIndex("id")));

user.setName(query.getString(query.getColumnIndex("name")));

user.setLogin(query.getString(query.getColumnIndex("login")));

}

executors.add(user);

}

query.close();

return executors;

}

/\*\*

\* Видаляє виконавці завдання по ідентифікатору

\*

\* @param idExecutor ідентифікатор виконавця

\* @param idTask ідентифікатор завдання

\* @throws SQLException помилка під час видалення виконавці

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\*/

public static void deleteExecutorById(int idExecutor, int idTask) throws SQLException {

DB.execSQL("DELETE FROM executors " +

" WHERE id\_user = ? " +

" AND id\_task = ?;",

new Integer[]{idExecutor, idTask});

}

}

**UserDao**

package com.rosokha.todolist.dao;

import android.annotation.SuppressLint;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import com.rosokha.todolist.authorization.MainActivity;

import com.rosokha.todolist.task.Task;

import com.rosokha.todolist.users.Leader;

import com.rosokha.todolist.users.User;

import com.rosokha.todolist.users.Worker;

/\*\*

\* Користувач взаємодіючий з базою даних

\*/

public class UserDao {

/\*\*

\* Об'єкт бази даних

\*/

private static SQLiteDatabase DB;

static {

DB = MainActivity.getDB();

}

/\*\*

\* Оновлює логін користувача

\*

\* @param user користувач, якому потрібно оновити логін

\* @param newLogin новий логін

\* @throws SQLException помилка під час оновлення

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\*/

public static void updateLogin(User user, String newLogin) throws SQLException {

DB.execSQL("UPDATE users " +

" SET login = ? " +

"WHERE id = ?;",

new String[]{newLogin, user.getId() + ""});

}

/\*\*

\* Оновлює ПІБ користувача

\*

\* @param user користувач, якому потрібно оновити ПІБ

\* @param newName новий ПШБ

\* @throws SQLException помилка під час оновлення

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\*/

public static void updateName(User user, String newName) throws SQLException {

DB.execSQL("UPDATE users " +

" SET name = ? " +

"WHERE id = ?;",

new String[]{newName, user.getId() + ""});

}

/\*\*

\* Завантажує користувача по ідентифікатору

\*

\* @param id ідентифікатор користувача

\* @return користувача user by id

\* @throws SQLException помилка при отриманні користувача

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see Cursor

\* @see Cursor#moveToFirst() Cursor#moveToFirst()

\* @see User

\*/

@SuppressLint("Range")

public static User getUserById(int id) throws SQLException {

User user = null;

Cursor query = DB.rawQuery("SELECT \* " +

" FROM users " +

" WHERE id = ?;",

new String[]{id + ""});

if (query.moveToFirst()) {

if (query.getString(query.getColumnIndex("role")).equals("Leader")) {

user = new Leader();

user.setId(id);

user.setLogin(query.getString(query.getColumnIndex("login")));

user.setName(query.getString(query.getColumnIndex("name")));

} else {

user = new Worker();

user.setId(id);

user.setLogin(query.getString(query.getColumnIndex("login")));

user.setName(query.getString(query.getColumnIndex("name")));

}

} else {

throw new SQLException();

}

return user;

}

/\*\*

\* Завантажує керівника по ідентифікатору

\*

\* @param id ідентиифікатор керівника

\* @return керівника leader by id

\* @throws SQLException помилка при отриманні керівника

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see Cursor

\* @see Cursor#moveToFirst() Cursor#moveToFirst()

\* @see Leader

\*/

@SuppressLint("Range")

public static Leader getLeaderById(int id) throws SQLException {

Leader leader = null;

Cursor query = DB.rawQuery("SELECT \* " +

" FROM users " +

" WHERE id = ?" +

" AND role = 'Leader';",

new String[]{id + ""});

if (query.moveToFirst()) {

leader = new Leader();

leader.setId(id);

leader.setLogin(query.getString(query.getColumnIndex("login")));

leader.setName(query.getString(query.getColumnIndex("name")));

}

return leader;

}

/\*\*

\* Завантажує працівника по ідентифікатору

\*

\* @param id ідентиифікатор працівника

\* @return працівника worker by id

\* @throws SQLException помилка при отриманні працівника

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see Cursor

\* @see Cursor#moveToFirst() Cursor#moveToFirst()

\* @see Worker

\*/

@SuppressLint("Range")

public static Worker getWorkerById(int id) throws SQLException {

Worker worker = null;

Cursor query = DB.rawQuery("SELECT \* " +

" FROM users " +

" WHERE id = ?" +

" AND role = 'Worker';",

new String[]{id + ""});

if (query.moveToFirst()) {

worker = new Worker();

worker.setId(id);

worker.setLogin(query.getString(query.getColumnIndex("login")));

worker.setName(query.getString(query.getColumnIndex("name")));

}

return worker;

}

/\*\*

\* Завантажує користувача по логіну

\*

\* @param login логін користувача

\* @return користувача

\* @throws SQLException помилка при отриманні користувача

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#rawQuery(String, String[]) SQLiteDatabase#rawQuery(String, String[])

\* @see Cursor

\* @see Cursor#moveToFirst()

\* @see User

\*/

@SuppressLint("Range")

public static User getUserByLogin(String login) throws SQLException {

User user = null;

Cursor query = DB.rawQuery("SELECT \* " +

" FROM users " +

" WHERE login = ?;",

new String[]{login});

if (query.moveToFirst()) {

if (query.getString(query.getColumnIndex("role")).equals("Leader")) {

user = new Leader();

user.setId(query.getInt(query.getColumnIndex("id")));

user.setLogin(login);

user.setName(query.getString(query.getColumnIndex("name")));

} else {

user = new Worker();

user.setId(query.getInt(query.getColumnIndex("id")));

user.setLogin(login);

user.setName(query.getString(query.getColumnIndex("name")));

}

} else {

throw new SQLException();

}

return user;

}

/\*\*

\* Позначає підправлення повідомлення про нове завдання

\*

\* @param executor виконавець, якому повинно прийти повідомлення

\* @param task завдання, про яке ідеться в повідомленні

\* @param isSent чи воно відправлене?

\* @throws SQLException помилка при позначення повідомлення

\*/

public static void markNotificationAboutCreatingTask(User executor, Task task, boolean isSent) throws SQLException {

DB.execSQL("UPDATE executors" +

" SET about\_creating\_task = ?" +

"WHERE id\_user = ?" +

" AND id\_task = ?;",

new Object[]{isSent ? 1 : 0, executor.getId(), task.getId()});

}

/\*\*

\* Позначає підправлення повідомлення з попередженням про невелику кількість часу до кінця завдання

\*

\* @param executor виконавець, якому повинно прийти повідомлення

\* @param task завдання, про яке ідеться в повідомленні

\* @param isSent чи воно відправлене?

\* @throws SQLException помилка при позначення повідомлення

\*/

public static void markNotificationTaskCompletionWarning(User executor, Task task, boolean isSent) throws SQLException {

DB.execSQL("UPDATE executors" +

" SET task\_completion\_warning = ?" +

"WHERE id\_user = ?" +

" AND id\_task = ?;",

new Object[]{isSent ? 1 : 0, executor.getId(), task.getId()});

}

/\*\*

\* Позначає підправлення повідомлення про несвоєчасне виконання завдання

\*

\* @param executor виконавець, якому повинно прийти повідомлення

\* @param task завдання, про яке ідеться в повідомленні

\* @param isSent чи воно відправлене?

\* @throws SQLException помилка при позначення повідомлення

\*/

public static void markNotificationLateCompletionOfTask(User executor, Task task, boolean isSent) throws SQLException {

DB.execSQL("UPDATE executors" +

" SET late\_completion\_of\_task = ?" +

"WHERE id\_user = ?" +

" AND id\_task = ?;",

new Object[]{isSent ? 1 : 0, executor.getId(), task.getId()});

}

}

**ViewProfileDao**

package com.rosokha.todolist.dao;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import com.rosokha.todolist.authorization.MainActivity;

/\*\*

\* Клас перегляду профілю, який працює з базою даних

\*/

public class ViewProfileDao {

/\*\*

\* Об'єкт бази даних

\*/

private static SQLiteDatabase DB;

static {

DB = MainActivity.getDB();

}

/\*\*

\* Додає користувача до вибраних

\*

\* @param idCurrentUser ідентифікатор поточного користувача

\* @param idFavoriteUser ідентифікатор вибраного користувача

\* @throws SQLException помилка при додаванні користувача

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\*/

public static void addFavoriteUser(int idCurrentUser, int idFavoriteUser) throws SQLException {

DB.execSQL("INSERT INTO favorite\_users (id\_current\_user, id\_favorite\_user) " +

" VALUES (?, ?);",

new Integer[]{idCurrentUser, idFavoriteUser});

}

/\*\*

\* Видаляє користувача до вибраних

\*

\* @param idCurrentUser ідентифікатор поточного користувача

\* @param idFavoriteUser ідентифікатор вибраного користувача

\* @throws SQLException помилка при видаленні користувача

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\*/

public static void deleteFavoriteUser(int idCurrentUser, int idFavoriteUser) throws SQLException {

DB.execSQL("DELETE FROM favorite\_users " +

" WHERE id\_current\_user = ? " +

" AND id\_favorite\_user = ?;",

new Integer[]{idCurrentUser, idFavoriteUser});

}

/\*\*

\* Перевіряє унікальність логіну користувача у базі даних без урахування старого логіну користувача

\*

\* @param newLogin новий логін

\* @param currentLogin поточний логін

\* @return поверта true якщо такого логіне ще не має. Повертає false якщо такий логін вже існує

\* @throws SQLException помилка при видаленні користувача

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\*/

public static boolean validationLoginWithoutCurrentUser(String newLogin, String currentLogin) throws SQLException{

Cursor query = DB.rawQuery("SELECT login " +

" FROM users " +

" WHERE login = ? " +

" AND login != ?;",

new String[]{newLogin, currentLogin});

return !query.moveToFirst();

}

/\*\*

\* Перевіряє чи є аккаунт користувача, профіль якого переглядається, у вибраних поточного користувача

\*

\* @param idCurrentUser ідентифікатор поточного користувача

\* @param idFavoriteUser ідентифікатор вибраного користувача

\* @return поверта true якщо користувач є у вибраних. Повертає false якщо користувача немає у вибраних

\* @throws SQLException помилка при видаленні користувача

\* @see SQLException

\* @see SQLiteDatabase

\* @see SQLiteDatabase#execSQL(String) SQLiteDatabase#execSQL(String)

\*/

public static boolean checkUserForFavorite(int idCurrentUser, int idFavoriteUser) throws SQLException{

Cursor query = DB.rawQuery("SELECT favorite\_users.\* " +

" FROM favorite\_users " +

" WHERE favorite\_users.id\_current\_user = ? " +

" AND favorite\_users.id\_favorite\_user = ?;",

new String[]{idCurrentUser + "", idFavoriteUser+""});

return query.moveToFirst();

}

}

**Comment**

package com.rosokha.todolist.task;

import com.rosokha.todolist.users.User;

import java.io.Serializable;

import java.time.LocalDateTime;

/\*\*

\* Клас Comment представляє собою коментар до завдання

\*

\* @see Serializable

\*/

public class Comment implements Serializable {

/\*\*

\* Ідентифікатор коментаря

\*/

private int id;

/\*\*

\* Текст коментаря

\*/

private String content;

/\*\*

\* Дата написання коментаря

\*/

private LocalDateTime dateTimeOfWriting;

/\*\*

\* Користувач, який відправив коментар

\*/

private User user;

/\*\*

\* Завдання до якого написаний коментар

\*/

private Task task;

/\*\*

\* Конструктор з пустими параметрами

\*/

public Comment() {

}

/\*\*

\* Конструктор зі всіма параметрами

\*

\* @param id ідентифікатор

\* @param content зміст

\* @param dateTimeOfWriting дата написання

\* @param user користувач, який написав коментар

\* @param task завдання, до якого написаний коментар

\*/

public Comment(int id, String content, LocalDateTime dateTimeOfWriting, User user, Task task) {

this.id = id;

this.content = content;

this.dateTimeOfWriting = dateTimeOfWriting;

this.user = user;

this.task = task;

}

/\*\*

\* Getter для поля id

\*

\* @return повертає ідентифікатор коментаря

\*/

public int getId() {

return id;

}

/\*\*

\* Setter для id

\*

\* @param id ідентифікатор коментаря

\*/

public void setId(int id) {

this.id = id;

}

/\*\*

\* Getter для поля content

\*

\* @return повертає текст коментаря

\*/

public String getContent() {

return content;

}

/\*\*

\* Setter для поля content

\*

\* @param content зміст коментаря

\*/

public void setContent(String content) {

this.content = content;

}

/\*\*

\* Getter для поля dateTimeOfWriting

\*

\* @return повертає дату написання

\*/

public LocalDateTime getDateTimeOfWriting() {

return dateTimeOfWriting;

}

/\*\*

\* Setter для поля dateTimeOfWriting

\*

\* @param dateTimeOfWriting дата написання

\*/

public void setDateTimeOfWriting(LocalDateTime dateTimeOfWriting) {

this.dateTimeOfWriting = dateTimeOfWriting;

}

/\*\*

\* Getter для поля task

\*

\* @return повертає завдання, до якого написаний коментар

\* @see Task

\*/

public Task getTask() {

return task;

}

/\*\*

\* Setter для поля task

\*

\* @param task азавдання, до якого написаний коментар

\* @see Task

\*/

public void setTask(Task task) {

this.task = task;

}

/\*\*

\* Getter для поля user

\*

\* @return повертає користувача, який написав коментар

\* @see User

\*/

public User getUser() {

return user;

}

/\*\*

\* Setter для поля user

\*

\* @param user користувач, який написав коментар

\* @see User

\*/

public void setUser(User user) {

this.user = user;

}

}

**FinderExecutorsActivity**

package com.rosokha.todolist.task;

import android.annotation.SuppressLint;

import android.app.Activity;

import android.content.pm.ActivityInfo;

import android.database.SQLException;

import android.graphics.Color;

import android.os.Bundle;

import android.text.Editable;

import android.text.TextWatcher;

import android.view.View;

import android.widget.Button;

import android.widget.CheckBox;

import android.widget.EditText;

import android.widget.LinearLayout;

import android.widget.Toast;

import androidx.annotation.Nullable;

import com.rosokha.todolist.R;

import com.rosokha.todolist.dao.LoaderOfFavoriteUsersDao;

import com.rosokha.todolist.dao.SearchDao;

import com.rosokha.todolist.dao.TaskDao;

import com.rosokha.todolist.users.HomePageOfLeaderActivity;

import com.rosokha.todolist.users.Leader;

import com.rosokha.todolist.users.User;

import java.util.LinkedHashSet;

import java.util.List;

import java.util.Set;

/\*\*

\* Представляє собою графічний користувацький інтерфейс пошуку виконавців до завдання

\*

\* @see Activity

\*/

public class FinderExecutorsActivity extends Activity {

/\*\*

\* Графічний користувацький інтерфейс пошуку користувачів

\*/

private View search;

/\*\*

\* Графічний користувацький інтерфейс вибору користувачів з вибраних

\*/

private View favoritesUsers;

/\*\*

\* Контейнер який містить в собі знайдених користувачів

\*/

private LinearLayout llFoundUsers;

/\*\*

\* Контейнер який містить в собі вибраних користувачів

\*/

private LinearLayout llFavoriteUsers;

/\*\*

\* Логіни виконавців завдання

\*/

private Set<String> executorsLogins = new LinkedHashSet<>();

/\*\*

\* Завдання

\*/

private Task task = null;

/\*\*

\* Поточкий керівник, який вибирає виконавців

\*/

private Leader currentUser = HomePageOfLeaderActivity.getCurrentLeader();

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see Activity#onCreate(Bundle)

\* @see Activity#setContentView(View)

\* @see Activity#setRequestedOrientation(int)

\* @see FinderExecutorsActivity#inicialComponents()

\*/

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_find\_executors);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);//блокує поворот екрану

inicialComponents();

}

/\*\*

\* Ініціалізує компоненти

\*

\* @see Bundle

\* @see Task

\*/

private void inicialComponents() {

Bundle arguments = getIntent().getExtras();

if (arguments != null) {

task = (Task) arguments.getSerializable(Task.class.getSimpleName());

}

Button btnSaveExecutors = findViewById(R.id.btnSaveExecutors);

Button btnSearch = findViewById(R.id.btnSearch);

Button btnSelectFromFavorites = findViewById(R.id.btnSelectFromFavorites);

search = findViewById(R.id.activity\_search);

EditText etSearch = search.findViewById(R.id.etSearch);

llFoundUsers = search.findViewById(R.id.llFoundUsers);

Button btnGoBackFromSearch = search.findViewById(R.id.btnGoBack);

favoritesUsers = findViewById(R.id.activity\_favorite\_users);

llFavoriteUsers = favoritesUsers.findViewById(R.id.llFavoriteUsers);

Button btnGoBackFromFavoritesUsers = favoritesUsers.findViewById(R.id.btnGoBack);

favoritesUsers.findViewById(R.id.btnFindNewUsers).setVisibility(View.GONE);

btnGoBackFromFavoritesUsers.setText("OK");

btnGoBackFromSearch.setText("OK");

btnSaveExecutors.setOnClickListener((View v) -> {

if (task != null) {

TaskDao.addExecutors(task.getId(), executorsLogins);

}

finish();

});

btnSearch.setOnClickListener((View v) -> search.setVisibility(View.VISIBLE));

btnSelectFromFavorites.setOnClickListener((View v) -> {

favoritesUsers.setVisibility(View.VISIBLE);

favoritesUsers.findViewById(R.id.btnFindNewUsers).setVisibility(View.GONE);

showFavoriteUsers();

});

btnGoBackFromSearch.setOnClickListener((View v) -> search.setVisibility(View.GONE));

btnGoBackFromFavoritesUsers.setOnClickListener((View v) -> favoritesUsers.setVisibility(View.GONE));

etSearch.addTextChangedListener(new TextWatcher() {

@Override

public void beforeTextChanged(CharSequence s, int start, int count, int after) {

}

@Override

public void onTextChanged(CharSequence s, int start, int before, int count) {

}

@Override

public void afterTextChanged(Editable s) {

findUsers(etSearch.getText().toString());

}

});

}

/\*\*

\* Шукає користувачів

\*

\* @param login логін по якому потрібно здійснити пошук

\* @see User

\* @see SearchDao

\* @see SearchDao#findUsers(String)

\* @see Toast

\* @see FinderExecutorsActivity#addOrRemoveExecutor(CheckBox)

\*/

@SuppressLint("Range")

private void findUsers(String login) {

llFoundUsers.removeAllViews();

List<User> foundUsers = null;

try {

foundUsers = SearchDao.findUsers(login);

} catch (SQLException ex) {

Toast.makeText(this, "Пошук не працює", Toast.LENGTH\_LONG).show();

return;

}

User foundCurrentUser = null;

for (User tempUser : foundUsers) {

if (tempUser.getId() == currentUser.getId()) {

foundCurrentUser = tempUser;

}

}

foundUsers.remove(foundCurrentUser);

if (task != null) {

for (User tempUser1 : task.getExecutors()) {

for (User tempUser2 : foundUsers) {

if (tempUser1.getLogin().equals(tempUser2.getLogin())) {

foundUsers.remove(tempUser2);

break;

}

}

}

}

for (User tempUser : foundUsers) {

CheckBox chkExecutor = new CheckBox(this);

chkExecutor.setText(tempUser.getLogin());

chkExecutor.setTextColor(Color.WHITE);

chkExecutor.setTextSize(20);

chkExecutor.setOnClickListener((View v) -> addOrRemoveExecutor(chkExecutor));

for (String checkExecutor : executorsLogins) {

if (checkExecutor.equals(tempUser.getLogin())) {

chkExecutor.setChecked(true);

break;

} else {

chkExecutor.setChecked(false);

}

}

llFoundUsers.addView(chkExecutor);

}

}

/\*\*

\* Додає або видаляє користувача з списку виконавців

\*

\* @param chkExecutor checkBox виконавця

\* @see CheckBox

\*/

private void addOrRemoveExecutor(CheckBox chkExecutor) {

if (chkExecutor.isChecked()) {

executorsLogins.add(chkExecutor.getText().toString());

} else {

for (String executorLogin : executorsLogins) {

if (executorLogin.equals(chkExecutor.getText())) {

executorsLogins.remove(executorLogin);

break;

}

}

}

LinearLayout llViewExecutors = findViewById(R.id.llViewExecutors);

llViewExecutors.removeAllViews();

for (String login : executorsLogins) {

CheckBox executorLogin = new CheckBox(this);

executorLogin.setText(login);

executorLogin.setTextSize(20);

executorLogin.setTextColor(Color.WHITE);

executorLogin.setChecked(true);

executorLogin.setOnClickListener((View v) -> addOrRemoveExecutor(executorLogin));

llViewExecutors.addView(executorLogin);

}

}

/\*\*

\* Показує вибраних користувачів

\*

\* @see User

\* @see Toast

\* @see FinderExecutorsActivity#addOrRemoveExecutor(CheckBox)

\*/

@SuppressLint("Range")

private void showFavoriteUsers() {

List<User> favoriteUsers = null;

llFavoriteUsers.removeAllViews();

try {

favoriteUsers = LoaderOfFavoriteUsersDao.loadFavoriteUsers(currentUser.getId());

} catch (SQLException ex) {

Toast.makeText(this, "Неможливо завантажити вибраних користувачів", Toast.LENGTH\_LONG).show();

return;

}

if (task != null) {

for (User tempUser1 : task.getExecutors()) {

for (User tempUser2 : favoriteUsers) {

if (tempUser1.getLogin().equals(tempUser2.getLogin())) {

favoriteUsers.remove(tempUser2);

break;

}

}

}

}

for (User favoriteUser : favoriteUsers) {

CheckBox chkExecutor = new CheckBox(this);

chkExecutor.setText(favoriteUser.getLogin());

chkExecutor.setTextColor(Color.WHITE);

chkExecutor.setTextSize(20);

chkExecutor.setOnClickListener((View v) -> addOrRemoveExecutor(chkExecutor));

for (String checkExecutor : executorsLogins) {

if (checkExecutor.equals(favoriteUser.getLogin())) {

chkExecutor.setChecked(true);

break;

} else {

chkExecutor.setChecked(false);

}

}

llFavoriteUsers.addView(chkExecutor);

}

}

}

**Task**

package com.rosokha.todolist.task;

import android.graphics.Color;

import android.os.Build;

import androidx.annotation.DrawableRes;

import com.rosokha.todolist.R;

import com.rosokha.todolist.users.Leader;

import com.rosokha.todolist.users.User;

import java.io.Serializable;

import java.time.LocalDateTime;

import java.time.chrono.ChronoLocalDateTime;

import java.util.List;

import java.util.Set;

/\*\*

\* Являє собою обє'кт завдання

\*

\* @see Serializable

\*/

public class Task implements Serializable {

/\*\*

\* Ідентифікатор завдання

\*/

private int id;

/\*\*

\* Назва завдання

\*/

private String name;

/\*\*

\* Зміст завдання

\*/

private String content;

/\*\*

\* Дата і час публікації

\*/

private LocalDateTime dateTimeOfPublication;

/\*\*

\* Дата і час, до якого потрібно виконати завдання

\*/

private LocalDateTime endDateTime;

/\*\*

\* Дата і час, в який виконано завдання

\*/

private LocalDateTime dateTimeExecution;

/\*\*

\* Керівник завдання

\*/

private Leader leader;

/\*\*

\* Чи є завдання виконаним?

\*/

private boolean isDone;

/\*\*

\* Колекція з коментарями до завдання

\*/

private List<Comment> comments = null;

/\*\*

\* Список виконавців

\*/

private Set<User> executors = null;

/\*\*

\* Повідомлення про створення завдання

\*/

private boolean notificationAboutCreatingTask = true;

/\*\*

\* Повідомлення з попередженням

\*/

private boolean notificationTaskCompletionWarning = true;

/\*\*

\* Повідомлення про закінчення завдання

\*/

private boolean notificationLateCompletionOfTask = true;

/\*\*

\* Конструктор

\*

\* @param name назва завдання

\* @param content зміст завдання

\* @param date дата, до якої потрібно виконати завдання

\*/

public Task(String name, String content, LocalDateTime date) {

this.name = name;

this.content = content;

this.endDateTime = date;

}

/\*\*

\* Конструктор

\*/

public Task() {

}

/\*\*

\* Повертає колір завдання, в залежності від дати закінчення і виконання

\*

\* @param tabPosition позиція сторінки яку переглядає користувач

\* @return колір завдання

\* @see Color

\* @see LocalDateTime#isAfter(ChronoLocalDateTime) LocalDateTime#isAfter(ChronoLocalDateTime)

\* @see LocalDateTime#isBefore(ChronoLocalDateTime) LocalDateTime#isBefore(ChronoLocalDateTime)

\*/

public int getColorTask(@DrawableRes int tabPosition) {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

if (tabPosition == 2) {

if (!isDone) {

if (LocalDateTime.now().isAfter(endDateTime)) {

return R.drawable.round\_red\_task;

}

if (LocalDateTime.now().isBefore(endDateTime)) {

return R.drawable.round\_green\_task;

}

}

if (isDone) {

if (dateTimeExecution.isAfter(endDateTime)) {

return R.drawable.round\_red\_task;

}

if (dateTimeExecution.isBefore(endDateTime)) {

return R.drawable.round\_green\_task;

}

}

}

if (!isDone) {

if (LocalDateTime.now().isAfter(endDateTime)) {

return R.drawable.round\_red\_task;

}

if (endDateTime.minusDays(1).isBefore(LocalDateTime.now())) {

return R.drawable.round\_yellow\_tasl;

}

if (LocalDateTime.now().isBefore(endDateTime)) {

return R.drawable.round\_green\_task;

}

}

if (isDone) {

if (dateTimeExecution.isAfter(endDateTime)) {

return R.drawable.round\_red\_task;

}

if (dateTimeExecution.isBefore(endDateTime)) {

return R.drawable.round\_green\_task;

}

}

}

return 0;

}

/\*\*

\* Getter для поля leader

\*

\* @return керівника, який створив завдання

\* @see Leader

\*/

public Leader getLeader() {

return leader;

}

/\*\*

\* Setter для поля leader

\*

\* @param leader керівник, який створив завдання

\* @see Leader

\*/

public void setLeader(Leader leader) {

this.leader = leader;

}

/\*\*

\* Getter для поля dateTimeExecution

\*

\* @return дату виконання

\*/

public LocalDateTime getDateTimeExecution() {

return dateTimeExecution;

}

/\*\*

\* Setter для поля dateTimeExecution

\*

\* @param dateTimeExecution дата і час виконання

\*/

public void setDateTimeExecution(LocalDateTime dateTimeExecution) {

this.dateTimeExecution = dateTimeExecution;

}

/\*\*

\* Getter для поля isDone

\*

\* @return чи є завдання виконаним?

\*/

public boolean getDone() {

return isDone;

}

/\*\*

\* Setter для поля isDone

\*

\* @param isDone чи є завдання виконаним?

\*/

public void setDone(boolean isDone) {

this.isDone = isDone;

}

/\*\*

\* Getter для поля comments

\*

\* @return список коментарів

\* @see Comment

\*/

public List<Comment> getComments() {

return comments;

}

/\*\*

\* Setter для поля comments

\*

\* @param comments список коментарів

\* @see Comment

\*/

public void setComments(List<Comment> comments) {

this.comments = comments;

}

/\*\*

\* Getter для поля id

\*

\* @return ідентифікатор завдання

\*/

public int getId() {

return id;

}

/\*\*

\* Setter для поля id

\*

\* @param id ідентифікатор завдання

\*/

public void setId(int id) {

this.id = id;

}

/\*\*

\* Getter для поля name

\*

\* @return назву завдання

\*/

public String getName() {

return name;

}

/\*\*

\* Setter для поля name

\*

\* @param name назва завдання

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* Getter для поля content

\*

\* @return зміст завдання

\*/

public String getContent() {

return content;

}

/\*\*

\* Setter для поля content

\*

\* @param content зміст завдання

\*/

public void setContent(String content) {

this.content = content;

}

/\*\*

\* Getter для поля endDateTime

\*

\* @return дату закінчення виконання завдання

\*/

public LocalDateTime getEndDateTime() {

return endDateTime;

}

/\*\*

\* Setter для поля endDateTime

\*

\* @param endDateTime дата закінчення виконання завдання

\*/

public void setEndDateTime(LocalDateTime endDateTime) {

this.endDateTime = endDateTime;

}

/\*\*

\* Getter для поля dateTimeOfPublication

\*

\* @return дату публікації завдання

\*/

public LocalDateTime getDateTimeOfPublication() {

return dateTimeOfPublication;

}

/\*\*

\* Setter для поля dateTimeOfPublication

\*

\* @param dateTimeOfPublication дата публікації завдання

\*/

public void setDateTimeOfPublication(LocalDateTime dateTimeOfPublication) {

this.dateTimeOfPublication = dateTimeOfPublication;

}

/\*\*

\* Getter для поля executors

\*

\* @return список виконавців завдання

\* @see User

\*/

public Set<User> getExecutors() {

return executors;

}

/\*\*

\* Setter для поля executors

\*

\* @param executors список виконавців завдання

\* @see User

\*/

public void setExecutors(Set<User> executors) {

this.executors = executors;

}

/\*\*

\* Getter для поля notificationAboutCreatingTask

\*

\* @return чи відправлене повідомлення про створення завдання

\*/

public boolean isNotificationAboutCreatingTask() {

return notificationAboutCreatingTask;

}

/\*\*

\* Setter для поля notificationAboutCreatingTask

\*

\* @param notificationAboutCreatingTask повідомлення про створення завдання

\*/

public void setNotificationAboutCreatingTask(boolean notificationAboutCreatingTask) {

this.notificationAboutCreatingTask = notificationAboutCreatingTask;

}

/\*\*

\* Getter для поля notificationTaskCompletionWarning

\*

\* @return чи відправлене повідомлення з попередженням

\*/

public boolean isNotificationTaskCompletionWarning() {

return notificationTaskCompletionWarning;

}

/\*\*

\* Setter для поля notificationTaskCompletionWarning

\*

\* @param notificationTaskCompletionWarning повідомлення з попередженням

\*/

public void setNotificationTaskCompletionWarning(boolean notificationTaskCompletionWarning) {

this.notificationTaskCompletionWarning = notificationTaskCompletionWarning;

}

/\*\*

\* Getter для поля notificationLateCompletionOfTask

\*

\* @return чи відправлене повідомлення про не виконання завдання

\*/

public boolean isNotificationLateCompletionOfTask() {

return notificationLateCompletionOfTask;

}

/\*\*

\* Setter для поля notificationLateCompletionOfTask

\*

\* @param notificationLateCompletionOfTask повідомлення про не виконання завдання

\*/

public void setNotificationLateCompletionOfTask(boolean notificationLateCompletionOfTask) {

this.notificationLateCompletionOfTask = notificationLateCompletionOfTask;

}

}

**TaskCreatorActivity**

package com.rosokha.todolist.task;

import android.annotation.SuppressLint;

import android.app.Activity;

import android.content.pm.ActivityInfo;

import android.database.SQLException;

import android.graphics.Color;

import android.os.Build;

import android.os.Bundle;

import android.text.Editable;

import android.text.TextWatcher;

import android.view.View;

import android.widget.Button;

import android.widget.CheckBox;

import android.widget.DatePicker;

import android.widget.EditText;

import android.widget.LinearLayout;

import android.widget.ScrollView;

import android.widget.TextView;

import android.widget.TimePicker;

import android.widget.Toast;

import androidx.annotation.Nullable;

import androidx.constraintlayout.widget.ConstraintLayout;

import com.rosokha.todolist.R;

import com.rosokha.todolist.dao.LoaderOfFavoriteUsersDao;

import com.rosokha.todolist.dao.SearchDao;

import com.rosokha.todolist.dao.TaskDao;

import com.rosokha.todolist.users.HomePageOfLeaderActivity;

import com.rosokha.todolist.users.HomePageOfWorkerActivity;

import com.rosokha.todolist.users.User;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.LinkedHashSet;

import java.util.List;

import java.util.Set;

/\*\*

\* Являє собою графічний користувацький інтерфейс створення завдання

\*

\* @see Activity

\*/

public class TaskCreatorActivity extends Activity {

/\*\*

\* Користувач, який вибиратиме виконавців

\*/

private User currentUser;

/\*\*

\* Заголовок

\*/

private TextView tvTitle;

/\*\*

\* Назва завдання

\*/

private EditText etTaskName;

/\*\*

\* Зміст завдання

\*/

private EditText etTaskContent;

/\*\*

\* Дата закінчення завдання

\*/

private DatePicker dpDate;

/\*\*

\* Час закінчення завдання

\*/

private TimePicker tpTime;

/\*\*

\* Кнопка, яка відкриває введення дати і часу

\*/

private Button btnInputDateAndTime;

/\*\*

\* Кнопк,а яка створює завданя

\*/

private Button btnCreateTask;

/\*\*

\* Кнопка, яка здійснює вихід з вікна

\*/

private Button btnGoBack;

/\*\*

\* Контейней у якому розміщені введені дані

\*/

private ScrollView sclInputData;

/\*\*

\* Сторінка пошуку

\*/

private View search;

/\*\*

\* Сторінка вибраних користувачів

\*/

private View favoritesUsers;

/\*\*

\* Логіни виконавців

\*/

private Set<String> executorsLogins = new LinkedHashSet<>();

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see Nullable

\* @see Activity#onCreate(Bundle)

\* @see Activity#setContentView(View)

\* @see Activity#setRequestedOrientation(int)

\*/

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_create\_task);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);//блокує поворот екрану

inicialComponents();

}

/\*\*

\* Ініціалізує компоненти

\*/

private void inicialComponents() {

tvTitle = findViewById(R.id.tvTitle);

etTaskName = findViewById(R.id.etTaskName);

etTaskContent = findViewById(R.id.etTaskContent);

btnInputDateAndTime = findViewById(R.id.btnInputDateAndTime);

Button btnSelectExecutors = findViewById(R.id.btnSelectExecutors);

btnCreateTask = findViewById(R.id.btnCreateTask);

btnGoBack = findViewById(R.id.btnGoBack);

sclInputData = findViewById(R.id.sclInputData);

ConstraintLayout clInputDateAndTime = findViewById(R.id.activity\_input\_date\_and\_time);

dpDate = clInputDateAndTime.findViewById(R.id.dpDate);

tpTime = clInputDateAndTime.findViewById(R.id.tpTime);

tpTime.setIs24HourView(true);

Button btnSaveDate = clInputDateAndTime.findViewById(R.id.btnSaveDate);

Button btnSaveTime = clInputDateAndTime.findViewById(R.id.btnSaveTime);

ConstraintLayout clDate = clInputDateAndTime.findViewById(R.id.clDate);

ConstraintLayout clTime = clInputDateAndTime.findViewById(R.id.clTime);

Button btnSaveExecutors = findViewById(R.id.btnSaveExecutors);

Button btnSearch = findViewById(R.id.btnSearch);

Button btnSelectFromFavorites = findViewById(R.id.btnSelectFromFavorites);

ConstraintLayout clSelectExecutors = findViewById(R.id.clSelectExecutors);

search = findViewById(R.id.activity\_search);

EditText etSearch = search.findViewById(R.id.etSearch);

Button btnGoBackFromSearch = search.findViewById(R.id.btnGoBack);

favoritesUsers = findViewById(R.id.activity\_favorite\_users);

Button btnGoBackFromFavoritesUsers = favoritesUsers.findViewById(R.id.btnGoBack);

btnGoBackFromFavoritesUsers.setText("OK");

btnGoBackFromSearch.setText("OK");

if (HomePageOfLeaderActivity.getCurrentLeader() != null) {

currentUser = HomePageOfLeaderActivity.getCurrentLeader();

} else {

currentUser = HomePageOfWorkerActivity.getCurrentWorker();

}

btnInputDateAndTime.setOnClickListener((View v) -> {

hideMainViews();

clInputDateAndTime.setVisibility(View.VISIBLE);

clDate.setVisibility(View.VISIBLE);

clTime.setVisibility(View.GONE);

btnInputDateAndTime.setText("Змінити дату");

});

btnSelectExecutors.setOnClickListener((View v) -> {

hideMainViews();

clSelectExecutors.setVisibility(View.VISIBLE);

});

btnCreateTask.setOnClickListener((View v) -> createTask());

btnSaveDate.setOnClickListener((View v) -> {

clDate.setVisibility(View.GONE);

clTime.setVisibility(View.VISIBLE);

});

btnSaveTime.setOnClickListener((View v) -> {

clInputDateAndTime.setVisibility(View.GONE);

showMainViews();

});

btnSaveExecutors.setOnClickListener((View v) -> {

clSelectExecutors.setVisibility(View.GONE);

showMainViews();

});

btnSearch.setOnClickListener((View v) -> search.setVisibility(View.VISIBLE));

btnSelectFromFavorites.setOnClickListener((View v) -> {

favoritesUsers.setVisibility(View.VISIBLE);

favoritesUsers.findViewById(R.id.btnFindNewUsers).setVisibility(View.GONE);

showFavoriteUsers();

});

btnGoBackFromSearch.setOnClickListener((View v) -> search.setVisibility(View.GONE));

btnGoBackFromFavoritesUsers.setOnClickListener((View v) -> favoritesUsers.setVisibility(View.GONE));

btnGoBack.setOnClickListener((View v) -> finish());

etSearch.addTextChangedListener(new TextWatcher() {

@Override

public void beforeTextChanged(CharSequence s, int start, int count, int after) {

}

@Override

public void onTextChanged(CharSequence s, int start, int before, int count) {

}

@Override

public void afterTextChanged(Editable s) {

findUsers(etSearch.getText().toString());

}

});

}

/\*\*

\* Шукає користувачів

\*

\* @param login логін по якому потрібно здійснити пошук

\* @see User

\* @see TaskCreatorActivity#addOrRemoveExecutor(CheckBox)

\*/

@SuppressLint("Range")

private void findUsers(String login) {

LinearLayout llFoundUsers = search.findViewById(R.id.llFoundUsers);

llFoundUsers.removeAllViews();

List<User> foundUsers = null;

try {

foundUsers = SearchDao.findUsers(login);

} catch (SQLException ex) {

Toast.makeText(this, "Пошук не працює", Toast.LENGTH\_LONG).show();

}

User foundCurrentUser = null;

for (User tempUser : foundUsers) {

if (tempUser.getId() == currentUser.getId()) {

foundCurrentUser = tempUser;

}

}

foundUsers.remove(foundCurrentUser);

for (User tempUser : foundUsers) {

CheckBox chkExecutor = new CheckBox(this);

chkExecutor.setText(tempUser.getLogin());

chkExecutor.setTextColor(Color.WHITE);

chkExecutor.setTextSize(20);

chkExecutor.setOnClickListener((View v) -> addOrRemoveExecutor(chkExecutor));

for (String checkExecutor : executorsLogins) {

if (checkExecutor.equals(tempUser.getLogin())) {

chkExecutor.setChecked(true);

break;

} else {

chkExecutor.setChecked(false);

}

}

llFoundUsers.addView(chkExecutor);

}

}

/\*\*

\* Додає або видаляє користувача до списку виконавців

\*

\* @param chkExecutor checkBox виконавця

\*/

private void addOrRemoveExecutor(CheckBox chkExecutor) {

if (chkExecutor.isChecked()) {

executorsLogins.add(chkExecutor.getText().toString());

} else {

for (String executorLogin : executorsLogins) {

if (executorLogin.equals(chkExecutor.getText())) {

executorsLogins.remove(executorLogin);

break;

}

}

}

LinearLayout llViewExecutors = findViewById(R.id.llViewExecutors);

llViewExecutors.removeAllViews();

for (String login : executorsLogins) {

CheckBox executorLogin = new CheckBox(this);

executorLogin.setText(login);

executorLogin.setTextSize(20);

executorLogin.setTextColor(Color.WHITE);

executorLogin.setChecked(true);

executorLogin.setOnClickListener((View v) -> addOrRemoveExecutor(executorLogin));

llViewExecutors.addView(executorLogin);

}

}

/\*\*

\* Показує вибраних користувачів

\*

\* @see User

\* @see TaskCreatorActivity#addOrRemoveExecutor(CheckBox)

\*/

@SuppressLint("Range")

private void showFavoriteUsers() {

List<User> favoriteUsers = null;

LinearLayout llFavoriteUsers = favoritesUsers.findViewById(R.id.llFavoriteUsers);

llFavoriteUsers.removeAllViews();

try {

favoriteUsers = LoaderOfFavoriteUsersDao.loadFavoriteUsers(currentUser.getId());

} catch (SQLException ex) {

Toast.makeText(this, "Неможливо завантажити вибраних користувачів", Toast.LENGTH\_LONG).show();

}

for (User favoriteUser : favoriteUsers) {

CheckBox chkExecutor = new CheckBox(this);

chkExecutor.setText(favoriteUser.getLogin());

chkExecutor.setTextColor(Color.WHITE);

chkExecutor.setTextSize(20);

chkExecutor.setOnClickListener((View v) -> addOrRemoveExecutor(chkExecutor));

for (String checkExecutor : executorsLogins) {

if (checkExecutor.equals(favoriteUser.getLogin())) {

chkExecutor.setChecked(true);

break;

} else {

chkExecutor.setChecked(false);

}

}

llFavoriteUsers.addView(chkExecutor);

}

}

/\*\*

\* Створює завдання

\*

\* @see TaskCreatorActivity#checkData()

\* @see TaskDao

\* @see TaskDao#createTask(int, String, String, String)

\* @see Toast

\* @see Activity#finish()

\*/

private void createTask() {

if (!checkData()) {

return;

}

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.M) {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

LocalDateTime endDateTime = LocalDateTime.of(dpDate.getYear(), dpDate.getMonth() + 1, dpDate.getDayOfMonth(), tpTime.getHour(), tpTime.getMinute());

try {

TaskDao.createTask(currentUser.getId(), etTaskName.getText().toString(),

etTaskContent.getText().toString(), endDateTime.format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

} catch (SQLException ex) {

Toast.makeText(this, "Не вдалося створити завдання", Toast.LENGTH\_LONG).show();

return;

}

}

}

try {

TaskDao.addExecutors(TaskDao.getIDLastTask(), executorsLogins);

} catch (SQLException ex) {

Toast.makeText(this, "Не вдалося додати виконавців до завдання", Toast.LENGTH\_LONG).show();

return;

}

Toast.makeText(this, "Завдання успішно створено", Toast.LENGTH\_LONG).show();

finish();

}

/\*\*

\* Перевіряє на коректність введених даних

\*

\* @return повертає true якщо дані введені коректно. Плвертає false якщо дані введені не коректно

\* @see Toast

\*/

private boolean checkData() {

if (etTaskName.getText().toString().equals("")) {

Toast.makeText(this, "Назва завдання не може бути порожньою", Toast.LENGTH\_LONG).show();

return false;

}

if (btnInputDateAndTime.getText().toString().equals("Ввести дату")) {

Toast.makeText(this, "Введіть дату і час", Toast.LENGTH\_LONG).show();

return false;

}

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.M) {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

if (LocalDateTime.of(dpDate.getYear(), dpDate.getMonth() + 1, dpDate.getDayOfMonth(), tpTime.getHour(), tpTime.getMinute()).isBefore(LocalDateTime.now())) {

Toast.makeText(this, "Дата виконання не може бути раніше теперішньої", Toast.LENGTH\_LONG).show();

return false;

}

}

}

if (executorsLogins.isEmpty()) {

Toast.makeText(this, "Виберіть виконавців", Toast.LENGTH\_LONG).show();

return false;

}

return true;

}

/\*\*

\* Сховує головні елементи activity при появі вікна вводу

\*/

private void hideMainViews() {

tvTitle.setVisibility(View.GONE);

sclInputData.setVisibility(View.GONE);

btnCreateTask.setVisibility(View.GONE);

btnGoBack.setVisibility(View.GONE);

}

/\*\*

\* Показує головні елементи activity при закритті вікна вводу

\*/

private void showMainViews() {

tvTitle.setVisibility(View.VISIBLE);

sclInputData.setVisibility(View.VISIBLE);

btnCreateTask.setVisibility(View.VISIBLE);

btnGoBack.setVisibility(View.VISIBLE);

}

}

**TasksViewerActivity**

package com.rosokha.todolist.task;

import android.app.Activity;

import android.content.Intent;

import android.content.pm.ActivityInfo;

import android.database.SQLException;

import android.graphics.Color;

import android.os.Build;

import android.os.Bundle;

import android.util.TypedValue;

import android.view.Gravity;

import android.view.View;

import android.widget.AdapterView;

import android.widget.ArrayAdapter;

import android.widget.Button;

import android.widget.CheckBox;

import android.widget.CompoundButton;

import android.widget.LinearLayout;

import android.widget.Spinner;

import android.widget.TextView;

import android.widget.Toast;

import androidx.annotation.Nullable;

import androidx.constraintlayout.widget.ConstraintLayout;

import com.google.android.material.tabs.TabLayout;

import com.rosokha.todolist.R;

import com.rosokha.todolist.dao.LoaderOfTasksDao;

import com.rosokha.todolist.dao.TaskDao;

import com.rosokha.todolist.users.Leader;

import com.rosokha.todolist.users.User;

/\*\*

\* Представляє собою графічний користувацький інтерфейс перегляду завдань

\*

\* @see Activity

\*/

public class TasksViewerActivity extends Activity {

/\*\*

\* Перемикач між типами завдань

\*/

private TabLayout tabLayout;

/\*\*

\* Користувач, який переглядає завдання

\*/

private User user;

/\*\*

\* Активні завдання

\*/

private Task[] activeTasks;

/\*\*

\* Виконані завдання

\*/

private Task[] completedTasks;

/\*\*

\* Завдання, які створив користувач

\*/

private Task[] createdTasks;

/\*\*

\* Тип, по якому потрібно філльтрувати завдання

\*/

private String filterType = "Даті публікації (зростання)";

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see Nullable

\* @see Activity#onCreate(Bundle)

\* @see Activity#setContentView(View)

\* @see Activity#setRequestedOrientation(int)

\* @see TasksViewerActivity#inicialComponents()

\*/

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_tasks);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);//блокує поворот екрану

inicialComponents();

}

/\*\*

\* Викликається при поверненні користувача в дане activity

\*

\* @see Activity#onResume()

\* @see TasksViewerActivity#setTabLayout(int)

\*/

@Override

protected void onResume() {

super.onResume();

setTabLayout(tabLayout.getSelectedTabPosition());

}

/\*\*

\* Ініціалізує компоненти

\*

\* @see Bundle

\* @see Spinner

\* @see TasksViewerActivity#setTabLayout(int)

\*/

private void inicialComponents() {

Bundle arguments = getIntent().getExtras();

if (arguments != null) {

user = (User) arguments.getSerializable(User.class.getSimpleName());

} else {

Toast.makeText(this, "Сталася помилка при передачі аргументів", Toast.LENGTH\_LONG).show();

finish();

}

tabLayout = findViewById(R.id.tabLayout);

Spinner spnFilter = findViewById(R.id.spnFilter);

String[] dataForFilter = new String[]{"Даті публікації (зростання)", "Даті публікації (спадання)",

"Даті закінчення (зростання)", "Даті закінчення (спадання)",

"Даті виконання (зростання)", "Даті виконання (спадання)"};

ArrayAdapter<String> adapter = new ArrayAdapter(this, android.R.layout.simple\_spinner\_item, dataForFilter);

adapter.setDropDownViewResource(android.R.layout.simple\_spinner\_dropdown\_item);

spnFilter.setAdapter(adapter);

spnFilter.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

@Override

public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {

filterType = (String) parent.getItemAtPosition(position);

setTabLayout(tabLayout.getSelectedTabPosition());

}

@Override

public void onNothingSelected(AdapterView<?> parent) {

}

});

Button btnGoBack = findViewById(R.id.btnGoBack);

Button btnCreateNewTask = findViewById(R.id.btnCreateNewTask);

btnGoBack.setOnClickListener((View v) -> finish());

btnCreateNewTask.setOnClickListener((View v) -> {

Intent intent = new Intent(TasksViewerActivity.this, TaskCreatorActivity.class);

startActivity(intent);

});

tabLayout.addTab(tabLayout.newTab().setText("Активні"));

tabLayout.addTab(tabLayout.newTab().setText("Виконані"));

if (user instanceof Leader) {

tabLayout.addTab(tabLayout.newTab().setText("Створені"));

}

tabLayout.setBackgroundColor(Color.rgb(0, 0, 55));

tabLayout.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {

@Override

public void onTabSelected(TabLayout.Tab tab) {

setTabLayout(tab.getPosition());

}

@Override

public void onTabUnselected(TabLayout.Tab tab) {

}

@Override

public void onTabReselected(TabLayout.Tab tab) {

}

});

}

/\*\*

\* Встановлює activity перегляду завдань залежно від позиції перемикача

\*

\* @param position позиція перемикача

\*/

private void setTabLayout(int position) {

ConstraintLayout clActiveTasks = findViewById(R.id.clActiveTasks);

ConstraintLayout clСompletedTasks = findViewById(R.id.clСompletedTasks);

ConstraintLayout clСreatedTasks = findViewById(R.id.clСreatedTasks);

switch (position) {

case 0: {

clActiveTasks.setVisibility(View.VISIBLE);

clСompletedTasks.setVisibility(View.GONE);

clСreatedTasks.setVisibility(View.GONE);

showActiveTasks();

break;

}

case 1: {

clActiveTasks.setVisibility(View.GONE);

clСompletedTasks.setVisibility(View.VISIBLE);

clСreatedTasks.setVisibility(View.GONE);

showСompletedTasks();

break;

}

case 2: {

clActiveTasks.setVisibility(View.GONE);

clСompletedTasks.setVisibility(View.GONE);

clСreatedTasks.setVisibility(View.VISIBLE);

showСreatedTasks();

break;

}

}

}

/\*\*

\* Показує активні завдання

\*

\* @see LoaderOfTasksDao

\* @see LoaderOfTasksDao#getListOfActiveTasks(int)

\* @see TasksViewerActivity#createDesignForTask(Task)

\*/

private void showActiveTasks() {

try {

if (android.os.Build.VERSION.SDK\_INT >= android.os.Build.VERSION\_CODES.O) {

activeTasks = LoaderOfTasksDao.getListOfActiveTasks(user.getId()).toArray(new Task[0]);

filterTasks(activeTasks);

} else {

throw new SQLException();

}

LinearLayout llActiveTasks = findViewById(R.id.llActiveTasks);

llActiveTasks.removeAllViews();

for (Task task : activeTasks) {

llActiveTasks.addView(createDesignForTask(task));

}

} catch (SQLException ex) {

Toast.makeText(this, "Не вдалося завантажити завдання", Toast.LENGTH\_LONG).show();

finish();

}

}

/\*\*

\* Показує виконані завдання

\*

\* @see LoaderOfTasksDao

\* @see LoaderOfTasksDao#getListOfСompletedTasks(int)

\* @see TasksViewerActivity#createDesignForTask(Task)

\*/

private void showСompletedTasks() {

try {

if (android.os.Build.VERSION.SDK\_INT >= android.os.Build.VERSION\_CODES.O) {

completedTasks = LoaderOfTasksDao.getListOfСompletedTasks(user.getId()).toArray(new Task[0]);

filterTasks(completedTasks);

} else {

throw new SQLException();

}

LinearLayout llСompletedTasks = findViewById(R.id.llСompletedTasks);

llСompletedTasks.removeAllViews();

for (Task task : completedTasks) {

llСompletedTasks.addView(createDesignForTask(task));

}

} catch (SQLException ex) {

Toast.makeText(this, "Не вдалося завантажити завдання", Toast.LENGTH\_LONG).show();

finish();

}

}

/\*\*

\* Показує створені користувачем завдання

\*

\* @see LoaderOfTasksDao

\* @see LoaderOfTasksDao#getListOfСreatedTasks(int)

\* @see TasksViewerActivity#createDesignForTask(Task)

\*/

private void showСreatedTasks() {

try {

if (android.os.Build.VERSION.SDK\_INT >= android.os.Build.VERSION\_CODES.O) {

createdTasks = LoaderOfTasksDao.getListOfСreatedTasks(user.getId()).toArray(new Task[0]);

filterTasks(createdTasks);

} else {

throw new SQLException();

}

LinearLayout llСreatedTasks = findViewById(R.id.llСreatedTasks);

llСreatedTasks.removeAllViews();

for (Task task : createdTasks) {

llСreatedTasks.addView(createDesignForTask(task));

}

} catch (SQLException ex) {

Toast.makeText(this, "Не вдалося завантажити завдання", Toast.LENGTH\_LONG).show();

finish();

}

}

/\*\*

\* Створює дизайн для завдання

\*

\* @param task завданння, для якого потрібно створити дизайн

\* @return дизайн завдання

\* @see Task

\* @see Intent

\*/

public View createDesignForTask(Task task) {

int height = (int) TypedValue.applyDimension(TypedValue.COMPLEX\_UNIT\_DIP, 75, getResources().getDisplayMetrics());

LinearLayout linearLayout = new LinearLayout(this);

linearLayout.setOrientation(LinearLayout.HORIZONTAL);

linearLayout.setBackgroundResource(task.getColorTask(tabLayout.getSelectedTabPosition()));

linearLayout.setOnClickListener((View v) -> {

Intent intent = new Intent(TasksViewerActivity.this, TaskViewerActivity.class);

intent.putExtra(Task.class.getSimpleName(), task);

intent.putExtra(User.class.getSimpleName(), user);

startActivity(intent);

});

LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams

(LinearLayout.LayoutParams.MATCH\_PARENT, height);

layoutParams.setMargins(5, 5, 5, 5);

linearLayout.setVerticalGravity(Gravity.CENTER\_VERTICAL);

linearLayout.setLayoutParams(layoutParams);

TextView tv = new TextView(this);

tv.setText(task.getName());

tv.setTextSize(20);

tv.setAllCaps(false);

CheckBox checkBox = new CheckBox(this);

checkBox.setChecked(task.getDone());

checkBox.setOnCheckedChangeListener((CompoundButton buttonView, boolean isChecked) -> {

if (isChecked) {

TaskDao.setDone(task, true);

} else {

TaskDao.setDone(task, false);

}

onResume();

});

linearLayout.addView(checkBox);

linearLayout.addView(tv);

return linearLayout;

}

/\*\*

\* Фільтрує масив завдань

\*

\* @param tasks масив із завданнями

\* @see Task

\*/

private void filterTasks(Task[] tasks) {

switch (filterType) {

case "Даті публікації (зростання)": {

for (int i = 0; i < tasks.length; i++) {

for (int j = 1; j < tasks.length; j++) {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

if (tasks[j - 1].getDateTimeOfPublication().isAfter(tasks[j].getDateTimeOfPublication())) {

Task temp = tasks[j - 1];

tasks[j - 1] = tasks[j];

tasks[j] = temp;

}

}

}

}

break;

}

case "Даті публікації (спадання)": {

for (int i = 0; i < tasks.length; i++) {

for (int j = 1; j < tasks.length; j++) {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

if (tasks[j - 1].getDateTimeOfPublication().isBefore(tasks[j].getDateTimeOfPublication())) {

Task temp = tasks[j - 1];

tasks[j - 1] = tasks[j];

tasks[j] = temp;

}

}

}

}

break;

}

case "Даті закінчення (зростання)": {

for (int i = 0; i < tasks.length; i++) {

for (int j = 1; j < tasks.length; j++) {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

if (tasks[j - 1].getEndDateTime().isAfter(tasks[j].getEndDateTime())) {

Task temp = tasks[j - 1];

tasks[j - 1] = tasks[j];

tasks[j] = temp;

}

}

}

}

break;

}

case "Даті закінчення (спадання)": {

for (int i = 0; i < tasks.length; i++) {

for (int j = 1; j < tasks.length; j++) {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

if (tasks[j - 1].getEndDateTime().isBefore(tasks[j].getEndDateTime())) {

Task temp = tasks[j - 1];

tasks[j - 1] = tasks[j];

tasks[j] = temp;

}

}

}

}

break;

}

case "Даті виконання (зростання)": {

for (int i = 0; i < tasks.length; i++) {

for (int j = 1; j < tasks.length; j++) {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

if (tasks[j - 1].getDateTimeExecution() != null && tasks[j].getDateTimeExecution() != null) {

if (tasks[j - 1].getDateTimeExecution().isAfter(tasks[j].getDateTimeExecution())) {

Task temp = tasks[j - 1];

tasks[j - 1] = tasks[j];

tasks[j] = temp;

}

}

}

}

}

break;

}

case "Даті виконання (спадання)": {

for (int i = 0; i < tasks.length; i++) {

for (int j = 1; j < tasks.length; j++) {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

if (tasks[j - 1].getDateTimeExecution() != null && tasks[j].getDateTimeExecution() != null) {

if (tasks[j - 1].getDateTimeExecution().isBefore(tasks[j].getDateTimeExecution())) {

Task temp = tasks[j - 1];

tasks[j - 1] = tasks[j];

tasks[j] = temp;

}

}

}

}

}

break;

}

default: {

Toast.makeText(this, "Помилка під час фільтрацій", Toast.LENGTH\_LONG).show();

break;

}

}

}

}

**TaskViewerActivity**

package com.rosokha.todolist.task;

import android.app.Activity;

import android.content.Intent;

import android.content.pm.ActivityInfo;

import android.database.SQLException;

import android.graphics.Color;

import android.os.Build;

import android.os.Bundle;

import android.util.TypedValue;

import android.view.Gravity;

import android.view.View;

import android.widget.Button;

import android.widget.CheckBox;

import android.widget.CompoundButton;

import android.widget.DatePicker;

import android.widget.EditText;

import android.widget.LinearLayout;

import android.widget.TextView;

import android.widget.TimePicker;

import android.widget.Toast;

import androidx.annotation.Nullable;

import androidx.constraintlayout.widget.ConstraintLayout;

import com.rosokha.todolist.R;

import com.rosokha.todolist.dao.CommentDao;

import com.rosokha.todolist.dao.TaskDao;

import com.rosokha.todolist.dao.ViewProfileDao;

import com.rosokha.todolist.users.Leader;

import com.rosokha.todolist.users.ProfileViewerActivity;

import com.rosokha.todolist.users.User;

import com.rosokha.todolist.users.Worker;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

/\*\*

\* Представляє собою графічний користувацький інтерфейс перегляду завдання

\*

\* @see Activity

\*/

public class TaskViewerActivity extends Activity {

/\*\*

\* Назва завдання

\*/

private EditText etName;

/\*\*

\* Зміст завдання

\*/

private EditText etContent;

/\*\*

\* Керівник завдання

\*/

private TextView tvLeader;

/\*\*

\* Дата публікації завдання

\*/

private EditText etDateOfPublication;

/\*\*

\* Дата закінчення завдання

\*/

private EditText etEndDate;

/\*\*

\* Дата виконання завдання

\*/

private EditText etDateExecution;

/\*\*

\* Кнопка виходу з activity

\*/

private Button btnGoBack;

/\*\*

\* Кнопка позначення завдання як виконане

\*/

private Button btnMarkAsDone;

/\*\*

\* Кнопка відкриття сторінки створення завдання

\*/

private Button btnOpenPageCreateComment;

/\*\*

\* Кнопка редагування завдання

\*/

private Button btnEditTask;

/\*\*

\* Кнопка перегляду виконавців

\*/

private Button btnViewExecutors;

/\*\*

\* Кнопка додавання нових користувачів

\*/

private Button btnAddNewExecutors;

/\*\*

\* Кнопка зміни кінцевої дати і часу

\*/

private Button btnChangeEndDate;

/\*\*

\* Кнопка видалення завдання

\*/

private Button btnDeleteTask;

/\*\*

\* Контейнер з інформаціїєю про завдання

\*/

private LinearLayout llTaskInformation;

/\*\*

\* Контейнер створення коментаря

\*/

private ConstraintLayout clCreateComment;

/\*\*

\* Контейнер перегляду виконавців

\*/

private ConstraintLayout clExecutors;

/\*\*

\* Контейнер вводу дати і часу

\*/

private ConstraintLayout clInputDateAndTime;

/\*\*

\* Календар

\*/

private DatePicker dpDate;

/\*\*

\* Годинник

\*/

private TimePicker tpTime;

/\*\*

\* Завдання яке переглядається

\*/

private Task task = null;

/\*\*

\* Користувач який переглядає завдання

\*/

private User user = null;

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see Nullable

\* @see Activity#onCreate(Bundle)

\* @see Activity#setContentView(View)

\* @see Activity#setRequestedOrientation(int)

\* @see TaskViewerActivity#inicialComponents()

\* @see TaskViewerActivity#setTaskValuesForActivity()

\* @see TaskViewerActivity#checkTaskDone()

\* @see TaskViewerActivity#showMainViews()

\*/

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_view\_task);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);//блокує поворот екрану

inicialComponents();

setTaskValuesForActivity();

checkTaskDone();

showMainViews();

}

/\*\*

\* Ініціалізує компоненти

\*

\* @see Bundle

\* @see Task

\* @see User

\*/

private void inicialComponents() {

Bundle arguments = getIntent().getExtras();

if (arguments != null) {

task = (Task) arguments.getSerializable(Task.class.getSimpleName());

user = (User) arguments.getSerializable(User.class.getSimpleName());

}

etName = findViewById(R.id.etName);

etContent = findViewById(R.id.etContent);

tvLeader = findViewById(R.id.tvLeader);

etDateOfPublication = findViewById(R.id.tvDateOfPublication);

etEndDate = findViewById(R.id.tvEndDate);

etDateExecution = findViewById(R.id.tvDateExecution);

btnGoBack = findViewById(R.id.btnGoBack);

btnMarkAsDone = findViewById(R.id.btnMarkAsDone);

btnOpenPageCreateComment = findViewById(R.id.btnOpenPageCreateComment);

Button btnCreateComment = findViewById(R.id.btnCreateComment);

Button btnGoToViewTask = findViewById(R.id.btnGoToViewTask);

btnEditTask = findViewById(R.id.btnEditTask);

btnViewExecutors = findViewById(R.id.btnViewExecutors);

btnAddNewExecutors = findViewById(R.id.btnAddNewExecutors);

Button btnGoBackFromExecutors = findViewById(R.id.btnGoBackFromExecutors);

btnChangeEndDate = findViewById(R.id.btnChangeEndDate);

btnDeleteTask = findViewById(R.id.btnDeleteTask);

llTaskInformation = findViewById(R.id.llTaskInformation);

clCreateComment = findViewById(R.id.clCreateComment);

clExecutors = findViewById(R.id.clExecutors);

clInputDateAndTime = findViewById(R.id.activity\_input\_date\_and\_time);

dpDate = clInputDateAndTime.findViewById(R.id.dpDate);

tpTime = clInputDateAndTime.findViewById(R.id.tpTime);

tpTime.setIs24HourView(true);

btnGoBack.setOnClickListener((View v) -> finish());

btnMarkAsDone.setOnClickListener((View v) -> markAsDone());

btnOpenPageCreateComment.setOnClickListener((View v) -> {

hideMainViews();

clCreateComment.setVisibility(View.VISIBLE);

});

btnCreateComment.setOnClickListener((View v) -> createComment());

btnGoToViewTask.setOnClickListener((View v) -> showMainViews());

tvLeader.setOnClickListener((View v) -> {

Intent intent = new Intent(TaskViewerActivity.this, ProfileViewerActivity.class);

intent.putExtra(Leader.class.getSimpleName(), task.getLeader());

startActivity(intent);

});

btnEditTask.setOnClickListener((View v) -> updateTask());

btnViewExecutors.setOnClickListener((View v) -> loadExecutors());

btnAddNewExecutors.setOnClickListener((View v) -> {

Intent intent = new Intent(TaskViewerActivity.this, FinderExecutorsActivity.class);

intent.putExtra(Task.class.getSimpleName(), task);

startActivity(intent);

showMainViews();

});

btnGoBackFromExecutors.setOnClickListener((View v) -> showMainViews());

btnChangeEndDate.setOnClickListener((View v) -> changeEndDate());

btnDeleteTask.setOnClickListener((View v) -> deleteTask());

}

/\*\*

\* Встановлює значення завдання для activity

\*

\* @see CommentDao

\* @see CommentDao#getListOfComments(Task)

\* @see TaskViewerActivity#createDesignForComment(Comment)

\*/

private void setTaskValuesForActivity() {

etName.setText(task.getName());

etContent.setText(task.getContent() == null ? "Зміст не доданий" : task.getContent());

tvLeader.setText(task.getLeader().getLogin());

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

etDateOfPublication.setText(task.getDateTimeOfPublication().format(DateTimeFormatter.ofPattern("Дата: yyyy.MM.dd\nЧас: HH:mm")));

}

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

etEndDate.setText((task.getEndDateTime().format(DateTimeFormatter.ofPattern("Дата: yyyy.MM.dd\nЧас: HH:mm"))));

}

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

etDateExecution.setText(task.getDateTimeExecution() == null

? "Завдання не виконане"

: task.getDateTimeExecution().format(DateTimeFormatter.ofPattern("Дата: yyyy.MM.dd\nЧас: HH:mm")));

}

task.setComments(CommentDao.getListOfComments(task));

LinearLayout llComments = findViewById(R.id.llComments);

llComments.removeAllViews();

for (Comment comment : task.getComments()) {

llComments.addView(createDesignForComment(comment));

}

}

/\*\*

\* Перевіряє чи завдання є виконаним

\*/

private void checkTaskDone() {

if (task.getDone()) {

btnMarkAsDone.setText("Відмінити виконання");

btnMarkAsDone.setBackgroundResource(R.drawable.round\_delete\_button);

} else {

btnMarkAsDone.setText("Позначити як виконане");

btnMarkAsDone.setBackgroundResource(R.drawable.round\_green\_button);

}

}

/\*\*

\* Створює коментар

\*

\* @see CommentDao

\* @see CommentDao#createComment(Task, User, String)

\* @see Toast

\* @see TaskViewerActivity#setTaskValuesForActivity()

\* @see TaskViewerActivity#showMainViews()

\*/

private void createComment() {

EditText etComment = findViewById(R.id.etComment);

try {

if(etComment.getText().toString().length() > 0) {

CommentDao.createComment(task, user, etComment.getText().toString());

Toast.makeText(this, "Коментар успішно створений", Toast.LENGTH\_SHORT).show();

setTaskValuesForActivity();

clCreateComment.setVisibility(View.GONE);

showMainViews();

etComment.setText("");

} else {

throw new Exception();

}

} catch (Exception ex) {

Toast.makeText(this, "Не вдалося створити коментар", Toast.LENGTH\_SHORT).show();

}

}

/\*\*

\* Завантажує виконавців

\*

\* @see TaskViewerActivity#hideMainViews()

\* @see User

\*/

private void loadExecutors() {

hideMainViews();

clExecutors.setVisibility(View.VISIBLE);

LinearLayout llExecutors = findViewById(R.id.llExecutors);

llExecutors.removeAllViews();

task.setExecutors(TaskDao.getListOfExecutorsForTask(task));

for (User executor : task.getExecutors()) {

if (task.getExecutors().size() == 1) {

llExecutors.addView(createDesignForExecutor(executor, false));

} else {

llExecutors.addView(createDesignForExecutor(executor, true));

}

}

btnAddNewExecutors.setVisibility(user.getId() == task.getLeader().getId() ? View.VISIBLE : View.INVISIBLE);

}

/\*\*

\* Створює дизайн для коментаря

\*

\* @param comment коментар

\* @return елемент з оформленим коментарем

\* @see Comment

\* @see CommentDao

\* @see CommentDao#deleteComment(Comment) CommentDao#deleteComment(Comment)

\* @see Toast

\*/

public View createDesignForComment(Comment comment) {

LinearLayout linearLayout = new LinearLayout(this);

linearLayout.setOrientation(LinearLayout.VERTICAL);

LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams

(LinearLayout.LayoutParams.MATCH\_PARENT, LinearLayout.LayoutParams.WRAP\_CONTENT);

layoutParams.setMargins(5, 5, 5, 5);

linearLayout.setLayoutParams(layoutParams);

linearLayout.setPadding(5, 15, 5, 15);

linearLayout.setBackgroundResource(R.drawable.round\_comment);

TextView tvText = new TextView(this);

tvText.setTextSize(18);

tvText.setTextColor(Color.WHITE);

layoutParams = new LinearLayout.LayoutParams

(LinearLayout.LayoutParams.WRAP\_CONTENT, LinearLayout.LayoutParams.MATCH\_PARENT);

tvText.setLayoutParams(layoutParams);

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

tvText.setText(comment.getContent());

}

linearLayout.addView(tvText);

TextView tvAboutComment = new TextView(this);

tvAboutComment.setTextSize(18);

tvAboutComment.setTextColor(Color.YELLOW);

layoutParams = new LinearLayout.LayoutParams

(LinearLayout.LayoutParams.WRAP\_CONTENT, LinearLayout.LayoutParams.MATCH\_PARENT);

tvAboutComment.setLayoutParams(layoutParams);

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

tvAboutComment.setText("Автор: " + comment.getUser().getLogin() + "\n" +

"Дата написання:\n" +

comment.getDateTimeOfWriting().format(DateTimeFormatter.ofPattern("Дата: yyyy.MM.dd\nЧас: HH:mm")));

}

linearLayout.addView(tvAboutComment);

if (user.getId() == comment.getUser().getId()) {

Button btn = new Button(this);

btn.setText("Видалити");

layoutParams = new LinearLayout.LayoutParams

(LinearLayout.LayoutParams.WRAP\_CONTENT, LinearLayout.LayoutParams.WRAP\_CONTENT);

layoutParams.setMargins(0, 5, 0, 5);

btn.setLayoutParams(layoutParams);

btn.setBackgroundResource(R.drawable.round\_delete\_button);

btn.setOnClickListener((View v) -> {

try {

CommentDao.deleteComment(comment);

setTaskValuesForActivity();

} catch (SQLException ex) {

Toast.makeText(this, "Не вдалося видалити коментар", Toast.LENGTH\_LONG).show();

}

});

linearLayout.addView(btn);

}

return linearLayout;

}

/\*\*

\* Створює дизайн для відображення виконавця

\*

\* @param executor виконавець

\* @return елемент дизайну виконавця

\* @see User

\* @see Intent

\* @see Toast

\* @see TaskDao

\* @see TaskDao#deleteExecutorById(int, int) TaskDao#deleteExecutorById(int, int)

\* @see TaskViewerActivity#loadExecutors() TaskViewerActivity#loadExecutors()

\*/

public View createDesignForExecutor(User executor, boolean isCheckBox) {

int height = (int) TypedValue.applyDimension(TypedValue.COMPLEX\_UNIT\_DIP, 50, getResources().getDisplayMetrics());

LinearLayout linearLayout = new LinearLayout(this);

linearLayout.setOrientation(LinearLayout.HORIZONTAL);

linearLayout.setBackgroundResource(R.drawable.round\_user);

linearLayout.setOnClickListener((View v) -> {

Intent intent = new Intent(TaskViewerActivity.this, ProfileViewerActivity.class);

if (executor instanceof Leader) {

intent.putExtra(Leader.class.getSimpleName(), executor);

} else {

intent.putExtra(Worker.class.getSimpleName(), executor);

}

startActivity(intent);

});

LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams

(LinearLayout.LayoutParams.MATCH\_PARENT, height);

layoutParams.setMargins(0, 5, 0, 0);

linearLayout.setVerticalGravity(Gravity.CENTER\_VERTICAL);

linearLayout.setLayoutParams(layoutParams);

TextView tv = new TextView(this);

tv.setText(executor.getLogin());

tv.setTextSize(20);

tv.setAllCaps(false);

if (user.getId() == task.getLeader().getId()) {

CheckBox checkBox = new CheckBox(this);

checkBox.setChecked(true);

checkBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {

@Override

public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {

try {

TaskDao.deleteExecutorById(executor.getId(), task.getId());

Toast.makeText(TaskViewerActivity.this, "Виконавець успішно видалений", Toast.LENGTH\_SHORT).show();

task.setExecutors(TaskDao.getListOfExecutorsForTask(task));

loadExecutors();

} catch (SQLException ex) {

Toast.makeText(TaskViewerActivity.this, "Не вдалося видалити виконавця", Toast.LENGTH\_SHORT).show();

}

}

});

if (isCheckBox) {

checkBox.setVisibility(View.VISIBLE);

} else {

checkBox.setVisibility(View.INVISIBLE);

}

linearLayout.addView(checkBox);

}

linearLayout.addView(tv);

return linearLayout;

}

/\*\*

\* Помічає завдання як виконане

\*

\* @see TaskDao#setDone(Task, boolean)

\* @see TaskViewerActivity#checkTaskDone()

\* @see TaskViewerActivity#setTaskValuesForActivity()

\* @see Toast

\*/

private void markAsDone() {

try {

if (btnMarkAsDone.getText().toString().equals("Позначити як виконане")) {

TaskDao.setDone(task, true);

} else {

TaskDao.setDone(task, false);

}

task = TaskDao.getTaskByID(task.getId());

checkTaskDone();

setTaskValuesForActivity();

} catch (SQLException ex) {

Toast.makeText(this, "Не вдалося позначити завдання як виконане", Toast.LENGTH\_LONG).show();

}

}

/\*\*

\* Змінює завдання

\*

\* @see Toast

\* @see TaskDao

\* @see TaskDao#updateTask(Task)

\*/

private void updateTask() {

if (etName.getText().toString().equals("")) {

Toast.makeText(this, "Навзва завдання не може бути пустою", Toast.LENGTH\_LONG).show();

return;

}

task.setName(etName.getText().toString());

task.setContent(etContent.getText().toString());

try {

TaskDao.updateTask(task);

Toast.makeText(this, "Зміни збережено", Toast.LENGTH\_LONG).show();

} catch (SQLException ex) {

Toast.makeText(this, "Не вдалося зберегти зміни", Toast.LENGTH\_LONG).show();

}

}

/\*\*

\* Видаляє завдання

\*

\* @see Toast

\* @see TaskDao

\* @see TaskDao#deleteTask(Task)

\* @see Activity#finish()

\*/

private void deleteTask() {

try {

TaskDao.deleteTask(task);

Toast.makeText(this, "Завдання успішно видалено", Toast.LENGTH\_LONG).show();

finish();

} catch (SQLException ex) {

Toast.makeText(this, "Не вдалося видалити завдання", Toast.LENGTH\_LONG).show();

}

}

/\*\*

\* Змінює дату закінчення завдання

\*

\* @see TaskViewerActivity#hideMainViews()

\* @see TaskViewerActivity#showMainViews()

\*/

private void changeEndDate() {

hideMainViews();

clInputDateAndTime.setVisibility(View.VISIBLE);

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

dpDate.init(task.getEndDateTime().getYear(), task.getEndDateTime().getMonthValue() - 1,

task.getEndDateTime().getDayOfMonth(), null);

tpTime.setHour(task.getEndDateTime().getHour());

tpTime.setMinute(task.getEndDateTime().getMinute());

}

ConstraintLayout clDate = clInputDateAndTime.findViewById(R.id.clDate);

ConstraintLayout clTime = clInputDateAndTime.findViewById(R.id.clTime);

Button btnSaveDate = clInputDateAndTime.findViewById(R.id.btnSaveDate);

Button btnSaveTime = clInputDateAndTime.findViewById(R.id.btnSaveTime);

btnSaveDate.setOnClickListener((View v) -> {

clDate.setVisibility(View.GONE);

clTime.setVisibility(View.VISIBLE);

});

btnSaveTime.setOnClickListener((View v) -> {

clDate.setVisibility(View.VISIBLE);

clTime.setVisibility(View.GONE);

clInputDateAndTime.setVisibility(View.GONE);

showMainViews();

if (android.os.Build.VERSION.SDK\_INT >= android.os.Build.VERSION\_CODES.M) {

if (android.os.Build.VERSION.SDK\_INT >= android.os.Build.VERSION\_CODES.O) {

LocalDateTime newEndDateTime = LocalDateTime.of(dpDate.getYear(), dpDate.getMonth() + 1, dpDate.getDayOfMonth(),

tpTime.getHour(), tpTime.getMinute());

if(newEndDateTime.isBefore(task.getDateTimeOfPublication())) {

Toast.makeText(this, "Дата виконання не може бути раніше дати публікації", Toast.LENGTH\_LONG).show();

return;

}

task.setEndDateTime(newEndDateTime);

etEndDate.setText(newEndDateTime.format(DateTimeFormatter.ofPattern("Дата: yyyy.MM.dd\nЧас: HH:mm")));

}

}

});

}

/\*\*

\* Приховує головні елементи activity при появі вікна вводу

\*/

private void hideMainViews() {

llTaskInformation.setVisibility(View.GONE);

btnGoBack.setVisibility(View.GONE);

btnMarkAsDone.setVisibility(View.GONE);

btnOpenPageCreateComment.setVisibility(View.GONE);

btnEditTask.setVisibility(View.GONE);

btnViewExecutors.setVisibility(View.GONE);

}

/\*\*

\* Показує головні елементи activity при закритті вікна вводу

\*/

private void showMainViews() {

if (task.getLeader().getId() == user.getId()) {

btnEditTask.setVisibility(View.VISIBLE);

btnMarkAsDone.setVisibility(View.INVISIBLE);

etName.setEnabled(true);

etContent.setEnabled(true);

btnChangeEndDate.setVisibility(View.VISIBLE);

btnDeleteTask.setVisibility(View.VISIBLE);

} else {

btnEditTask.setVisibility(View.INVISIBLE);

btnMarkAsDone.setVisibility(View.VISIBLE);

etName.setEnabled(false);

etContent.setEnabled(false);

btnChangeEndDate.setVisibility(View.GONE);

btnDeleteTask.setVisibility(View.GONE);

}

llTaskInformation.setVisibility(View.VISIBLE);

btnGoBack.setVisibility(View.VISIBLE);

btnOpenPageCreateComment.setVisibility(View.VISIBLE);

btnViewExecutors.setVisibility(View.VISIBLE);

clCreateComment.setVisibility(View.GONE);

clExecutors.setVisibility(View.GONE);

}

}

**FavoriteUsersViewerActivity**

package com.rosokha.todolist.users;

import android.annotation.SuppressLint;

import android.app.Activity;

import android.content.Intent;

import android.content.pm.ActivityInfo;

import android.database.SQLException;

import android.os.Bundle;

import android.util.TypedValue;

import android.view.Gravity;

import android.view.View;

import android.widget.Button;

import android.widget.CheckBox;

import android.widget.CompoundButton;

import android.widget.LinearLayout;

import android.widget.TextView;

import android.widget.Toast;

import androidx.annotation.Nullable;

import com.rosokha.todolist.R;

import com.rosokha.todolist.dao.LoaderOfFavoriteUsersDao;

import com.rosokha.todolist.dao.ViewProfileDao;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Представляє собою графічний користувацький інтерфейс перегляду вибраних користувачів

\*

\* @see Activity

\*/

public class FavoriteUsersViewerActivity extends Activity {

/\*\*

\* Користувач, який переглядає вибраних працівників

\*/

User user;

/\*\*

\* Список вибраних користувачів

\*/

List<User> favoriteUsers = new ArrayList<>();

/\*\*

\* Контейнер для вибраних користувачів

\*/

LinearLayout llFavoriteUsers;

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see Nullable

\* @see Activity#onCreate(Bundle)

\* @see Activity#setContentView(View)

\* @see Activity#setRequestedOrientation(int)

\* @see FavoriteUsersViewerActivity#inicialComponents()

\* @see FavoriteUsersViewerActivity#showFavoriteUsers()

\*/

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_view\_favorite\_users);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);//блокує поворот екрану

inicialComponents();

showFavoriteUsers();

}

/\*\*

\* Викликається при поверненні користувача в дане activity

\*

\* @see Activity#onResume()

\* @see FavoriteUsersViewerActivity#showFavoriteUsers()

\*/

@Override

protected void onResume() {

super.onResume();

showFavoriteUsers();

}

/\*\*

\* Ініціалізує компоненти

\*

\* @see Bundle

\* @see Toast

\* @see User

\*/

private void inicialComponents() {

Button btnFindNewUsers = findViewById(R.id.btnFindNewUsers);

Button btnGoBack = findViewById(R.id.btnGoBack);

llFavoriteUsers = findViewById(R.id.llFavoriteUsers);

btnGoBack.setOnClickListener((View v) -> finish());

btnFindNewUsers.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent(FavoriteUsersViewerActivity.this, SearchUsersActivity.class);

startActivity(intent);

}

});

Bundle arguments = getIntent().getExtras();

if (arguments != null) {

user = (User) arguments.getSerializable(User.class.getSimpleName());

} else {

Toast.makeText(this, "Сталася помилка при передачі аргументів", Toast.LENGTH\_LONG).show();

}

}

/\*\*

\* Показує вибраних користувачів

\*

\* @see LoaderOfFavoriteUsersDao

\* @see LoaderOfFavoriteUsersDao#loadFavoriteUsers(int)

\* @see User

\* @see Toast

\*/

@SuppressLint("Range")

private void showFavoriteUsers() {

llFavoriteUsers.removeAllViews();

try {

favoriteUsers = LoaderOfFavoriteUsersDao.loadFavoriteUsers(user.getId());

} catch (SQLException ex) {

Toast.makeText(this, "Неможливо завантажити вибраних користувачів", Toast.LENGTH\_LONG).show();

}

for (User favoriteUser : favoriteUsers) {

int height = (int) TypedValue.applyDimension(TypedValue.COMPLEX\_UNIT\_DIP, 50, getResources().getDisplayMetrics());

LinearLayout linearLayout = new LinearLayout(this);

linearLayout.setOrientation(LinearLayout.HORIZONTAL);

linearLayout.setBackgroundResource(R.drawable.round\_user);

linearLayout.setOnClickListener((View v) -> openProfile(favoriteUser));

LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams

(LinearLayout.LayoutParams.MATCH\_PARENT, height);

layoutParams.setMargins(0, 5, 0, 0);

linearLayout.setVerticalGravity(Gravity.CENTER\_VERTICAL);

linearLayout.setLayoutParams(layoutParams);

TextView tv = new TextView(this);

tv.setText(favoriteUser.getLogin());

tv.setTextSize(20);

tv.setAllCaps(false);

CheckBox checkBox = new CheckBox(this);

checkBox.setChecked(true);

checkBox.setOnCheckedChangeListener((CompoundButton buttonView, boolean isChecked) -> {

if (isChecked) {

ViewProfileDao.addFavoriteUser(user.getId(), favoriteUser.getId());

showFavoriteUsers();

} else {

ViewProfileDao.deleteFavoriteUser(user.getId(), favoriteUser.getId());

showFavoriteUsers();

}

});

linearLayout.addView(checkBox);

linearLayout.addView(tv);

llFavoriteUsers.addView(linearLayout);

}

}

/\*\*

\* Відкриває профіль вибраного користувача

\*

\* @param user користувач, профіль якого потрібно відкрити

\* @see User

\* @see Leader

\* @see Worker

\* @see Intent

\* @see ProfileViewerActivity

\*/

private void openProfile(User user) {

if (user instanceof Leader) {

Leader tempLeader = (Leader) user;

Intent intent = new Intent(FavoriteUsersViewerActivity.this, ProfileViewerActivity.class);

intent.putExtra(Leader.class.getSimpleName(), tempLeader);

startActivity(intent);

} else {

Worker tempWorker = (Worker) user;

Intent intent = new Intent(FavoriteUsersViewerActivity.this, ProfileViewerActivity.class);

intent.putExtra(Worker.class.getSimpleName(), tempWorker);

startActivity(intent);

}

}

}

**HomePageOfLeaderActivity**

package com.rosokha.todolist.users;

import android.app.Activity;

import android.content.Intent;

import android.content.pm.ActivityInfo;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.Toast;

import androidx.annotation.Nullable;

import com.rosokha.todolist.R;

import com.rosokha.todolist.notifications.NotificationService;

import com.rosokha.todolist.task.TasksViewerActivity;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

/\*\*

\* Представляє собою графічний користувацький інтерфейс кабінету керівника

\*

\* @see Activity

\*/

public class HomePageOfLeaderActivity extends Activity {

/\*\*

\* Об'єкт класу Leader, який представляє собою керівника

\*/

private static Leader currentLeader;

/\*\*

\* Сервіс для перевірки повідомлень

\*/

private NotificationService notificationService;

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see Nullable

\* @see Activity#onCreate(Bundle)

\* @see Activity#setContentView(View)

\* @see Activity#setRequestedOrientation(int

\* @see HomePageOfLeaderActivity#inicialComponents()

\* @see Intent

\* @see TasksViewerActivity

\*/

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_home\_page\_of\_leader);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);//блокує поворот екрану

inicialComponents();

notificationService = new NotificationService(currentLeader, this);

notificationService.start();

Intent intent = new Intent(HomePageOfLeaderActivity.this, TasksViewerActivity.class);

intent.putExtra(User.class.getSimpleName(), currentLeader);

startActivity(intent);

}

/\*\*

\* Метод, який ініціалізує компоненти

\*

\* @see Bundle

\* @see Leader

\* @see Toast

\* @see Intent

\* @see ProfileViewerActivity

\* @see FavoriteUsersViewerActivity

\* @see TasksViewerActivity

\* @see Activity#finish()

\*/

private void inicialComponents() {

Bundle arguments = getIntent().getExtras();

if (arguments != null) {

currentLeader = (Leader) arguments.getSerializable(Leader.class.getSimpleName());

} else {

Toast.makeText(this, "Сталася помилка при передачі аргументів", Toast.LENGTH\_LONG).show();

return;

}

Button btnViewProfile = findViewById(R.id.btnViewProfile);

Button btnViewFavoriteUsers = findViewById(R.id.btnFavoriteUsers);

Button btnViewTasks = findViewById(R.id.btnViewTasks);

Button btnLogOut = findViewById(R.id.btnLogOut);

btnViewProfile.setOnClickListener((View v) -> {

Intent intent = new Intent(HomePageOfLeaderActivity.this, ProfileViewerActivity.class);

intent.putExtra(Leader.class.getSimpleName(), currentLeader);

startActivity(intent);

});

btnViewFavoriteUsers.setOnClickListener((View v) -> {

Intent intent = new Intent(HomePageOfLeaderActivity.this, FavoriteUsersViewerActivity.class);

intent.putExtra(User.class.getSimpleName(), currentLeader);

startActivity(intent);

});

btnViewTasks.setOnClickListener((View v) -> {

Intent intent = new Intent(HomePageOfLeaderActivity.this, TasksViewerActivity.class);

intent.putExtra(User.class.getSimpleName(), currentLeader);

startActivity(intent);

});

btnLogOut.setOnClickListener((View v) -> {

notificationService.interrupt();

cleanFile();

currentLeader = null;

finish();

});

}

/\*\*

\* Очищує файл з даними користувача

\*

\* @see FileOutputStream

\* @see Activity#openFileOutput(String, int)

\* @see FileOutputStream#write(int)

\*/

private void cleanFile(){

String text = "";

FileOutputStream fos = null;

try {

fos = openFileOutput("user.txt", MODE\_PRIVATE);

} catch (FileNotFoundException e) {

}

try {

fos.write(text.getBytes());

} catch (IOException e) {

}

if (fos != null) {

try {

fos.close();

} catch (IOException e) {

}

}

}

/\*\*

\* Викликається при натиску клавіші "Назад"

\*/

@Override

public void onBackPressed() {

}

/\*\*

\* Getter для поля currentLeader

\*

\* @return поточного керівника

\* @see Leader

\*/

public static Leader getCurrentLeader() {

return currentLeader;

}

}

**HomePageOfWorkerActivity**

package com.rosokha.todolist.users;

import android.app.Activity;

import android.content.Intent;

import android.content.pm.ActivityInfo;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.Toast;

import androidx.annotation.Nullable;

import com.rosokha.todolist.R;

import com.rosokha.todolist.notifications.NotificationService;

import com.rosokha.todolist.task.TasksViewerActivity;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

/\*\*

\* Представляє собою графічний користувацький інтерфейс кабінету працівника

\*

\* @see Activity

\*/

public class HomePageOfWorkerActivity extends Activity {

/\*\*

\* Об'єкт класу Worker, який представляє собою працівника

\*/

private static Worker currentWorker;

/\*\*

\* Сервіс для перевірки повідомлень

\*/

private NotificationService notificationService;

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see Nullable

\* @see Activity#onCreate(Bundle)

\* @see Activity#setContentView(View)

\* @see Activity#setRequestedOrientation(int

\* @see HomePageOfWorkerActivity#inicialComponents()

\* @see Intent

\* @see TasksViewerActivity

\*/

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_home\_page\_of\_worker);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);//блокує поворот екрану

inicialComponents();

notificationService = new NotificationService(currentWorker, this);

notificationService.start();

Intent intent = new Intent(HomePageOfWorkerActivity.this, TasksViewerActivity.class);

intent.putExtra(User.class.getSimpleName(), currentWorker);

startActivity(intent);

}

/\*\*

\* Метод, який ініціалізує компоненти

\*

\* @see Bundle

\* @see Leader

\* @see Toast

\* @see Intent

\* @see ProfileViewerActivity

\* @see FavoriteUsersViewerActivity

\* @see TasksViewerActivity

\* @see Activity#finish()

\*/

private void inicialComponents() {

Bundle arguments = getIntent().getExtras();

if (arguments != null) {

currentWorker = (Worker) arguments.getSerializable(Worker.class.getSimpleName());

} else {

Toast.makeText(this, "Сталася помилка при передачі аргументів", Toast.LENGTH\_LONG).show();

return;

}

Button btnViewProfile = findViewById(R.id.btnViewProfile);

Button btnViewFavoriteUsers = findViewById(R.id.btnFavoriteUsers);

Button btnViewTasks = findViewById(R.id.btnViewTasks);

Button btnLogOut = findViewById(R.id.btnLogOut);

btnViewProfile.setOnClickListener((View v) -> {

Intent intent = new Intent(HomePageOfWorkerActivity.this, ProfileViewerActivity.class);

intent.putExtra(Worker.class.getSimpleName(), currentWorker);

startActivity(intent);

});

btnViewFavoriteUsers.setOnClickListener((View v) -> {

Intent intent = new Intent(HomePageOfWorkerActivity.this, FavoriteUsersViewerActivity.class);

intent.putExtra(User.class.getSimpleName(), currentWorker);

startActivity(intent);

});

btnViewTasks.setOnClickListener((View v) -> {

Intent intent = new Intent(HomePageOfWorkerActivity.this, TasksViewerActivity.class);

intent.putExtra(User.class.getSimpleName(), currentWorker);

startActivity(intent);

});

btnLogOut.setOnClickListener((View v) -> {

notificationService.interrupt();

cleanFile();

currentWorker = null;

finish();

});

}

/\*\*

\* Очищує файл з даними користувача

\*

\* @see FileOutputStream

\* @see Activity#openFileOutput(String, int)

\* @see FileOutputStream#write(int)

\*/

private void cleanFile(){

String text = "";

FileOutputStream fos = null;

try {

fos = openFileOutput("user.txt", MODE\_PRIVATE);

} catch (FileNotFoundException e) {

}

try {

fos.write(text.getBytes());

} catch (IOException e) {

}

if (fos != null) {

try {

fos.close();

} catch (IOException e) {

}

}

}

/\*\*

\* Викликається при натиску клавіші "Назад"

\*/

@Override

public void onBackPressed() { }

/\*\*

\* Getter для поля currentWorker

\*

\* @return поточного працівника

\* @see Worker

\*/

public static Worker getCurrentWorker() {

return currentWorker;

}

}

**Leader**

package com.rosokha.todolist.users;

import android.database.SQLException;

import com.rosokha.todolist.dao.UserDao;

/\*\*

\* Клас Leader представляє собою керівника, який створює завдання

\*

\* @see User

\*/

public class Leader extends User {

/\*\*

\* Оновлює логін керівника

\*

\* @param newLogin новий логін

\* @throws SQLException помилка під час оновлення логіну

\* @see UserDao

\* @see UserDao#updateLogin(User, String)

\*/

@Override

public void updateLogin(String newLogin) throws SQLException {

UserDao.updateLogin(this, newLogin);

setLogin(newLogin);

}

/\*\*

\* Оновлює ПІБ керівника

\*

\* @param newUserName новий ПІБ

\* @throws SQLException помилка під час оновлення ПІБ

\* @see UserDao

\* @see UserDao#updateName(User, String)

\*/

@Override

public void updateUserName(String newUserName) throws SQLException {

UserDao.updateName(this, newUserName);

setName(newUserName);

}

}

**ProfileViewerActivity**

package com.rosokha.todolist.users;

import android.app.Activity;

import android.content.pm.ActivityInfo;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.CheckBox;

import android.widget.EditText;

import android.widget.TextView;

import android.widget.Toast;

import androidx.annotation.Nullable;

import com.rosokha.todolist.R;

import com.rosokha.todolist.dao.ViewProfileDao;

/\*\*

\* Представляє собою графічний користувацький інтерфейс перегляду профіля користувача

\*

\* @see Activity

\*/

public class ProfileViewerActivity extends Activity {

/\*\*

\* Заголовок

\*/

TextView tvProfile;

/\*\*

\* Логін користувача

\*/

EditText etLogin;

/\*\*

\* ПІБ користувача

\*/

EditText etUserName;

/\*\*

\* Кнопка збереження змін

\*/

Button btnSaveChanges;

/\*\*

\* CheckBox вибору користувача до вибраних

\*/

CheckBox chkFavoriteUser;

/\*\*

\* Працівник, аакаунт якого потрібно переглянути

\*/

Worker worker = null;

/\*\*

\* Керівник, акаунт якого потрібно переглянути

\*/

Leader leader = null;

/\*\*

\* Працівник який переглядає акаунт

\*/

Worker currentWorker = null;

/\*\*

\* Керівник який переглядає акаунт

\*/

Leader currentLeader = null;

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see Nullable

\* @see Activity#onCreate(Bundle)

\* @see Activity#setContentView(View)

\* @see Activity#setRequestedOrientation(int)

\* @see ProfileViewerActivity#inicialComponents()

\* @see ProfileViewerActivity#setThisUser()

\* @see ProfileViewerActivity#checkUser()

\*/

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_view\_profile);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);//блокує поворот екрану

inicialComponents();

setThisUser();

checkUser();

}

/\*\*

\* Ініціалізує компоненти

\*

\* @see HomePageOfLeaderActivity

\* @see HomePageOfWorkerActivity

\* @see HomePageOfLeaderActivity#getCurrentLeader()

\* @see HomePageOfWorkerActivity#getCurrentWorker()

\*/

private void inicialComponents() {

currentWorker = HomePageOfWorkerActivity.getCurrentWorker();

currentLeader = HomePageOfLeaderActivity.getCurrentLeader();

tvProfile = findViewById(R.id.tvProfileTitle);

etLogin = findViewById(R.id.etLogin);

etUserName = findViewById(R.id.etUserName);

Button btnGoBack = findViewById(R.id.btnGoBack);

btnSaveChanges = findViewById(R.id.btnSaveChanges);

chkFavoriteUser = findViewById(R.id.chkFavoriteUser);

btnGoBack.setOnClickListener((View v) -> finish());

btnSaveChanges.setOnClickListener((View v) -> saveChanges());

chkFavoriteUser.setOnClickListener((View v) -> addOrDeleteFavoriteUser());

}

/\*\*

\* Встановлює профілько користувача якого потрібно переглянути

\*

\* @see Bundle

\*/

private void setThisUser() {

Bundle arguments = getIntent().getExtras();

if (arguments == null) {

Toast.makeText(this, "Сталася помилка при передачі аргументів", Toast.LENGTH\_LONG).show();

return;

}

leader = (Leader) arguments.getSerializable(Leader.class.getSimpleName());

worker = (Worker) arguments.getSerializable(Worker.class.getSimpleName());

}

/\*\*

\* Встановлює видимість кнопок та можливість редагування профілю в залежності від

\* користувача який переглядає профіль, та від профілю користувача якого переглядають

\*

\* @see ViewProfileDao

\* @see ViewProfileDao#checkUserForFavorite(int, int)

\*/

private void checkUser() {

if (leader != null) {

etLogin.setText(leader.getLogin());

etUserName.setText(leader.getName());

if (currentLeader != null) {

//Перегляд керівником свій профіль

if (leader.getId() == currentLeader.getId()) {

chkFavoriteUser.setVisibility(View.GONE);

btnSaveChanges.setVisibility(View.VISIBLE);

etLogin.setFocusableInTouchMode(true);

etUserName.setFocusableInTouchMode(true);

tvProfile.setText("Ваш профіль");

}

//Перегляд керівником профіль керівника

else {

chkFavoriteUser.setVisibility(View.VISIBLE);

btnSaveChanges.setVisibility(View.GONE);

etLogin.setFocusable(false);

etUserName.setFocusable(false);

tvProfile.setText("Профіль керівника");

chkFavoriteUser.setChecked(ViewProfileDao.checkUserForFavorite(currentLeader.getId(), leader.getId()));

}

}

//Перегляд працівником профіль керівника

else {

chkFavoriteUser.setVisibility(View.VISIBLE);

btnSaveChanges.setVisibility(View.GONE);

etLogin.setFocusable(false);

etUserName.setFocusable(false);

tvProfile.setText("Профіль керівника");

chkFavoriteUser.setChecked(ViewProfileDao.checkUserForFavorite(currentWorker.getId(), leader.getId()));

}

}

if (worker != null) {

etLogin.setText(worker.getLogin());

etUserName.setText(worker.getName());

if (currentWorker != null) {

//Перегляд працівником свій профіль

if (worker.getId() == currentWorker.getId()) {

chkFavoriteUser.setVisibility(View.GONE);

btnSaveChanges.setVisibility(View.VISIBLE);

etLogin.setFocusableInTouchMode(true);

etUserName.setFocusableInTouchMode(true);

tvProfile.setText("Ваш профіль");

} else {

//Перегляд працівником профіль працівника

chkFavoriteUser.setVisibility(View.VISIBLE);

btnSaveChanges.setVisibility(View.GONE);

etLogin.setFocusable(false);

etUserName.setFocusable(false);

tvProfile.setText("Профіль працівника");

chkFavoriteUser.setChecked(ViewProfileDao.checkUserForFavorite(currentWorker.getId(), worker.getId()));

}

}

//Перегляд керівником профіль працівника

else {

chkFavoriteUser.setVisibility(View.VISIBLE);

btnSaveChanges.setVisibility(View.GONE);

etLogin.setFocusable(false);

etUserName.setFocusable(false);

tvProfile.setText("Профіль працівника");

chkFavoriteUser.setChecked(ViewProfileDao.checkUserForFavorite(currentLeader.getId(), worker.getId()));

}

}

}

/\*\*

\* Зберігає зміни логіну та імені користувача

\*

\* @see Leader#updateLogin(String)

\* @see Leader#updateUserName(String)

\* @see Worker#updateLogin(String)

\* @see Worker#updateUserName(String)

\* @see Toast

\* @see Activity#finish()

\*/

private void saveChanges() {

if (checkData() == false) {

return;

}

if (currentLeader != null) {

currentLeader.updateLogin(etLogin.getText().toString());

currentLeader.updateUserName(etUserName.getText().toString());

}

if (currentWorker != null) {

currentWorker.updateLogin(etLogin.getText().toString());

currentWorker.updateUserName(etUserName.getText().toString());

}

Toast.makeText(this, "Ваші дані успішно збережені", Toast.LENGTH\_LONG).show();

finish();

}

/\*\*

\* Перевіряє на коректність введених даних

\*

\* @return повертає true, якщо дані введені коректно. Плвертає false, якщо дані введені не коректно

\* @see Toast

\* @see ViewProfileDao

\* @see ViewProfileDao#validationLoginWithoutCurrentUser(String, String)

\*/

private boolean checkData() {

if (etLogin.length() < 3) {

Toast.makeText(this, "Логін повинен містити від трьох символів", Toast.LENGTH\_LONG).show();

return false;

}

if (etUserName.length() < 1) {

Toast.makeText(this, "ПІБ користувача не може бути пустим", Toast.LENGTH\_LONG).show();

return false;

}

if(currentLeader != null) {

if (!ViewProfileDao.validationLoginWithoutCurrentUser(etLogin.getText().toString(), currentLeader.getLogin())) {

Toast.makeText(this, "Користувач з таким логіном вже існує", Toast.LENGTH\_LONG).show();

return false;

}

} else {

if (!ViewProfileDao.validationLoginWithoutCurrentUser(etLogin.getText().toString(), currentWorker.getLogin())) {

Toast.makeText(this, "Користувач з таким логіном вже існує", Toast.LENGTH\_LONG).show();

return false;

}

}

return true;

}

/\*\*

\* Додає або видаляє користувача з вибраних

\*

\* @see ViewProfileDao

\* @see ViewProfileDao#addFavoriteUser(int, int)

\* @see ViewProfileDao#deleteFavoriteUser(int, int)

\*/

private void addOrDeleteFavoriteUser() {

if (chkFavoriteUser.isChecked()) {

if (currentLeader != null) {

if (leader != null) {

ViewProfileDao.addFavoriteUser(currentLeader.getId(), leader.getId());

} else {

ViewProfileDao.addFavoriteUser(currentLeader.getId(), worker.getId());

}

}

if (currentWorker != null) {

if (leader != null) {

ViewProfileDao.addFavoriteUser(currentWorker.getId(), worker.getId());

} else {

ViewProfileDao.addFavoriteUser(currentWorker.getId(), leader.getId());

}

}

} else {

if (currentLeader != null) {

if (leader != null) {

ViewProfileDao.deleteFavoriteUser(currentLeader.getId(), leader.getId());

} else {

ViewProfileDao.deleteFavoriteUser(currentLeader.getId(), worker.getId());

}

}

if (currentWorker != null) {

if (leader != null) {

ViewProfileDao.deleteFavoriteUser(currentWorker.getId(), worker.getId());

} else {

ViewProfileDao.deleteFavoriteUser(currentWorker.getId(), leader.getId());

}

}

}

}

}

**SearchUsersActivity**

package com.rosokha.todolist.users;

import android.annotation.\*;

import android.app.\*;

import android.content.\*;

import android.content.pm.\*;

import android.database.\*;

import android.os.\*;

import android.text.\*;

import android.util.TypedValue;

import android.view.\*;

import android.widget.\*;

import androidx.annotation.\*;

import com.rosokha.todolist.\*;

import com.rosokha.todolist.dao.\*;

import java.util.\*;

/\*\*

\* Представляє собою графічний користувацький інтерфейс пошуку користувачів

\*

\* @see Activity

\*/

public class SearchUsersActivity extends Activity {

/\*\*

\* Користувач, який шукає інших користувачів

\*/

User user;

/\*\*

\* Список знайдених користувачів

\*/

List<User> foundUsers = new ArrayList<>();

/\*\*

\* Контейнер у якому відображуються знайдені користувачі

\*/

LinearLayout llFoundUsers;

/\*\*

\* Проводиться початкове налаштування activity

\*

\* @param savedInstanceState попередня властивість activity, якщо вона була збережена

\* @see Bundle

\* @see Activity#onCreate(Bundle)

\* @see Activity#setContentView(View)

\* @see Activity#setRequestedOrientation(int)

\* @see SearchUsersActivity#inicialComponents()

\*/

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_search\_users);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);//блокує поворот екрану

inicialComponents();

}

/\*\*

\* Викликається при поверненні користувача в дане activity

\*

\* @see Activity#onResume()

\* @see SearchUsersActivity#findUsers(String)

\*/

@Override

protected void onResume() {

super.onResume();

EditText etSearch = findViewById(R.id.etSearch);

findUsers(etSearch.getText().toString());

}

/\*\*

\* Ініціалізує компоненти

\*

\* @see HomePageOfLeaderActivity

\* @see HomePageOfWorkerActivity

\* @see HomePageOfLeaderActivity#getCurrentLeader()

\* @see HomePageOfWorkerActivity#getCurrentWorker()

\*/

private void inicialComponents() {

if (HomePageOfLeaderActivity.getCurrentLeader() != null) {

user = HomePageOfLeaderActivity.getCurrentLeader();

} else {

user = HomePageOfWorkerActivity.getCurrentWorker();

}

EditText etSearch = findViewById(R.id.etSearch);

llFoundUsers = findViewById(R.id.llFoundUsers);

Button btnGoBack = findViewById(R.id.btnGoBack);

btnGoBack.setOnClickListener((View v) -> finish());

etSearch.addTextChangedListener(new TextWatcher() {

@Override

public void beforeTextChanged(CharSequence s, int start, int count, int after) {

}

@Override

public void onTextChanged(CharSequence s, int start, int before, int count) {

}

@Override

public void afterTextChanged(Editable s) {

findUsers(etSearch.getText().toString());

}

});

}

/\*\*

\* Шукає користувачів

\*

\* @param login логін по якому потрібно здійснити пошук

\* @see SearchDao

\* @see SearchDao#findUsers(String)

\* @see Toast

\* @see User

\*/

@SuppressLint("Range")

private void findUsers(String login) {

llFoundUsers.removeAllViews();

try {

foundUsers = SearchDao.findUsers(login);

} catch (SQLException ex) {

Toast.makeText(this, "Пошук не працює", Toast.LENGTH\_LONG).show();

return;

}

for (User tempUser : foundUsers) {

int height = (int) TypedValue.applyDimension(TypedValue.COMPLEX\_UNIT\_DIP, 50, getResources().getDisplayMetrics());

LinearLayout linearLayout = new LinearLayout(this);

linearLayout.setOrientation(LinearLayout.HORIZONTAL);

linearLayout.setBackgroundResource(R.drawable.round\_user);

linearLayout.setOnClickListener((View v) -> openProfile(tempUser));

LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams

(LinearLayout.LayoutParams.MATCH\_PARENT, height);

layoutParams.setMargins(0, 5, 0, 0);

linearLayout.setVerticalGravity(Gravity.CENTER\_VERTICAL);

linearLayout.setLayoutParams(layoutParams);

TextView tv = new TextView(this);

tv.setText(tempUser.getLogin());

tv.setTextSize(20);

tv.setAllCaps(false);

CheckBox checkBox = new CheckBox(this);

checkBox.setChecked(ViewProfileDao.checkUserForFavorite(user.getId(), tempUser.getId()));

checkBox.setOnCheckedChangeListener((CompoundButton buttonView, boolean isChecked) -> {

if (isChecked) {

ViewProfileDao.addFavoriteUser(user.getId(), tempUser.getId());

} else {

ViewProfileDao.deleteFavoriteUser(user.getId(), tempUser.getId());

}

});

if(user.getId() == tempUser.getId()) {

checkBox.setVisibility(View.INVISIBLE);

}

linearLayout.addView(checkBox);

linearLayout.addView(tv);

llFoundUsers.addView(linearLayout);

}

}

/\*\*

\* Відкриває профіль вибраного користувача

\*

\* @param user користувач, профіль якого потрібно відкрити

\* @see User

\* @see Leader

\* @see Worker

\* @see Intent

\*/

private void openProfile(User user) {

if (user instanceof Leader) {

Leader tempLeader = (Leader) user;

Intent intent = new Intent(SearchUsersActivity.this, ProfileViewerActivity.class);

intent.putExtra(Leader.class.getSimpleName(), tempLeader);

startActivity(intent);

} else {

Worker tempWorker = (Worker) user;

Intent intent = new Intent(SearchUsersActivity.this, ProfileViewerActivity.class);

intent.putExtra(Worker.class.getSimpleName(), tempWorker);

startActivity(intent);

}

}

}

**User**

package com.rosokha.todolist.users;

import java.io.Serializable;

/\*\*

\* Представляє собою коритсувача

\*/

public abstract class User implements Serializable {

/\*\*

\* Ідентифікатор користувача

\*/

private int id = 0;

/\*\*

\* ПІБ користувача

\*/

private String name;

/\*\*

\* Логін користувача

\*/

private String login;

/\*\*

\* Конструктор без параметрів

\*/

public User() {

}

/\*\*

\* Конструктор з параметрами

\*

\* @param id ідентифікатор користувача

\* @param name ПІБ користувача

\* @param login логін користувача

\*/

public User(int id, String name, String login) {

this.id = id;

this.name = name;

this.login = login;

}

/\*\*

\* Оновлює логін керівника

\*

\* @param newLogin новий логін користувача

\*/

public abstract void updateLogin(String newLogin);

/\*\*

\* Оновлює ПІБ керівника

\*

\* @param newUserName новий ПІБ користувача

\*/

public abstract void updateUserName(String newUserName);

/\*\*

\* Getter для поля id

\*

\* @return ідентифікар користувача

\*/

public int getId() {

return id;

}

/\*\*

\* Setter для поля id

\*

\* @param id ідентифікатор користувача

\*/

public void setId(int id) {

this.id = id;

}

/\*\*

\* Getter для поля name

\*

\* @return ПІБ користувача

\*/

public String getName() {

return name;

}

/\*\*

\* Setter для поля name

\*

\* @param name ПІБ користувача

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* Getter для поля login

\*

\* @return логін користувача

\*/

public String getLogin() {

return login;

}

/\*\*

\* Setter для поля login

\*

\* @param login логін користувача

\*/

public void setLogin(String login) {

this.login = login;

}

}

**Worker**

package com.rosokha.todolist.users;

import android.database.SQLException;

import com.rosokha.todolist.dao.UserDao;

/\*\*

\* Клас Worker представляє собою працівника, який отримує від керівника завдання і виконує їх

\*

\* @see User

\*/

public class Worker extends User {

/\*\*

\* Оновлює логін працівника

\*

\* @param newLogin новий логін

\* @throws SQLException помилка під час оновлення логіну

\* @see UserDao

\* @see UserDao#updateLogin(User, String)

\*/

@Override

public void updateLogin(String newLogin) throws SQLException {

UserDao.updateLogin(this, newLogin);

setLogin(newLogin);

}

/\*\*

\* Оновлює ПІБ працівника

\*

\* @param newUserName новий ПІБ

\* @throws SQLException помилка під час оновлення ПІБ

\* @see UserDao

\* @see UserDao#updateName(User, String)

\*/

@Override

public void updateUserName(String newUserName) throws SQLException {

UserDao.updateName(this, newUserName);

setName(newUserName);

}

}

**NotificationService**

package com.rosokha.todolist.notifications;

import android.app.Notification;

import android.app.PendingIntent;

import android.content.Context;

import android.content.Intent;

import android.database.SQLException;

import android.os.Build;

import androidx.core.app.NotificationCompat;

import androidx.core.app.NotificationManagerCompat;

import com.rosokha.todolist.R;

import com.rosokha.todolist.authorization.MainActivity;

import com.rosokha.todolist.dao.LoaderOfTasksDao;

import com.rosokha.todolist.dao.UserDao;

import com.rosokha.todolist.task.Task;

import com.rosokha.todolist.users.User;

import java.time.LocalDateTime;

import java.util.List;

/\*\*

\* Являє собою сервіс перевірки повідомлень для користувача

\*

\* @see Thread

\*/

public class NotificationService extends Thread {

// Ідентифікатор повідомлення

private static int NOTIFY\_ID = 1;

// Ідентифікатор канала

private static String CHANNEL\_ID = "To Do List channel";

/\*\*

\* Користувач, для якого перевіряються повідомлення

\*/

User user;

/\*\*

\* Вікно, з якого викликаний клас

\*/

Context context;

/\*\*

\* Конструктор

\*

\* @param user користувач

\* @param context вікно, з якого викликаний клас

\*/

public NotificationService(User user, Context context){

this.context = context;

this.user = user;

}

/\*\*

\* Викликається при початку роботи потоку

\*/

@Override

public void run() {

checkingNotifications();

if(Thread.currentThread().isInterrupted()){

return;

}

super.run();

}

/\*\*

\* Перевіряє на наявність повідомлень кожні 5 секунд

\*/

private void checkingNotifications() {

List<Task> tasks;

while (true) {

if(Thread.currentThread().isInterrupted()){

return;

}

try {

tasks = LoaderOfTasksDao.getListOfActiveTasks(user.getId());

} catch (SQLException ex) {

return;

}

for (Task task : tasks) {

if (!task.isNotificationAboutCreatingTask()) {

sendNotification("Нове завдання", "У вас нове завдання:\n" + task.getName());

UserDao.markNotificationAboutCreatingTask(user, task, true);

task.setNotificationAboutCreatingTask(true);

}

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

if (!task.isNotificationTaskCompletionWarning() && task.getEndDateTime().minusDays(1).isBefore(LocalDateTime.now())) {

sendNotification("Попередження", "Залишилося менше 1 дня до закінчення завдання:\n" + task.getName());

UserDao.markNotificationTaskCompletionWarning(user, task, true);

task.setNotificationTaskCompletionWarning(true);

}

}

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

if (!task.isNotificationLateCompletionOfTask() && LocalDateTime.now().isAfter(task.getEndDateTime())) {

sendNotification("Завдання не виконано", "Завдання:\n" + task.getName() + "\nне виконане вчасно");

UserDao.markNotificationLateCompletionOfTask(user, task, true);

task.setNotificationLateCompletionOfTask(true);

}

}

}

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

Thread.currentThread().interrupt(); //відновлення статусу перерваний\*

e.printStackTrace();

}

}

}

/\*\*

\* Відправляє повідомлення

\*

\* @param title заголовок повідомлення

\* @param content зміст повідомлення

\* @see NotificationCompat

\* @see NotificationManagerCompat

\* @see NotificationManagerCompat#notify(int, Notification)

\*/

private void sendNotification(String title, String content) {

Intent notificationIntent = new Intent(context, MainActivity.class);

notificationIntent.setFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP | Intent.FLAG\_ACTIVITY\_SINGLE\_TOP | Intent.FLAG\_ACTIVITY\_NEW\_TASK);

PendingIntent contentIntent = PendingIntent.getActivity(context,

0, notificationIntent, PendingIntent.FLAG\_CANCEL\_CURRENT);

NotificationCompat.Builder builder =

new NotificationCompat.Builder(context, CHANNEL\_ID)

.setSmallIcon(R.drawable.logo)

.setContentTitle(title)

.setContentIntent(contentIntent)

.setAutoCancel(true)

.setStyle(new NotificationCompat.BigTextStyle().bigText(content));

NotificationManagerCompat notificationManager =

NotificationManagerCompat.from(context);

notificationManager.notify(NOTIFY\_ID++, builder.build());

}

}