МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРИВОРІЗЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Фізико-математичний факультет

Кафедра інформатики та прикладної математики

«Допущено до захисту»

Завідувач кафедри

_____ Соловйов В.М.

«___» _____ 2021 р.

Реєстраційний № _____

«___» _____ 2021 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВИЯВЛЕННЯ МОЖЛИВИХ ФАЛЬСИФІКАЦІЙ ПІД ЧАС ЗБОРУ ПІДПИСІВ У ПІДТРИМКУ ЕЛЕКТРОННИХ ПЕТИЦІЙ

Кваліфікаційна робота студента групи

Ім-16

ступінь вищої освіти «магістр»

спеціальності

014.09 Середня освіта (інформатика)

Коваленка Івана Ігоровича

Керівник: доц., канд. фіз.-мат.наук

Мерзликін Павло Володимирович

Оцінка:
Національна шкала _____
Шкала ECTS ____ Кількість балів _____
Голова ЕК_____ _____
Члени ЕК _____ _____
_____ _____
_____ _____
_____ _____

Кривий Ріг – 2021

ЗАПЕВНЕННЯ

Я, Коваленко Іван Ігорович, розумію і підтримую політику Криворізького державного педагогічного університету з академічної доброчесності. Запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають покликання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Криворізького державного педагогічного університету ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

ЗМІСТ

ВСТУП

У сучасному світі дуже помітний вплив громадян на політичні процеси й розвиток держав, позаяк люди безпосередньо беруть участь у політиці й голос кожного враховується при ухваленні рішень на державному рівні. Якщо раніше збір підписів був досить трудомістким процесом, зараз це відбувається і в Інтернеті: кожен може зайти на сайт офіційного інтернет-представництва Президента України, Верховної ради або органів місцевого самоврядування й віддати свій голос за ту чи іншу петицію. Ця процедура має цілком реальні юридичні наслідки, тому стає цікаво, як проходить збір голосів за активними петиціями та чи не відбувається маніпуляцій чи фальсифікацій. Щоб проаналізувати динаміку набору голосів за днями, зручно мати мобільний додаток, який дозволить кожному охочому побачити статистику й зробити власні висновки.

Смартфони з кожним роком стають доступнішими для людей і переважно ці телефони використовують операційну систему Android. Виходячи з цього краще розробляти додаток мовою Java, оскільки вона активно використовується для створення мобільних додатків під операційну систему Android. Перевага мови Java в розробці мобільних додатків над іншими мовами полягає в тому, що API Android дуже схожий на API мови Java, і Android підтримує якщо не всі доступні в J2SE SDK класи, то точно найбільш важливі. Ще одна перевага – це можливість використовувати для розробки під Android ті ж інструменти, що і для Java. Наприклад, IDE Eclipse. Google надає для Eclipse плагін для розробки додатків Android.

Сутність розв'язуваної задачі – змоделювати та реалізувати застосунок, який виявить можливі стрибки в голосуванні й покаже, за який проміжок та до яких петицій могла бути застосована «накрутка» голосів.

Користувач мобільного клієнта системи громадського моніторингу електронних петицій в будь-який час може подивитися, як проходить голосування й самостійно оцінити активність за петиціями, представленими на офіційному сайті петицій Президента України і Верховної Ради України.

Об’єкт дослідження: громадський моніторинг процесу збору підписів під електронними петиціями.

Предмет дослідження: програмне виявлення можливих фальсифікацій під час збору підписів у підтримку електронних петицій.

Метою магістерської роботи є написання мобільного застосунку – клієнта системи громадського моніторингу електронних петицій.

Для досягнення мети слід розв’язати такі **задачі:**

- проаналізувати чинну нормативну базу, що стосується електронних петицій;
- порівняти наявні інструменти громадського контролю збору підписів під електронними петиціями;
- розглянути загальну структуру мобільних застосунків операційної системи Android;
- висунути вимоги до мобільного клієнту системи моніторингу збору підписів;
- обґрунтувати вибір інструментів розробки;
- спроєктувати базу даних;
- розробити мобільний клієнт, створити алгоритми аналізу онлайн-голосувань на сайті електронних петицій;
- створити інструкцію користувача.

Новизна роботи полягає в тому, що вперше створено мобільний застосунок для моніторингу збору голосів під електронними петиціями, що відповідає чинному законодавству України щодо електронних петицій.

Практична значимість роботи полягає в тому, що в її рамках створено завершений мобільний застосунок, що може бути використаний для фіксації можливих порушень під час збору підписів під електронними петиціями.

Структура роботи. Пояснювальна записка складається з вступу, трьох розділів, висновків, списку використаних джерел з 27 найменувань та додатків.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ Й ПОСТАНОВКА ЗАДАЧІ

1.1. Електронна петиція як аспект публічної влади, її юридичний статус та життєвий цикл

Згідно [1], «Електронна петиція — це особлива форма колективного звернення громадян до Президента України, Верховної Ради України, Кабінету Міністрів України, органу місцевого самоврядування.

Електронна петиція є одним з інструментів електронної демократії. Українське законодавство визначає електронну петицію як колективне звернення в електронній формі до суб'єкту владних повноважень у вигляді тексту скарги (протесту) та/або пропозиції. Суб'єкт владних повноважень має на електронну петицію публічно оголосити свою позицію стосовно згоди чи незгоди по суті петиції, інформувати про аргументи у разі незгоди та організовати спільну з авторами та їх прихильниками роботу з розроблення та втілення плану реалізації петиції у разі згоди.

Електронна петиція подається та розглядається в порядку, передбаченому статтею 23-1 Закону України "Про звернення громадян".

З електронними петиціями громадяни можуть звернутися до Президента України, Верховної Ради України, Кабінету Міністрів України, органу місцевого самоврядування через офіційний веб-сайт органу, якому вона адресована, або веб-сайт громадського об'єднання, яке здійснює збір підписів на підтримку електронної петиції.

Електронна петиція, адресована відповідно Президенту України, Верховній Раді України, Кабінету Міністрів України, розглядається у особливому порядку за умови збору на її підтримку не менш як 25000 підписів громадян протягом не більше трьох місяців з дня оприлюднення петиції.

Вимоги до кількості підписів громадян на підтримку електронної петиції до органу місцевого самоврядування та строку збору підписів визначаються статутом територіальної громади.

Особливий порядок розгляду петицій, які набрали належну кількість підписів, визначається актами адресатів петиції.

Якщо петиція не набрала необхідної кількості підписів, її розглядають в порядку, встановленому законом для звичайних звернень громадян.

28 серпня 2015 року Президент України видав Указ "Про Порядок розгляду електронної петиції, адресованої Президентові України" [2], згідно з яким вже наступного дня на офіційному сайті президента було запущено сервер петицій. Першою петицією, що досягла необхідної кількості голосів, стала петиція від Української асоціації власників зброї, в якій вимагається законодавчо затвердити право громадян України на захист. Під час обробки інформації про підписи було виявлено так званих "ботів", які намагалися "накручувати" кількість підписів, але заступник глави адміністрації Президента України Дмитро Шимків повідомив, що ці маніпуляції не зашкодять розгляду петиції »[1].

Розглянемо, як працює сервіс електронних петицй до Президента України на основі інформації з офіційного вебсайту[3].

«Громадяни можуть звернутися до Президента України з електронними петиціями через спеціальний розділ веб-сайту Офіційного інтернет – представництва Президента України, доступний за адресою petition.president.gov.ua. Для того, щоб підтримати електронну петицію або створити нову, потрібно зареєструватися на сервісі <…> Електронна петиція, яка відповідає встановленим вимогам, оприлюднюється у розділі електронних петицій на сторінці «Триває збір підписів» протягом двох робочих днів з дня її надсилання автором (ініціатором). У разі невідповідності електронної петиції встановленим вимогам оприлюднення такої петиції не здійснюється, про що автор (ініціатор) електронної петиції отримує повідомлення на вказану ним адресу електронної пошти.

Електронна петиція, адресована Президентові України, розглядається за умови збору на її підтримку не менш як 25 тисяч підписів громадян протягом не більше трьох місяців з дня оприлюднення петиції.

Електронна петиція, яка в установлений строк не набрала необхідної кількості голосів на її підтримку, після завершення строку збору підписів на її підтримку розглядається як звернення громадян відповідно до Закону України «Про звернення громадян». Інформація про початок розгляду електронної петиції, яка в установлений строк набрала необхідну кількість голосів на її підтримку, оприлюднюється у розділі електронних петицій на сторінці «На розгляді» не пізніш як через три робочі дні після набрання необхідної кількості підписів на її підтримку. Розгляд електронної петиції здійснюється невідкладно, але не пізніше десяти робочих днів від дня оприлюднення інформації про початок її розгляду.

Про підтримку або не підтримку електронної петиції, адресованої Президентові України, публічно оголошується Президентом України на веб-сайті Офіційного інтернет-представництва Президента України. Відповідь на електронну петицію не пізніше наступного робочого дня після закінчення її розгляду оприлюднюється у розділі електронних петицій на сторінці «З відповідями», а також надсилається у письмовому вигляді автору (ініціатору) електронної петиції.

У відповіді на електронну петицію повідомляється про результати розгляду порушених у ній питань із відповідним обґрунтуванням. У разі визнання за доцільне викладені в електронній петиції пропозиції можуть бути реалізовані Президентом України шляхом прийняття з питань, віднесених до його компетенції, відповідного рішення. Главою держави за результатами розгляду електронної петиції можуть розглядатися та вноситися в установленому порядку на розгляд Верховної Ради України законопроекти, спрямовані на вирішення порушених у петиції питань <…>» [3].

1.2. Наявні інструменти громадського контролю електронних голосувань

Вебсайт «Громадський простір» [4] зазначає, що електронна петиція є одним з інструментів електронної демократії. Відповідно до Закону України «Про звернення громадян», петиція – це особлива форма колективного звернення громадян, викладені в письмовій або усній формі пропозиції (зауваження), заяви (клопотання) і скарги.

Електроні петиції створюють можливість створення позитивного діалогу між громадянами та владою. Вони дають громадянам можливість висловити власні ідеї, об’єднатись з однодумцями та забов’язати владу оперативно розглянути пропозиції населення. В той самий час дає владі можливість дізнатись в онлайн режимі про потреби громадян та першочергово виконувати пріоритетні задачі.

Наразі громадяни можуть подати е-петиції на офіційних веб-сайтах: Президента України, Верховної Ради, Кабінету Міністрів України та органів місцевого самоврядування. А також – через онлайн - платформи такі як: «Єдина система місцевих петицій» (152 громади), «Розумне місто» (101 громада), «Мій Голос» [4].

Необхідність у сервісі, вебсайті або мобільному за стосунку, здатному проаналізувати або надати матеріал для подальшого аналізу голосувань у підтримку електронних петицій і виявлення можливих фальсифікацій однозначно є. Прикладом може служити зафіксована раніше й висвітлена в статті порталу новин «Українська правда» [5] фальсифікація на сайті Криворізької міськради, де були розміщені дві петиції про розформування Криворізької муніципальної гвардії.

Так, згідно [5], «коли дві петиції проти Муніципальної гвардії наблизилися до позначки 500 підписів в підтримку, сайт закрили на технічні роботи. Після майже місячної перерви, сайт онлайн петицій до місцевої влади відкрився. Майже одразу на ньому з’явилася петиція "про підтримку

правопорядку у Кривому Розі". І вже протягом лише 7 годин вона зібрала необхідну кількість підписів – 501<...>

Місцеві активісти виявили, що значну частину підписів під цією онлайн петицією ставили працівники шкіл, комунальних підприємств, бюджетних установ та райрад. Список з 165-ти підписантів і їхні місця роботи опублікувала у себе на ФБ-сторінці Ольга Перетятко.

Громадський активіст Євген Василенко провів аналіз усіх петицій з сайту міськради. І виявив, що 487 з 501 підписантів "за муніципальну гвардію" вперше з'явилися на сайті саме для підпису цієї петиції. До цього вони ні чим не цікавилися, ніде не світилися, нічого не підписували, каже він у розмові з "Українською правдою": "Їх або попросили, або примусили. Як у нас робиться: лагідно попросили, що треба зробити. Я вважаю, що це адміністративний ресурс"» [5].

Також наявність в інтернеті вебсайту nakrutka.net [6], який надає послуги з накрутки онлайн-голосувань різного роду, і зокрема накрутки голосв на сайті петіцій до Президента України, підтверджує актуальність створення описаного далі в цій роботі програмного забезпечення і необхідність у додатковій перевірці та аналізі голосів.

Приклад організації, що проаналізувала збір онлайн голосів з петицій це ГО «Електронна демократія» [27]. Наступного дня після запуску сайту е-петицій до Президента України ними було запущено постійний автоматичний моніторинг petition.president.gov.ua щодо кількості петицій та підписів за них. На прикладі першої петиції, що набрала необхідні для розгляду 25000 голосів, організація зробила аналітику, спробувавши відповісти на питання – "чи можна за графіком виявити накрутку?". Це була Петиція № 40 "про право на захист". Приклад моніторінгу наведенно на рис. 1.1.

Рис 1.1. Графік даних голосування за петицією №40 [27].

Аналізуючи отриманий графік моніторингу голосування та списку тих, хто підписав петицію, організація зробила певні висновки.

На їхню думку питання можливості та наявності накруток необхідно розділити на дві частини: по-перше, принципова можливість одиничних накруток і по-друге наявність значних накруток, які впливають на подолання планки в 25 тисяч підписів. Враховуючи механізм реєстрації та підписування практично виключається автоматична накрутка, без участі людини. В результаті можна створити десятки несправжніх підписів, але не десятки тисяч, які необхідні для проходження петиції.

Ще одним способом виявлення автоматичних накруток є пошук характерних елементів на графіці – «різкий стрибок», «зуби пили» (через вмикання/вимикання автоматичного підписання) наявність виражених елементів, що повторюються, різка і повторювана зміна темпу збору підписів не залежить від часу доби.

Також додано зауваження щодо терміну збору підписів: згідно з вимогами закону на збір підписів виділено три місяці. Тому, якщо після перевірки та “чистки” списку залишиться менше 25 тисяч підписів, то остаточне рішення про розгляд або не розгляд петиції можна буде ухвалити лише після закінчення цього терміну.

Запропоновано перевірку через інші джерела: одним із способів перевірки реальної популярності петиції може стати реакція на неї у соціальних мережах (статистика за постами, що містять посилання на сторінку петиції). При цьому підтримку таких постів можна розглядати як непряме підтвердження підтримки петиції.. Що стосується розглянутої петиції, загалом активність у соціальній мережі корелює із загальною кількістю тих, хто підписався [27].

Робота проведена ДО «Електронна демократія» та отримані висновки показують можливості для аналізу, які дає моніторинг голосування. Не всі їх висновки зараз актуальні, оскільки спосіб реєстрації та голосування на сайті змінився. Але підозрілі підписання петицій продовжуються, і глибший аналіз голосування дозволить вивити можливі накрутки підписів.

Цікаву інформацію надали в ГО «Ліга інтернів» [28], яка «вивчає електронні петиції до Верховної Ради України за підтримки Програми USAID РАДА, що виконується Фондом Східна Європа, з березня 2017 року. Протягом цього часу здійснювався моніторинг е-петицій, а також було підготовлено низку інформаційних та навчальних матеріалів» [28]. За їхніми даними «У 2017 році було подано запит на реєстрацію 592 е-петицій, з яких тільки 357 пройшли модерацію та з’явилися на веб-порталі Верховної Ради України. Протягом двох попередніх років ці показники були такими: у 2015 році – 616 та 246, у 2016 році – 544 та 277 відповідно.

За тематикою третій рік поспіль лідирують е-петиції, які стосуються діяльності Верховної Ради України, Президента України, Кабінету Міністрів України (таких щонайменше 144 з 357 оприлюднених) <…>

У 2017 році кількість зареєстрованих користувачів сягнула відмітки 128 902, з яких понад 100 000 – активні (105 123), що приблизно в три рази більше порівняно з 2015 роком (37 252 та 32 655 відповідно)» [28].

Аналізуючи діяльність спільноти ГО «Ліга інтернів», що займаєтсся збіром статистичних данних стосовно е-петицій, можна виділити серйозних підхід до збору даних і обсяг оброблюваної інофмації. Завдання, яке ставить

перед собою організація це аналіз роботи самого механізму онлайн голосвання, його ефективніть і популярність. Виявленням можливих накруток на сайті вони не займаються.

Також в інтернеті знайшлася статистика за е-петиціями, подана аналітічнім центром «Комунікація змін» [7]. «Статистика показує, що українці досить активно користуються своїм правом впливати на владу та стабільно подають петиції з метою вирішення нагальних проблем у державі. Питання в тому, яку роль у цьому демократичному процесі відіграє Президент?

Для того, щоб картина із річницею функціонування е-петицій була повною, варто розглянути відповіді Президента України на питання, поставлені у петиціях, які успішно подолали необхідний бар'єр для їх обов'язкового розгляду. Приклад наведенно на рис. 1.2.

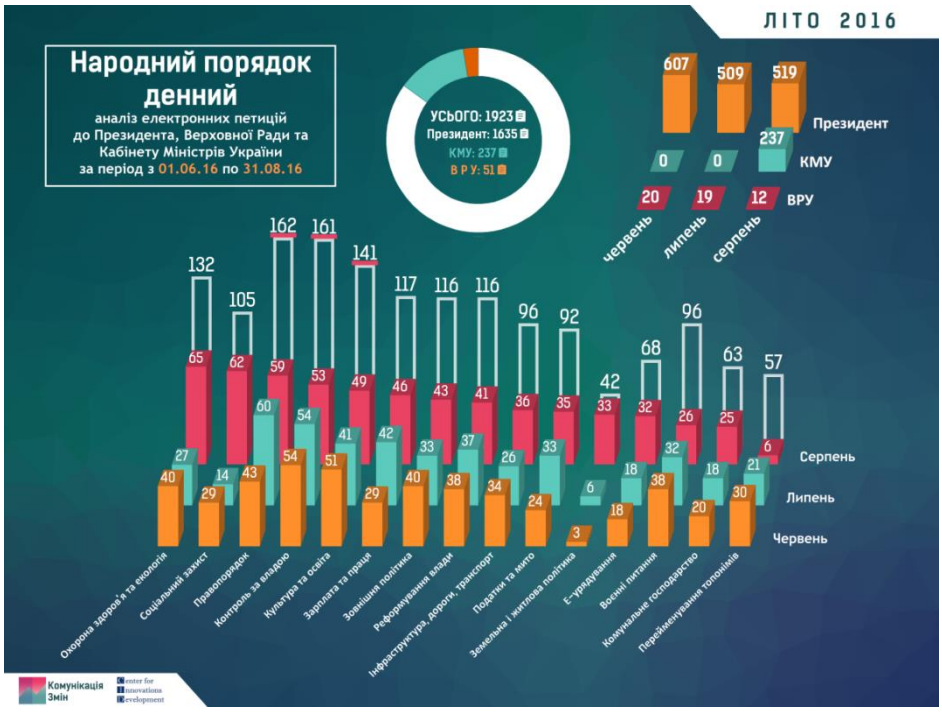

Рис 1.2. Аналіз електроних петицій за період з 01.06.16 по 31.08.16.

Відповідь на питання чим є петиції – ефективним інструментом демократії чи її імітацією – є неоднозначною. Чи може кожен українець, а навіть і не українець подати е-петицію до Президента України, Верховної Ради України, Кабінету Міністрів, органів місцевого самоврядування? –

Беззаперечно так і це є великою перемогою, що добута у двобої бюрократів і активістів, переможцями якого став український народ» [7].

1.3. Загальна характеристика Android-застосунку

Згідно [8], «Android – операційна система на основі ядра Linux з інтерфейсом користувача на основі прямого маніпулювання, призначена в першу чергу для сенсорних мобільних пристроїв, таких як смартфони й планшетні комп'ютери. Операційна система використовує сенсорне введення. З 2011 року в Android має найбільшу базу встановленого обладнання з-понад усіх мобільних ОС станом на 2013 рік. Пристрої які використовують Android, продають в світі більше ніж Windows, і пристроїв Mac OS, разом узятих. Станом на липень 2013 року Google Play магазин мав більш 1 млн. опублікованих програм і більше 50 мільярдів завантажених додатків. Станом на квітень-травень 2013 року встановлено, що 71% розробників мобільних додатків програмують для Android» [8].

Згідно [9], «Android додатки пишуться на Java. В Android інструменти SDK компілюють всі коди програми з будь-якими даними та ресурсами в файли APK: Android-пакет, який є архівним файлом з «.apk» суфіксом. Один APK файл містить весь Android застосунок і файл для декларацій пристроїв, які потребує програма.

Після встановлення на пристрої, кожен застосунок «живе» у своєму власному ізольованому програмному середовищі безпеки:

- Операційна система Android є багатокористувацькою системою Linux, в якій кожен застосунок є окремим користувачем.
- За замовчуванням система привласнює кожному додатку ID (унікальний ідентифікатор) користувача Linux (ID використовується тільки системою та не видимий для додатку). Система встановлює права доступу для всіх файлів, так що тільки даний ідентифікатор, наданий додатку може отримати до них доступ.

- Кожен процес має свою віртуальну машину та є ізольованим від інших програм. За замовчуванням кожен застосунок запускається у своєму процесі Linux.
- Android запускає процес, коли будь-який з компонентів додатку, необхідний для виконання, має бути запущений. Потім знищує процес, якщо він більше не потрібен, чи то коли система повинна відновити пам'ять для інших додатків.

Таким чином, Android система реалізує принципи найменших привілеїв. Тобто, кожен застосунок, за замовчуванням, має доступ тільки до компонентів, які він вимагає, але не більше того. Це створює дуже безпечне середовище, в якому застосунок не може отримати доступ до частини системи, до якої не дозволено звертатись.

Тим не менш, є шляхи для обміну даними з іншими додатками та для доступу до системних послуг:

- Можна розширити Linux Id між двума додатками, щоб вони були в змозі отримати доступ до файлів один одного. Щоб заощадити системні ресурси, додатки з одним і тим же ідентифікатором можуть запускатись в одному процесі й ділити між собою одну й ту ж віртуальну машину.
- Додаток може запитати дозвіл на доступ до персональних даних пристрою, таких як контакти користувача, SMS повідомлення, розділ монтування (SD карта), камера, Bluetooth і багато іншого. Всі дозволи програмі мають бути надані користувачем під час встановлення.

Компоненти додатку є основними будівельними блоками Android програми. Кожен компонент є іншою точкою, через яку система може виконувати програму. Не всі компоненти є фактичними точками входу для користувача, а деякі навіть залежать один від одного, але кожен з них існує

як власне вікно й відіграє особливу роль – кожен з них є унікальним будівельним блоком, який допомагає визначити загальну поведінку вашої програми.

Є чотири різних типи компонентів програми. Кожен тип слугує задля визначеної мети і має яскраво виражений життєвий цикл, який визначає, як компонент створюється й завершується.

Є чотири типи компонентів додатків:

- Activity(діяльність) – являє собою один екран з призначеним для користувача інтерфейсом. Наприклад, електронна пошта може мати один вид Activity , який показує список нових повідомлень електронної пошти, другий вид Activity – щоб скласти лист, та іншу Activity для читання електронної пошти.
- Служби (Services) - являє собою компонент, який працює у фоновому режимі, щоб виконати тривалі операції або роботи для віддалених процесів. Служби не забезпечують користувальницький інтерфейс. Наприклад, служба може відтворювати музику у фоновому режимі, поки користувач знаходиться в іншому додатку, або може отримати дані мережою, не блокуючи взаємодію користувача з Activity. Ще один компонент, наприклад, Activity, може запустити службу і працювати в звязці.
- Контент-провайдери (Content providers) – управляє загальним набором даних додатку. Ви можете зберігати дані у файловій системі, базі даних SQLite, в Інтернеті, або будь-якому місці, до якого додаток може отримати доступ. Через контент-провайдер, інші додатки можуть запитувати або навіть змінювати дані (якщо контент-провайдер це дозволяє). Наприклад, Android система забезпечує контент-провайдер, який управляє інформацією про контакти користувача.
- Broadcast receivers – це компонент, який відповідає за загальносистемні широкомовні анонси. Багато трансляцій відбуваються з системи,

наприклад, оголошення, що екран вимкнувся, батарея розряджена. Додатки можуть також ініціювати трансляції, наприклад, щоб інші програми знали про дані, які були завантажені в пристрій для використання. Хоча широкомовні приймачі не відображають для користувача інтерфейс, вони можуть створити повідомлення, щоб попередити користувача, коли відбувається широкомовна подія. Однак, найчастіше, широкомовний приймач є просто "шлюзом" для інших компонентів і має за мету виконати мінімум роботи.

Унікальною особливістю архітектури Android є те, що будь-яка програма може почати компонент іншої програми. Тоді ваш додаток може використовувати його, щоб зробити знімок, замість того щоб "винаходити велосипед". Вам не потрібно вмикати або навіть посилатись у вихідному коді на додаток камери. Замість цього ви можете просто почати свою діяльність у Activity камери, яка захоплює фото. Після завершення фото можна навіть повернутись у ваш додаток, щоб ви могли використовувати результат(в даному випадку знімок). Для користувача здається, ніби камера є частиною вашої програми.

Коли система запускає компонент, починається процес для цього додатка (якщо він ще не запущений) і створює екземпляри класів, необхідних для компонента. Наприклад, якщо ваш додаток починає свою діяльність в додатоку камери, яка захоплює фото, то діяльність виконується в процесі, котрому належить додаток камери, а не в процесі вашої програми. Тому, на відміну від програм у інших операційних системах, програми Android не мають єдиної точки входу (наприклад немає основої функції).

Оскільки система виконує кожен додаток в окремому процесі з правами доступу до файлів, які обмежують доступ до інших додатків, ваш додаток не може безпосередньо активувати компонент з іншої програми. Але це може система Android.

Так, щоб активувати компонент в іншому додатку, необхідно надіслати повідомлення до системи, яка визначає свій намір стартувати деякий компонент. Потім система активує компонент для користувача.

Android-додаток складається з більш, ніж просто коду - це вимагає ресурсів, які відокремлені від вихідного коду, таких як зображення, аудіо файли і все, що відноситься до візуального сприйняття програми.

Використання ресурсів програми дозволяє легко оновлювати різні характеристики додатку без модифікації коду і, надаючи набори альтернативних ресурсів, дозволяє оптимізувати ваш додаток для різних конфігурацій пристроїв (таких, як різні мови і розміри екрану).

Для кожного ресурсу, який включається до проекту, SDK визначає унікальний цілочисельний ідентифікатор, який можна використовувати для посилання на ресурс в коді програми або з інших джерел, визначених у XML. Наприклад, якщо додаток містить файл зображення з ім'ям logo.png , то SDK генерують ID ресурсу з ім'ям R.drawable.logo, який можна використовувати в певному інтерфейсі в якості посилання на зображення.

Одним з найбільш важливих аспектів забезпечення ресурсів окремо від вихідного коду є можливість забезпечити певний набір альтернативних ресурсів для використання їх для різних конфігурацій пристроїв.

Android підтримує безліч різних наборів для альтернативних ресурсів. Класифікатором є короткий рядок, що розробник включив в ім'я каталогів ресурсів, щоб визначити конфігурацію пристрою, для якого ці ресурси повинні бути використані. В якості іншого прикладу, можна часто створювати різні макети для Activity, залежно від орієнтації екрану пристрою і розміру. Щоб змінити макет залежно від орієнтації, можна визначити дві різні схеми і застосувати відповідний класифікатор для ім'я каталогу кожного макету. Потім система автоматично застосує відповідний макет залежно від поточної орієнтації пристрою» [9].

1.4. Постановка задачі

Виходячи з аналізу переваг і недоліків наявних підходів до виявлення фальсифікацій під час збору підписів у підтримку електронних петицій, було висунуто вимоги до програмного забезпечення.

Основне завдання: створити мобільний клієнт, використовуючи мову програмування Java та реалізувати аналіз онлайн голосувань на вебсайтах електронних петицій Президента України та Верховної Ради України.

З порівняння наявних мобільних застосунків видно, що інструменти Java дозволяють створювати приголомшливі речі, що робить роботу з Android більш захопливою, а використання різних бібліотек дозволяє робити складні речі просто.

Мобільний клієнт повинен бути схожий на журнал, в якому кожен може подивитися список з петицій і вибрати якусь для перегляду статистики, а також порівняти списки підписантів за різними петиціями за конкретні дні, в які, на погляд користувача, була можлива накрутка або була велика активність (застосунок буде виділяти таки петиції, де кількість голосів за день привищує заданий показник). Для такого порівняння повинно існувати окреме вікно, де можна переглянути подпісантів, які віддали свій голос за 2 або більше петицій в обрані користувачем дати. Застосунок буде надавать можливість перейти на сторінку петиції на сайті, де вже можна проголосувати самому, а також знайти потрібну петицію за назвою або номером. Дані з петицій будуть оновлюваться послідовно, також будуть завдання для запуску сканування петицій та голосів щодо них у певний час. Крім цього, буде можливість оновити інформацію про петицію в будь-який час за допомогою призначеної для цього кнопки. У застосунку будуть представлені петиції з різними статусами, до яких можна застосувати сортування. Після знаходження збігів серед підписантів буде передбачено можливість зробити експорт отриманого списку.

Висновки до розділу 1

Аналіз нормативної бази та проектів з моніторингу електронних голосувань показав, що електронні петиції відіграють значну роль у суспільному житті, однак в Україні моніторинг таких голосувань здійснюється зазвичай вручну, отже існує потреба в інструменті, що автоматизував би пошук підозрілої активності стосовно окремих петицій для привернення уваги суспільства до можливих фальсифікацій.

У процесі вивчення предметної області стало зрозуміло, що за допомогою Java можна робити як прості, так і складні застосунки, враховуючи навички програміста та його бажання, можна створити дуже привабливі речі. При розробці клієнту можна використовувати наявні бібліотеки. Вони дозволяють по-різному взаємодіяти з користувачем, та завдяки їм можливо максимально використовувати систему Android без занурення в програмування самої системи. Вбудовані в Android Studio функції відрізняються від стандартних в Java, але документація й опис до них зазвичай додаються і розібратися в них не складе великих труднощів.

РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ ГРОМАДСЬКОГО КОНТРОЛЮ ЗБОРУ ПІДПИСІВ У ПІДТРИМКУ ЕЛЕКТРОННИХ ПЕТИЦІЙ

2.1. Обґрунтування вибору інструментів розробки та архітектури системи

Щоб реалізувати задуманий клієнт, потрібно отримати дані з сайту і робити це щоразу при оновленні списку петицій, для отримання актуальної інформації. За це відповідає парсер, робота якого ґрунтується на бібліотеці jsoup [12]. Щоб отримані дані коректно відображалися в застосунку для декодування UTF-8 використовуватиметься бібліотека apache [14]. Інша стороння бібліотека, яка буде використовуватись в проекті – це MPAndroidChart [13]. Вона будує графік за даними, які збирає парсер. Для порівняння підписантів буде використовуватись бібліотека guava [16]. За всю фонову роботу (завдання) Android відповідатиме WorkManager [15].

Поза тим для досягнення остаточного результату будуть використані вбудовані функції й методи Android SDK. Для роботи з даними в ролі СУБД використовувтиметься SQLite.

Принципи дизайну класів програми.

Згідно [10], класи – це блоки, з яких будується застосунок Java. І якщо класи неякісно написані, то вони одного разу можуть привести до проблемної ситуації в процесі роботи програми. З іншого боку, добре розроблені і якісно написані класи можуть прискорити процес кодування і зменшити кількість помилок.

Існує п'ять основних принципів дизайну класів в об'єктно-орієнтованому проектуванні, які слід мати на увазі при написанні коду. Скорочено вони називаються S.O.L.I.D., вони є досить важливими елементами написання класів [10]. Тому при проектуванні й подальшій розробці програми увага приділялася саме цим методам. Нижче йде більш детальний опис цих методів.

Single Responsibility Principle (Принцип єдинчних обов'язків).

На кожен об'єкт має бути покладено один єдиний обов'язок. Тобто в проекті кожен клас написаний тільки для однієї мети. Модель класу являє собою одну функцію або дію. Це дає можливість вносити зміни в майбутньому, не боячись впливу змін на інші об'єкти. Кожен об'єкт має один обов'язок і цей обов'язок повністю ізольований у класі. Також усі його сервіси спрямовані виключно на забезпечення цього обов'язку. Об'єднання двох сутностей, що змінюються з різних причин і в різний час, вважається поганим проектним рішенням.

Open Closed Principle (Принцип відкритості/закритості).

Програмні сутності (класи, модулі, функції) відкриті для розширення, але закриті для зміни. Це означає, що класи розроблені таким чином, що при необхідності підлаштувати клас до даних конкретних умов застосування, досить розширити клас і перевизначити деякі функції. Таким чином, система гнучка й має можливість роботи в умовах, що змінюються без зміни вихідного коду. Це важливо, якщо інші розробники вирішать спроектувати іншу бажану поведінку в класу, щоб змінити логіку використовуваного класу або розширити клас і підлаштувати його для виконання конкретного завдання.

Liskov's Substitution Principle (принцип підстановки Барбари Лісков).

Об'єкти в програмі можуть бути замінені їх спадкоємцями без зміни властивостей програми. Цей принцип є варіацією принципу відкритості/закритості. Клас, розроблений на підставі базового класу шляхом розширення, працює в застосунку без збоїв. Тобто, якщо розширювати клас і використовувати його в застосунку, він не порушує роботу програми або створює фатальні помилки для всього додатку. Це реалізується, позаяк базовий клас робить строго одну справу і при його зміні може виникнути тільки одна проблема. Це може привести до деяких помилок в одній області, але вся програма не буде працювати неправильно.

Interface Segregation Principle (Принцип поділу інтерфейсу).

Клієнти не зобов'язані реалізовувати непотрібні методи, які вони не будуть використовувати. Складні інтерфейси розділені на більш дрібні й специфічні, щоб клієнти маленьких інтерфейсів знали тільки про методи, які необхідні їм у роботі. У підсумку, при зміні методу інтерфейсу не змінюються клієнти, які цей метод не використовують. Наприклад, в інтерфейсі SignatureDao, де прописані методи роботи з даними підписантів, не використовуються методи HistoryDao, хоч вони і обидва потрібні для роботи з таблицями.

Dependency Inversion Principle (Принцип інверсії залежностей).

Залежності всередині системи будуються на основі абстракцій. Модулі верхнього рівня не залежать від модулів нижнього рівня. Абстракції не залежить від деталей, при цьому деталі залежать від абстракцій. Застосунок створювався таким чином, що різні модулі автономні та з'єднуються один з одним за допомогою абстракції. Модулі надають доступ лише до абстракції, яка може використовуватися в іншому модулі.

2.2. Проектування бази даних

Одним з головних елементів проєктованої системи є база даних. У ній зберігається структурована інформація, яка заповнюється при скануванні сторінок сайту. Дані у базі даних зберігаються у таблицях, які мають відповідні поля.

Проєктування структури бази даних виконано на підставі аналізу предметної області, описаного в розділі 1.

Для створення таблиць бази даних застосунку, з сайтів збору електронних петицій до Президента України та Верховної Ради України була зібрана необхідна інформація, щоб наочно побачити яку інформацію збиратиме парсер при скануванні сторінок. Нижче будуть наведені описи таблиць DataLine, Petitions, Signature.

У кожній таблиці може існувати первинний ключ – поле або набір полів, що однозначно ідентифікують запис. Значення первинного ключа таблиці повинно бути унікальним, тобто в таблиці не повинно існувати двох чи більше записів з однаковими значеннями первинного ключа.

Опис створених таблиць:

Таблиця 2.1.

DataLine (голоси)

Поле	Опис	Тип даних
uuid	uuid петиции	TEXT
num	Номер петиції	TEXT
lastUpd	Останнє оновлення	TEXT
votes	Кількість голосів	INTEGER

Таблиця 2.2.

Petitions (петиція)

Поле	Опис	Тип даних
uuid	uuid петиции	TEXT
num	Номер петиції	INT
name	Назва петиції	TEXT
url	Посилання на петицію	TEXT
tag	Мітка	TEXT
site	Сайт	TEXT
status	Статус петиції	TEXT
publicDate	Дата публікації	TEXT
votes	Кількість голосів	INTEGER
leftDay	Залишок днів	INTEGER
signatories	Кількість підписантів	INTEGER
lastUpd	Останнє оновлення	TEXT
updateDate	Останнє оновлення голосів	TEXT
isActive	Статус активності	INTEGER
maxDelta	Максимальна зміна	INTEGER
author	Автор	TEXT
isWorker	Сканування голосів в даний момент в ручному режимі	INTEGER

Таблиця 2.3.

Signature (підписанти)

Поле	Опис	Тип даних
uuid	uuid петиции	TEXT
id	Позиція в списку	INTEGER
name	Підписант	TEXT
voteDate	Дата голосування	TEXT
num	Номер петиції	INTEGER

У результаті сканування сторінок сайту, на основі отриманих даних та порівняння подписантів за окремими петиціями, повинна бути створена таблиця, працювати з даними якої вже можуть журналісти або інші охочі проаналізувати активність петицій і стрибки в конкретні дні голосувань.

Висновки до розділу 2

Після аналізу наявних рішення для аналізу голосувань у підтримку електронних петицій і кращого ознайомлення з ситемою Android було розпочато проєктування майбутньої системи. У ході проєктування були знайдені рішення для реалізації поставлених завдань, у вигляді алгоритму створення архітектури додатку і застосування необхідних бібліотек Java, для розробки програми, з урахуванням тих бажань і цілей, реалізація яких дасть змогу отримати остаточний результат у потрібному вигляді.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО КЛІЄНТУ ДЛЯ ГРОМАДСЬКОГО КОНТРОЛЮ ЗБОРУ ПІДПИСІВ У ПІДТРИМКУ ЕЛЕКТРОННИХ ПЕТИЦІЙ

3.1. Особливості програмної реалізації

За замовчуванням, операційна система Android присвоює кожному застосунку унікальний ідентифікатор користувача. Після запуску застосунків Android, коєен із них виконується у своєму процесі, у своїй власній віртуальній машині [9].

Згідно [11], у міру необхідності, операційна система Android керує запуском і зупинкою процесів застосунків. Це означає, що всі застосунки Android працюють ізольовано один від одного, але, зрозуміло, можуть запитувати доступ до апаратних та інших системних ресурсів.

При встановленні або запуску програми Android, вона запитує права, необхідні для доступу до інтернету, телефонної книги або інших системних ресурсів. Користувач явно надає ці права, інакше в дії буде відмовлено. Всі ці права доступу описуються у файлі маніфесту застосунку Android. На відміну від Java, маніфест Android являє собою XML-файл, в якому перераховані всі компоненти програми та налаштування для них.

Чотири основних компоненти застосунку Android: Activity з фрагментами, сервіси, постачальники контенту й широкомовні приймачі (broadcast receivers). З них найчастіше зустрічаються Activity, відповідні окремій екранній формі застосунку Android. Наприклад, у грі для операційної системи Android може бути кілька екранів: для входу в систему, рекордів, інструкцій і екран самої гри. Кожен з цих елементів відповідає різним activity у застосунку.

Як і в Java, в ОС Android добре те, що вона виконує деякі завдання замість розробника, наприклад, створює об'єкти активностей. За організацію Activity відповідає клас System. Якщо потрібно запустити activity, достатньо викликати метод startActivity () з об'єктом Intent як параметр. У відповідь на

цей виклик, клас System або створить новий об'єкт Activity або повторно використовує старий.

Аналогічно збиранню сміття в мові Java, що відповідає за надзвичайно важливе завдання повторного використання пам'яті, Android керує запуском, зупинкою, створенням і знищенням застосунків. Може здатися, що він занадто сильно їх обмежує, але це не так. Android надає події життєвого циклу, які можна перевизначати для втручання в цей процес [11].

3.2. Проектування алгоритмів і структур даних

При запуску програми створюється головне activity застосунку MainActivity і підключається база даних AppDatabase. Далі створюється адаптер зв'язку з базою PetitionListAdapter для завантаження даних в PetitionViewModel через спостерігач застосунку. Дані відображаються в CardView та RecycleView.

Для взаємодії з користувачем реєструються обробники кнопок для побудови графіку й відкриття сторінки з петицією на сайті та кнопка збирання голосів. При натисканні на ці кнопки відкриваються Activity – LineActivity, WebActvity або запускається задача ручного сканування петиції (ManualWorker). Окрім цього задаються кнопки для сортування, фільтру та пошуку петицій. Також при старті застосунку створюються завдання на сканування петицій і голосів за ними (тільки за активними петиціями).

Завдання, які були створені за петиціями – перестворюються, тобто, при кожному запуску програми вони створюються і запускаються заново. Завдання на сканування голосів створюються при першому запуску і при наступних запусках не змінюються (інтервал сканування – раз на годину).

Також реалізована можливість на головному activity фільтрувати й сортувати дані та вибрати активний сайт для подальшого аналізу. При перемиканні між сайтами список петицій для порівняння не очищується, тобто є можливість порівнювати збіги й між ними.

Алгоритм поновлення виглядає так: парсер працює, посторінково аналізуючи сайт, поки на сторінці є дані. Коли буде зібрана інформація за сторінкою з петиціями, починається аналіз акивних петицій, окремо отримуються дані за голосами, зберігається список підписантів.

Оскільки застосунок створювався в Android Studio, опис алгоритмів і структури даних виконуватиметься із зазначенням назви файлів, позаяк вони створені в структурі самої Android Studio. Нижче будуть розглянути головні activity застосунку та фрагменти. Але спочатку треба зазначити, що обов'язковим є файл AndroidManifest – за ним збирається весь застосунок, цей файл створює Android Studio. У ньому знаходиться клас Permission. Він визначає, що має право робити додаток. INTERNET – дає право завантажувати з Інтернету, WRITE_EXTERNAL_STORAGE – дозволяє за стосунку виконувати запис у зовнішнє сховище, ACCESS_NETWORK_STATE – дозволяє застосунку отримувати доступ до інформації про мережі, RECEIVE_BOOT_COMPLETED – коли завантажена система, отримати повідомлення про її завантаження. WAKE_LOCK – розбудити пристрій, щоб процес не спав або екран не згасав. У цьому ж документі оголошується головна Activity – MainActivity, з якої починається завантаження проєкту.

Також тут прописуються інші activity типу LineActivity для побудови графіка, AboutActivity та WebActivity.

Використовувані бібліотеки підключаються у файлі boild.gradle, в ньому автоматично прописуються стандартні біліотеку від Google, а зовнішні додаються вручну.

Файли логічно розділені за папкам у структурі проекту, нижче будуть розглянуті не всі, а тільки ті, що несуть цільове навантаження проекту і розкривають логіку відображення й оновлення даних стосовно петицій та їх аналізу. Спочатку буде розглянуто все, що пов'язано з базою даних.

App

На самому початку App розширює клас Aplication, в якому прописаний калс App, який створюється під час запуску програми і є статичним. В ньому прописані теги для планувальника завдань, а також інтервал запуску завдань зі збору петицій та голосів. Тут же підключається база my.db. Далі запуск завдань через планувальник реалізований в Android WorkManager (за умови, що є інтернет, інакше завдання парсинга запускати немає сенсу). Це завдання на сканування петицій з різними статусами та отримання голосів за цими петиціями.

AppDataBase

Тут прописані entities: Petition, DataLine та Signature – це використовувані таблиці. Для перевірки актуальності версії бази використовується значення version = 2. Тут використовуються абстрактні класи і Entities описів як інтерфейс, тобто, описується ,що потрібно робити, а не як. Бібліотека Room дозволяє працювати з таблицями без занурення в роботу класів, не реалізовуючи методи читання і запису в БД, а за рахунок реалізації на системному рівні . Android робить це сам. Самі методи описані в PetitionDao.

PetitionDao

Відповідає за вибірку з бази даних. У ньому видно, що описані не класи, а інтерфейси, вони не є об'єктами. Є метод getAllActive, який відповідає за "живі дані" з петицій, це потрібно для спостерігача, getPetitionActive для отримання всіх активних петицій та інші. Їх реалізує система за рахунок бібліотеки Room. Фрагмент коду представлено у додатку А.

Petitions

У ньому створюється клас Petitions, де задаються поля таблиці, як-от унікальний ідентифікатор петиції (uuid), номер, назва петиції та інші. Тут же знаходиться функція-конструктор з параметром petition (петиція передана з парсеру) для класу Petitions.

Оскільки зберігання даних в SQL передбачене в тектстовому або цілому вигляді, доводиться використовувати конвертер для значень site, status, publicDate, lastUpd, updateDate. Це пов'язано з тим, що функції для роботи є, а тип даних не передбачений. Коли конвертер отримує на вхід рядок, він форматує його, наприклад як дату, і повертає. І також назад з дати в рядок, це потрібно для того щоб була змога читати з таблиці дані й записувати їх в неї. Фрагмент коду представлено у додатку Б.

DataLine

Тут призначається uuid та номер петиції, остання дата голосування й кількість голосів.

PetitionViewModel

Тут клас PetitionViewModel розширює клас ViewModel. ViewModel для фрагментів, потрібна для того щоб фрагменти не втрачали дані при повороті екрану або перемиканні між вкладками. Тут же присвоюється покажчик на базу даних petitionDao. LiveData повертає як би «живі дані». Коли відбувається поновлення в LiveData, весь список оновлюється і потім повертається.

Тут прописуються параметри для петицій з різним статусом (активні, з відповіддю, на розгляді). Вибираються дані з петицій коли змінюється параметр – статус петиції. Далі вказуються дані для фільтра петицій – тип сортування за полями і яке поле перше. Задаються поле для сортування, порядок сортування і фільтр за голосами.

У PetitionViewModel прописано додавання "дня голосування з графіка" до порівняння і його видалення. Тут же реалізована перевірка, чи є вже такий «день» у списку для порівнянь, очищення списку. addNames і clearNames додає й очищає імена в списку збігів. У смом кінці «живі дані» списку порівнянь і збігів. Фрагмент коду представлено у додатку В.

PetitionListAdapter

Він потрібен для зв'язку recycle view і одержуваних даних. У ньому присвоюється List <Petitions>, кеш, який оновлюється, якщо завантажуються

нові петиції. OnBindViewHolder повертає поточну позицію, якщо в цей момент проглядається Activity. Тут задаються слухачі оброблювачів натискань конпок. ViewHolder для RecyclerView, який відображає петицію в списку. Тут же задаються кольори для різних дельт у голосуванні з петицій. Фрагмент коду представлено у додатку Г.

LineActivity

Відповідає за виведення графіка на єкран. При створенні отримує параметри, uuid, номер петиції і її назву. DataViewModel використовується для збереження даних після завантаження, коли повертається екран при поверненні до Activity, щоб заново не завантажувати дані. Далі обробник натискання на вертикальну лінію – додавання дня голосування з графіка в список порівнянь. Тут же будується гафік і задається, за скільки днів відображати графік при відкритті сторінки. За масштаб відповідає setGranularity зі значенням «1f» – це один день. LiveData теж використовуються, і якщо зміни були – графік теж зміниться. Тут же викликається обробник при обробці даних формату string. Потім створюється форматер, який одержує на вході дату, а на виході повертає рядок у форматі день, місяць. Формується файл міток.

Використовується значення, починаючи з 1. Це зручно, коли значення x відповідає дню. Форматер (утиліта) після передачі їй масиву, за значенням x повертає позицію з масива, тоді 1 = 1-й день, 2 = 2-й день і т.д. Тут же прописаний обробник повернення на попередню Activity. Фрагмент коду представлено у додатку Д.

MainActivity

Це основне вікно додатку – головне Activity, воно розширює клас AppCompatActivity. Тут підключаються кнопки з інтерфейсом, модель даних петицій, кнопки фільтра і сортування. Далі отримується модель даних петиції. При натисканні кнопок задається фільтр за назвою (пошук) і за кількістю голосів петиції, а також сортування петицій за кількістю голосів і датою створення петиції.

У самому кінці коду створюється меню додатку, тут же є обробник вибору меню, де вибирається активний сайт (Rada або President) і вікно про додаток (AboutActivity). PreferenceManager дозволяє глобально зберігати налаштування програми. Фрагмент коду представлено у додатку Е.

PetitionsWorker

Завдання сканування петицій. Це один з трьох файлів, де реалізується завдання планувальника. У ньому спочатку отримується сайт, для якого буде відбуватися сканування (RADA або PRESIDENT). Отримується статус петиції перед скануванням. У разі помилки - повтор завдання. Саме завдання сканування петицій відбувається в такій послідовності: береться список петицій з сайту й перевіряється за списком. Якщо в базі даних немає цієї петиції, створюється запис зі значеннями і зберігається. Якщо петиція вже є – оновлюється. Після цього формується рядок виводу про це завдання і виводитися повідомлення на екран. У нічний час повідомлення не показуються. Фрагмент коду представлено у додатку Ж.

SignersWorker

Завдання сканування голосів з активних петицій. Тут задається максимальна кількість голосів для формування списку сканування. Голоси автоматично збираються з сайту за розкладом тільки за активними петиціями, збирається пачка петицій не більше 100000 голосів. Тут також отримується сайт, для якого збираються голоси. Спочатку петиції (список) упорядковуються, перші ті, за якими не було оновлення та з великою кількістю голосів, потім за датою оновлення. Коли список петицій для сканування сформований, запускається збір голосів. Якщо набереться достатня кількість голосів, сканування припиниться.

Тут же виводиться повідомлення із зазначенням кількості відібраних петицій для сканування. Визначається максимальна кількість голосів і підраховуються голоси за днями. Після збереження результатів виводиться повідомлення про кількість відсканованих петицій і завершується завдання.

Реалізована можливість скасувати сканування. Фрагмент коду представлено у додатку З.

ManualWorker

Завдання для ручного сканування петицій. У ньому отримується UUID петиції та запускається завдання парсингу голосів. Повідомляється, що процес довгий, і виводиться повідомлення про скануванні з кнопкою скасування. У разі помилки – оновлюється статус і задається повторення в планувальнику.

Вже парсер голосів отримує кількість голосів за днями, оновлює статус петиції, формує список для бази даних, підраховує голоси за днями і формує їх список. Видаляються та заново зберігаються голоси та дати за петиціями, оскільки позиція у списку змінюється: сьогодні підписант може бути першим у списку тих, хто проголосував, а завтра він може вже бути n-м. Сортування відбувається за спаданням дати, а не за зростанням кількості підписантів на сайті. Також немає унікального ідентифікатора крім імені, тому простіше видалити та перезаписати, ніж оновлювати кожен запис. Аналогічні дії відбуваються при збереженні результатів у парсері SignersWorker.

Після завершення виводиться повідомлення з кількістю просканованих голосів, також присутня кнопка скасування завдання. Фрагмент коду представлено у додатку И.

Парсинг сторінок електронних петицій до Президента України та Верховної Ради.

За сканування сторінок відповідають PresidentParser та RadaParser. У цих файлах прописана вся логіка поновлення списку петицій у застосунку. Також окремо прописані класи Петиція (Petition) і Підписант (Signer). Перерахування Site – тип сайту; тип статусу петиції (Status): «Активна», «На розгляді», «З відповіддю», «Не підтримано».

Інтерфейс Parser задає, як повинен бути реалізований парсер. А саме: повертає url петицій з різним статусом, встановлює флаг припинення

сканування, повертає статус петиції, сканує петицію окремо, а також списком, сканує голоси петиції, повертає завантажені сторінки з url і "тіло" сторінки.

PresidentParser – парсер сайту з петиціями до президента України. На самому початку коду вказуються змінні, це адреса сторінок сайту, звідки беруться дані. Після цього розшифровується статус петиції за текстовим значенням. При парсингу петиції отримується сторінка (url) і скануються дані за селекторами (номер петиції, дата початку, автор та інші). Далі парситься список петицій.

Використовується бібліотека jsoup [12]. Формується дата, яка записується в базу. Нижче сторінка сайту розбивається на елементи. Парсинг починається з рядка Elements pets = document.select (". Pet_content"); Парситься по еліментам, отримуючи потрібні поля зі сторінки сайту. Тобто спочатку парсер проходить по списку з петиціями, відкривається петиція, яка є на сторінці і беруться дані з неї.

Спочатку скануються петиції (без голосів). Сканується список петицій за URL. Береться сторінка й перевіряється список петицій на ній: якщо порожньо – помилка. Після визнаячення кількості сторінок починається парсинг за сторінками. Буває таке, що парсер не може відкрити сторінку (сайт блокує або зависає) – він пробує це зробити 3 рази з паузою між зверненнями в секунду. Якщо отримати дані не вдається – помилка, завдання скасовується і планувальник перезапускає його сам через деякий час. Якщо все нормально, тоді після отримання списку петицій за селектором петиції будуть скануваться поки вони є в списку, або поки завдання парсинга не буде перерване.

Сканування голосів відбувається ідентично. Отримавши сторінку і пройшовшись по селектарам, парсер рахує голоси, скануючи сторінки поки на них є підписанти. Наприкінці рядки конвертуються в дату і число. Фрагмент коду представлено у додатку К.

У RadaParser алгоритм сканування петиціій до Верховної Ради ідентичний сайту з петиціями до Президента України, тількі інші селекктори для отримання даних.

Аналіз підписантів петицій.

Маючи дані з петицій, а саме кількість голосів за датами і список підписантів, хто проголосував за петицію в ці дні, можна порівняти списки людей, які віддали свій голос у конкретні дати за різними петиціями. У разі знаходження збігів, отриману інформацію можна аналізувати далі й працювати з підписантами, чиї імена повторюються регулярно в петиціях, де зафіксовані стрибки голосів або в тих, де підозрюється можлива фальсифікація або накрутка голосів.

При натисканні на гістограму береться список підписантів за конкретний день, для подальшого порівняння. За весь цей процес відповідає CompareFragment– тут порівнюються статичні списки підписантів за днями, які виділяє користувач на графіках.

Також тут підвантажується PetitionViewModel – модель, де зберігаються дані. Після завантаження моделі створюються слухач зміни списку петицій для порівняння та змін списку підписантів, яких порівняли. При тривалому натисканні позиція видаляється.

У цьому ж фалі прописана кнопка порівняння, яка запускає завдання пошуку збігів – MyAsyncTask та кнопка експорту, що запускає завдання збереження результату порівняння.

При запуску MyAsyncTask створюється список унікальних петицій і якщо унікальних петицій більше одної (інакше нема з чим порівнювати) – почнеться порівняння.

Створюється карта для порівняння – підписант і кількість збігів. Тобто до неї додаються підписанти та кількість збігів. А потім відбираються тільки ті, у кого кількість збігів дорівнює кількості петицій. Тобто, якщо підписант голосував за всі петиції з вибірки.

Ось фрагмент реалізації завдання пошуку збігів:

```
for(String s:signatures){
            if (!maps.containsKey(s))
              maps.put(s,0);
            int count = maps.get(s);
            count++;
            maps.put(s,count);
          }
```

Тут підраховується, скільки разів голосував підписант, а він в одній петиції не може голосувати двічі, отже він лише один раз зустрічатиметься за кожну петицію. Тому пройдемося за підрахунком:

```
for(Map.Entry<String,Integer> entry: maps.entrySet()){
```

Та виберемо лише тих, хто зустрічається рівно стільки разів, скільки є петицій:

```
            if (entry.getValue()==size){
              names.add(entry.getKey());
            }
          }
```

Цей метод працює, оскільки в одній петиції можна голосувати один раз. Тому більше одного разу за петицію підписувач не потрапить до списку. І якщо петицій 5, то він буде зустрічатися не більше ніж 5 разів. Якщо з 5 обраних петицій він голосував у 3 – він не потрапить у підсумок, а якщо у всіх 5, то вважатиметься як збіг.

Завдання також включає створення та виведення індикатору виконання для повідомлення про початок порівняння.

Наприкінці формується список підписантів, що збіглися, і виводиться результат. За збереження результату порівняння відповідає функція SaveTask, таблиця зберігається в системну папку Download на смартфоні.

На сторінці порівняння можна очистити список петицій з датами (за це вдповідає функція clearMessages), порівняти їх або завантажити результат. При натисканні на кнопку експорту перед завантаженням перевіряються права застосунку на запис, і якщо є дані для завантаження, то запускається SaveTask. Фрагмент коду представлено у додатку Л.

3.3. Інструкція користувача

Проаналізувавши аналоги зі збору статистики електроних петицій та розібравшись в роботі Android застосунків було створено алгоритм роботи майбутнього клієнту, що складається з activity, Android фрагментів та завданнь WorkManager для отримання актуальної інформації про голосування на вебсайтах електроних петицій до Президента та Верховної Ради Украни й порівняння підписантів на збіг з різних петицій, для подальшого аналізу отриманих даних також передбачено експорт таблиці з результатами порівняння. Програма має кілька вікон із різними компонентами, які можна переглядати викликом потрібного.

Після запуску мобільного клієнта користувач бачить головне вікно програми. Вікна активних петицій, петицій з відповіддю та на розгляді знаходяться в окремих фрагментах, поруч із ними вікно порівняння. У верхній частині головного вікна, під назвою поточного сайту для аналізу знаходяться: пошук за петиціями, фільтр за кількістю голосів і кнопки сортування.

За замовчуванням стоїть фільтр для відображення петицій, що мають 1000 голосів і більше. Це значення можна змінити та підтвердити вибір кнопкою поруч. Праворуч знаходяться кнопки сортування за датою публікації та кількістю голосів. Нижче розташований рядок пошуку за назвою чи номером петиції. Ці інструменти застосовуються до всіх статусів петицій. Приклад роботи фільтра та пошуку за петиціями зображено на рис. 3.2.

Під час запуску програми скануються петиції з двох сайтів і за активними петиціями раз на годину підвантажуються голоси. Для інших є кнопка завантаження голосів вручну. Служба працює у фоновому режимі, що не зобов'язує користувача тримати відкритим вікно додатку протягом усього оновлення. Справа в горі меню, яке включає в себе кнопку «Про програму» та кнопки вибору сайту для аналізу. Головне activity клієнта, та меню з вибором сайту зображені на рис. 3.1.

Рис. 3.1. Головне Activity клієнта та приклад перемикання сайтів.

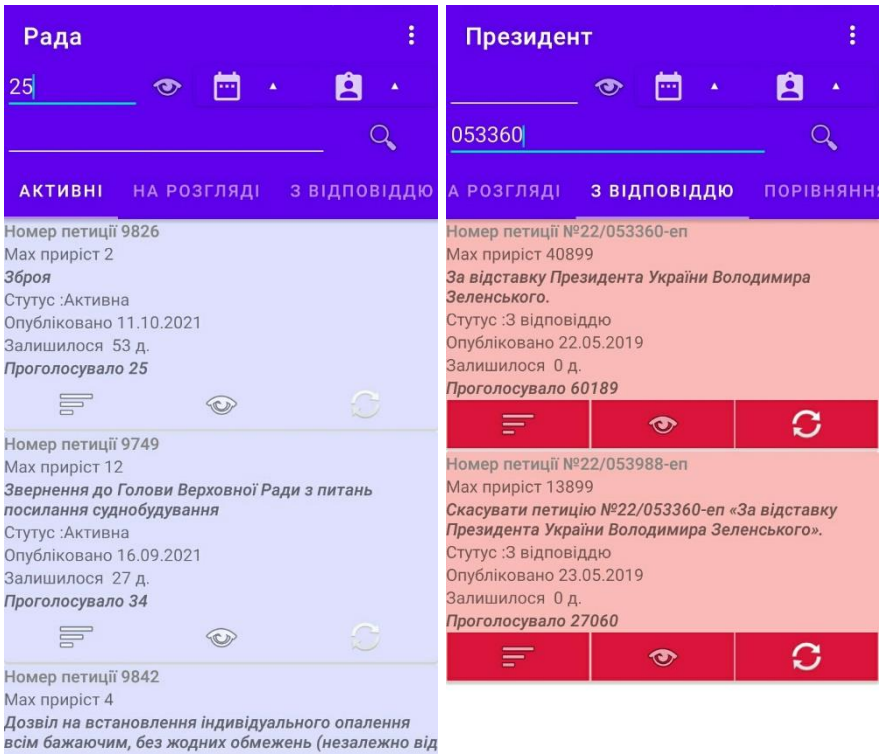

Рис. 3.2. Застосування фільтра за кількістю голосів та пошуку петицій.

Після отримання голосів можна переглянути графік активності або перейти на веб-сторінку з петицією. Петиції, які мають за день більше 1000 голосів, виділені червоним кольором. Петиції, де на день буває понад 100

голосів, також виделені кольором, але вони не так цікаві, тому що такі сплески за кількістю голосів бувають часто, особливо з важливих петицій. Натиснувши кнопку діаграми, користувач побачить activity з гістограмою голосів за днями. Поруч знаходяться кнопка для переходу на вебсайт за адресою з петицією та кнопка ручного оновлення голосів. Приклад зображено на рис. 3.3.

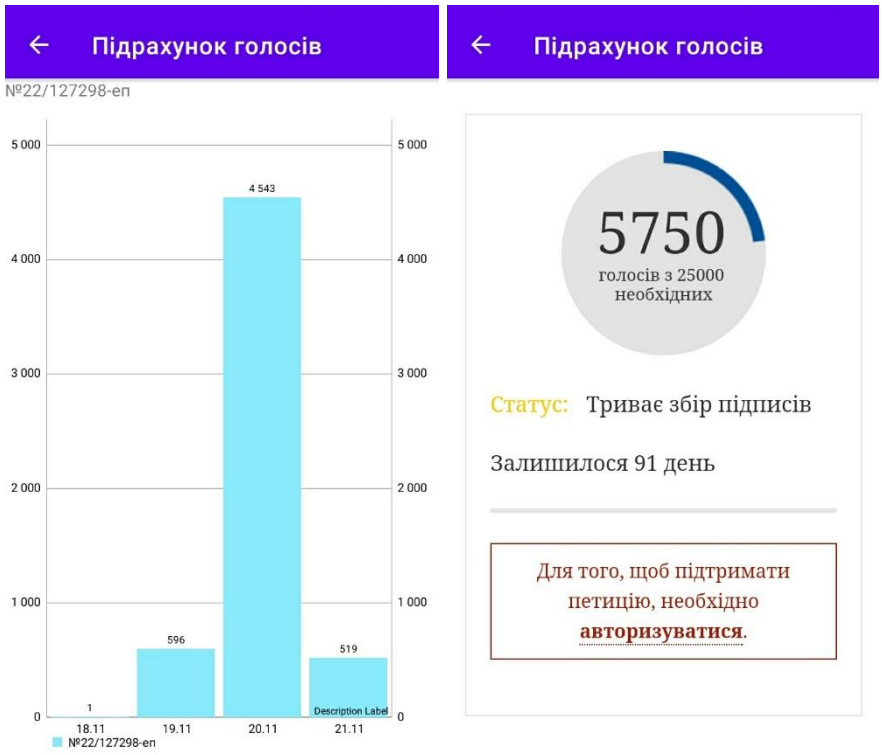

Рис. 3.3. Activity з побудованим графіком та сторінка з петицією.

Для пошуку збігів підписантів з різних петицій необхідно вибрати дату на графіку петиції, з якої підтягнуться підписанти – це робиться натисканням на конкретний день гістограми. При перемиканні між сайтами список петицій для порівнянь не очищається, тому є можливість порівнювати збіги між ними. Мінімальна кількість петицій для порівняння – дві. Можна вибрати більше.

Після вибору необхідних днів для подальшого аналізу потрібно відкрити вікно «Порівняння». Вилучення дня голосування зі списку порівняння робиться довгим натисканням на конкретну дату. Приклад роботи зі списками підписантів зображено на рис. 3.4.

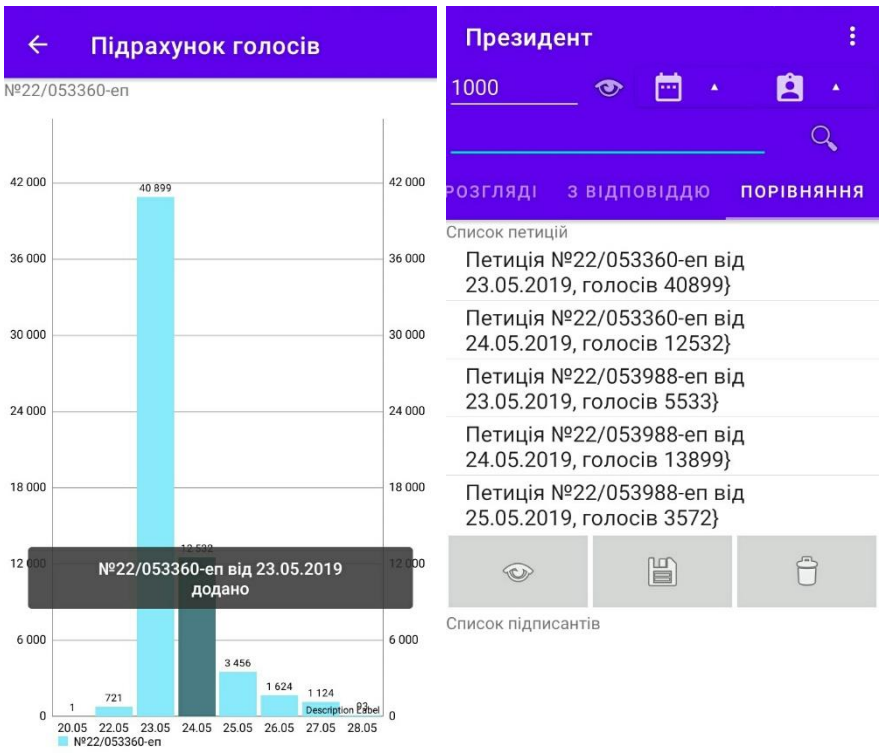

Рис. 3.4. Вибір дати для порівняння та вікно «Порівняння» зі списком.

У відкритому фрагменті порівняння видно список з обраних петицій і конкретні дати та кількість голосів. Для запуску пошуку збігів потрібно натиснути ліву кнопку під списком для порівняння, після цього при необхідності можна зробити експорт даних, натиснувши кнопку збереження поряд. Для очищення списку та результатів порівняння – крайня кнопка видалення праворуч. У разі виявлення збігів підписантів та натисканння кнопки експорту, таблиця зі списком у форматі csv буде збережена у стандартну папку для завантаженнь на пристрої. Приклад процесу порівняння та його результат зображено на рис. 3.5.

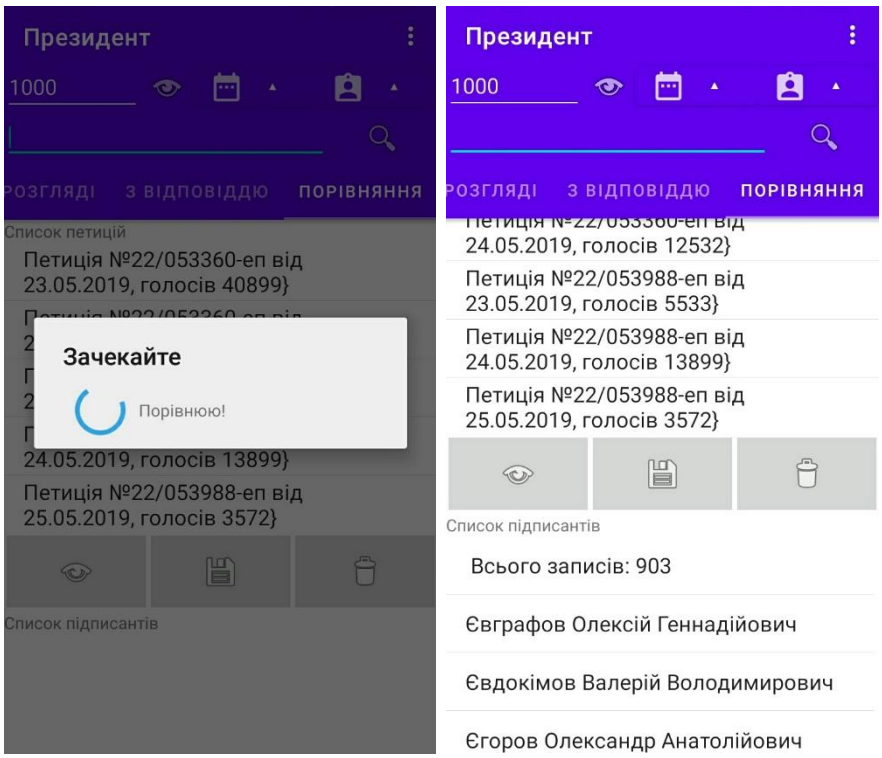

Рис. 3.5. Результат порівняння петицій на на пошук збігів серед підписантів.

Висновки до розділу 3

Створено мобільний клієнт, який надає можливість користувачу дізнатися, за якими онлайн петиціями більш за все та активніше голосують громадяни, та подивитися статистику за голосами, а також порівняти на можливий збіг подпісантів за різними петиціями у конкретні дні голосувань.

Також застосунок дає можливість отримати більш детальнішу інформацію про петицію, та взяти участь у голосуванні, перейшовши на сторінку сайту, де проводиться збір підписів. Користувач застосунку може скористатися пошуком, або фільтрами та сортуванням петицій для більш швидшого знаходження цікавої або потрібної саме йому петиції.

Мобільний клієнт є безкоштовним і може вільно використовуватися на всіх пристроях з операційною системою Android.

Наведено приклади різних петицій, які показують активність і занепокоєння громадян щодо різних тем е-петицій, або можливі накрутки.

ВИСНОВКИ

Аналіз чинної нормативної бази показує, що електронні петиції – відносно новий для України інструмент громадського впливу на ухвалення рішень, однак його популярність зростає.

Вивчивши наявні спроби моніторингу електронних петицій України, можна стверджувати, що ця діяльність знаходиться лише на початковій стадії й здійснюється зазвичай вручну, що доводить потребу створення спеціалізованого програмного забезпечення.

Вивчення загальної структури мобільних застосунків для операційної системи Android дозволило виявити можливості й обмеження цієї платформи для майбутнього програмного забезпечення.

Виходячи з цього, було висунуто вимоги до мобільного клієнту моніторингу збору підписів, серед яких підтримка роботи з петиціями до Президента України та Верховної Ради України, можливість обирати часові періоди для порівняння з різних петицій, візуальне виділення петицій, які викликають підозру.

Було обрано інструменти розробки, серед яких jsoup, MPAndroidChart, guava, WorkManager, SQLite.

Спроєктовано базу даних, у якій зберігатимуться дані про підписантів, завантажені з офіційних серверів у форматі, зручному для подальшого порівняння. Зокрема передбачені таблиці для зберігання голосів, інформації про петиції та інформації про підписантів.

Спроектовано алгоритми та створено програмну реалізацію, що становить основний результат роботи. В процесі тестування суттєвих недоліків не виявлено, отже завдання виконане в повному обсязі.

Створено інструкцію користувача, яка дозволить усім охочим використовувати застосунок для моніторингу процесу збору підписів під електронними петиціями.

Створене в рамках магістерської роботи програмне забезпечення в майбутньому може бути вдосконалене за рахунок покращення зручності

клієнта, надання нових можливостей для користувача у взаємодії з петиціями, вдосконалення та ускладнення аналізу статистики голосувань, реалізації моніторингу звернень до інших органів влади та онлайн-голосувань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронна петиція в Україні. URL : https://uk.wikipedia.org/wiki/Електронна_петиція_в_Україні (дата звернення: 22.07.2020).
2. Указ Президента України №523/2015 Про Порядок розгляду електронної петиції, адресованої Президентові України. URL : https://www.president.gov.ua/documents/5232015-19384 (дата звернення: 22.07.2020).
3. Як працює сервіс електронних петицій. URL : https://petition.president.gov.ua/help (дата звернення: 18.10.2021).
4. Електронні петиції в Україні. URL : https://www.prostir.ua/?library=elektronni-petytsiji-v-ukrajini (дата звернення: 24.07.2020).
5. Державні тітушки Вілкула. Особиста охорона за бюджетні гроші. URL : https://www.pravda.com.ua/articles/2017/01/16/7132578/ (дата звернення: 07.09.2020).
6. Накрутка петиции petition.president. URL : http://nakrutka.net/petition-president/ (дата звернення: 11.10.2020).
7. Е-петиція: бюрократія чи демократія? URL : https://blog.liga.net/user/aemelyanova/article/25016 (дата звернення: 14.10.2020).
8. Програмування під Android. URL : https://uk.wikibooks.org/wiki/Програмування_під_Android (дата звернення: 26.05.2021).
9. Програмування під Android/Структура Android програми. URL : https://uk.wikibooks.org/wiki/Програмування_під_Android/Структура_Android_програми (дата звернення: 26.05.2021).
10. Пять основных принципов дизайна классов (S.O.L.I.D.) в Java. URL :

https://javarush.ru/groups/posts/osnovnye-principy-dizajna-klassov-solid-v-java (дата звернення: 26.05.2021).

11. Как работает Android. Введение для Java-разработчиков. URL : https://javarush.ru/groups/posts/481-kak-rabotaet-android-vvedenie-dlja-java-razrabotchikov (дата звернення: 26.05.2021).
12. jsoup: Java HTML Parser. URL : https://jsoup.org (дата звернення: 15.07.2021).
13. MPAndroidChart HTML. URL : https://github.com/PhilJay/MPAndroidChart (дата звернення: 15.07.2021).
14. Apache Commons Lang. URL : https://commons.apache.org/proper/commons-lang/ (дата звернення: 15.07.2021).
15. WorkManager. URL : https://developer.android.com/jetpack/androidx/releases/work (дата звернення: 22.09.2021).
16. Guava: Google Core Libraries for Java. URL : https://github.com/google/guava (дата звернення: 15.07.2021).
17. Использование Java JSoup для анализа кода HTML. URL : https://o7planning.org/ru/10399/jsoup-java-html-parser-tutorial (дата звернення: 15.07.2021).
18. Парсинг сайта в java с помощью jsoup. URL : https://ru.stackoverflow.com/questions/219399/Парсинг-сайта-в-java-с-помощью-jsoup (дата звернення: 15.07.2021).
19. Guava. Class Sets. URL : https://google.github.io/guava/releases/21.0/api/docs/com/google/common/collect/Sets.html (дата звернення: 15.07.2021).
20. Android charting libraries. URL : https://stackoverflow.com/questions/26467376/android-charting-libraries (дата звернення: 19.08.2021).

21. Создание архитектуры программы или как проектировать табуретку. URL : https://habr.com/ru/post/276593/ (дата звернення: 22.12.2020).
22. Java онлайн для разработчиков. URL : http://java-online.ru/ (дата звернення: 22.12.2020).
23. Android Developers. URL : https://developer.android.com (дата звернення: 03.04.2021).
24. Харди Б., Филлипс Б., Стюарт К., Марсикано К. «Android. Программирование для профессионалов. 2-е изд.» СПб.: Питер, 2016 г.
25. Start Android - учебник по Android для начинающих и продвинутых. URL : https://startandroid.ru (дата звернення: 17.07.2021).
26. Обучение разработке мобильных приложений и игр для ANDROID. URL : http://www.fandroid.info (дата звернення: 18.07.2021).
27. Аналитика первой электронной петиции - можно ли по графику выявить накрутки? URL : https://ed.org.ua/petition.html (дата звернення: 22.11.2021).
28. Е-петиції до ВРУ: підсумки 2017 року. URL : https://www.prostir.ua/?news=e-petytsiji-do-vru-pidsumky-2017-roku (дата звернення: 23.11.2021).

ДОДАТКИ

Додаток А

PetitionDao.java

```
// DAO для петицій
@Dao
@TypeConverters({DateConverter.class})
public interface PetitionDao {
   // "живі дані" з петицій, потрібно для спостерігача
   @Query("SELECT * FROM petition WHERE isActive = 1 order by maxDelta desc")
   LiveData<List<Petition>> getAllActive();
   // отримати всі активні петиції
   @Query("SELECT * FROM petition WHERE isActive = 1")
   List<Petition> getAll();
   @Query("select * from Petition where status = 'З відповіддю'")
   List<Petition> getAnswer();
   @Query("select * from Petition where status = 'З відповіддю' order by maxDelta desc")
   LiveData<List<Petition>> getAnswerLive();
   @Query("select * from Petition where status = 'На розгляді' order by maxDelta desc")
   LiveData<List<Petition>> getUnderLive();
   @Query("select * from Petition where status = 'На розгляді'")
   List<Petition> getUnder();
   // активні
   @Query("select * from Petition where isActive =1 and leftDay<=1 order by leftDay ")
   List<Petition> getCheckActive();
   @Query("SELECT * FROM petition WHERE isActive = 1 and lastUpd <> :lastUpd")
   List<Petition> getByLastUpd(Date lastUpd);
   @Query("select * from petition where num=:num")
   Petition getPetitionByNum(String num);
   // вставка
   @Insert(onConflict = OnConflictStrategy.REPLACE)
   void insertMany(List<Petition> petitions);
  // вставка списком
   @Insert(onConflict = OnConflictStrategy.REPLACE)
   void insert(Petition petition);
 // оновлення
   @Update
   void update(Petition petition);
  // оновлення списком
```

```
    @Update
    void updateMany(List<Petition> petitions);
    // видалення
    @Delete
    void delete(Petition petition); }

//максимальні голоси підраховуються запитом
    @Query("update Petition  set maxDelta  = (select max(votes) from DataLine as p where num=Petition.num)
\n" + "where EXISTS (select 1 from DataLine as d where d.num = Petition.num)")
    void updateMaDelta();
}
```

Додаток Б

Petitions.java

```
// таблиця петиція
@Entity(primaryKeys = {"uuid"})
public class Petitions {
   @NonNull
   Integer id;
   // uuid петиції
   @NonNull
   String uuid;
   String num; // номер
   String name; // назва
   String url; // URL
   String tag; // таг
   // вебсайт котрий сканується
   @NonNull
   @TypeConverters(SiteConverter.class)
   Site site;
   // статус петиції
   @TypeConverters(StatusConverter.class)
   Status status; // статутс
   @TypeConverters(DateConverter.class)
   Date publicDate; // дата публікації
   int  votes ; // кількість голосів
   int leftDay; // залишилось днів
   int signatories;// кількість підписантів
   @TypeConverters(DateConverter.class)
   Date lastUpd; // останє оновлення
   @TypeConverters(DateConverter.class)
   Date updateDate; // останє оновлення голосів
   int isActive = 1; // статус активності
   int  maxDelta = 0; // максимальна зміна
   String author; // автор
```

Додаток В

PetitionViewModel.java

```
public class PetitionViewModel  extends ViewModel {
   private static PetitionViewModel myViewModel;
   public PetitionViewModel() {   }
   // зберігання моделі в в єдиному екземплярі
   public static synchronized PetitionViewModel getInstance() {
      if (myViewModel == null) {
         myViewModel = new PetitionViewModel();
         return myViewModel;      }
      return myViewModel;   }
   // DAO петицій
   private PetitionDao petitionDao = App.getInstance().getDatabase().petitionDao();
   public MutableLiveData<String> filter = new MutableLiveData<>(new String());
   public LiveData<String> getFilter(){
      return filter;   }
   // параметр для активних петицій
   private MutableLiveData<Param> activeParam =
         new MutableLiveData<>(new Param(Status.ACTIVE,Site.PRESIDENT,1000));
   // параметр для петицій з відповіддю
   private  final MutableLiveData<Param> processedParam =
         new MutableLiveData<>(new Param(Status.PROCESSED,Site.PRESIDENT,1000));
   // параметр для петицій на розгляді
   private  final MutableLiveData<Param> inprocessParam =
         new MutableLiveData<>(new Param(Status.INPROCESS,Site.PRESIDENT,1000));
   // для зберігання даних для фрагмента порівняння голосів
   private MutableLiveData<Set<DataLine>>  dataSet =new MutableLiveData<>(new LinkedHashSet<>());
   private MutableLiveData<Set<String>> namesSet = new MutableLiveData<>(new TreeSet<>());
```

Додаток Г

PetitionListAdapter.java

```
//  адаптер петицій для RecycleView
public class PetitionListAdapter extends Adapter<PetitionListAdapter.PetitionViewHolder> {
  private Context context;
  private LayoutInflater layoutInflater;
  // список петиція
  private MutableLiveData<List<Petitions>> petitions;
  // слухачі обробників натискань конпок
  private OnClickShowButtonListener showButtonClickListener;
  private OnClickWebButtonListener goWebButtonClickListener;
  private OnClickUpdateButtonListener updateButtonClickListener;
// ViewHolder для RecycleView, відображає петицію у списку
  public class PetitionViewHolder extends RecyclerView.ViewHolder {
    private TextView tvNum;
    private  TextView tvMaxDelta;
    private  TextView tvName;
    private  TextView tvStatus;
    private  TextView tvPublish;
    private  TextView tvVotes;
    private  TextView tvLeftDay;
    private ImageButton btShow;
    private ImageButton btWeb;
    private  ImageButton btUpdate;
    CardView card;
    void bind(final Petitions petition) {
      if (petition != null) {
        // запис даних з петиції у поля
        tvNum.setText(String.format("Номер петиції %s",petition.getNum()));
        if (petition.isWorker()){
          tvNum.setTextColor(Color.RED);
        }else{
          tvNum.setTextColor(Color.GRAY);
        }
        tvMaxDelta.setText(String.format("Max приріст %d ",petition.getMaxDelta()));
        tvName.setText(petition.getName());
        tvStatus.setText(String.format("Стутус :%s",petition.getStatus().getName()));
        tvPublish.setText(String.format("Опубліковано %s",formatDate(petition.getPublicDate())));
```

```
        tvVotes.setText(String.format("Проголосувало %d",petition.getVotes()));
        tvLeftDay.setText(String.format("Залишилося %d д.",petition.getLeftDay()));
        btShow.setOnClickListener(v -> {
            if (showButtonClickListener != null)
                showButtonClickListener.onShowButtonClicked(petition);
        });
        btWeb.setOnClickListener(v -> {
            if (goWebButtonClickListener != null)
                goWebButtonClickListener.OnGoWebClicked(petition);
        });
        btUpdate.setOnClickListener(v->{
            if (updateButtonClickListener!=null)
                updateButtonClickListener.OnUpdateClicked(petition);
        });
        // розфарбуємо
        if (petition.getMaxDelta()>1000){
            card.setCardBackgroundColor(Color.parseColor("#FCB9B9"));
            btShow.setBackgroundColor(Color.parseColor("#DC143C"));
            btWeb.setBackgroundColor(Color.parseColor("#DC143C"));
            btUpdate.setBackgroundColor(Color.parseColor("#DC143C"));
        }else if (petition.getMaxDelta()>100)
        {
            card.setCardBackgroundColor(Color.parseColor("#7FFFD4"));
            btShow.setBackgroundColor(Color.parseColor("#00CED1"));
            btWeb.setBackgroundColor(Color.parseColor("#00CED1"));
            btUpdate.setBackgroundColor(Color.parseColor("#00CED1"));
        } else {
            card.setCardBackgroundColor(Color.parseColor("#E0E0FF"));
            btShow.setBackgroundColor(Color.parseColor("#E0E0FF"));
            btWeb.setBackgroundColor(Color.parseColor("#E0E0FF"));
            btUpdate.setBackgroundColor(Color.parseColor("#E0E0FF"));
        }
    }
}
```

Додаток Д

LineActivity.java

```
// активіті виведення графіка
public class LineActivity extends AppCompatActivity {
   private DataViewModel dataViewModel;
   TextView tvNum;
   String caller;
   PetitionViewModel petitionModel;
   String num ;
   String uuid;
   private BarChart chart
// обробник натискання на bar
      chart.setOnChartValueSelectedListener(new OnChartValueSelectedListener() {
         @Override
         public void onValueSelected(Entry e, Highlight h) {
            DataLine line = data.get((int) e.getX()-1);
            if (petitionModel.checkDataLine(line)){
               Toast.makeText(getApplicationContext(), String.format("%s від %s вже є в списку ", line.getNum(),
HelpUtil.formatDate(line.getLastUpd()) ), Toast.LENGTH_SHORT).show();
            }else {
               petitionModel.addDataLine(line);
               Toast.makeText(getApplicationContext(), String.format("%s від %s додано", line.getNum(),
HelpUtil.formatDate(line.getLastUpd()) ), Toast.LENGTH_SHORT).show();}}
// кількість днів, які можна відобразити відразу
      chart.setMaxVisibleValueCount(90);
      chart.setPinchZoom(false);
      chart.setDrawGridBackground(false);
      XAxis xAxis = chart.getXAxis();
      xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
      xAxis.setDrawGridLines(false);
      xAxis.setGranularity(1f); //  масштаб до 1 дня
      YAxis leftAxis = chart.getAxisLeft();
      leftAxis.setLabelCount(8, false);
      leftAxis.setSpaceTop(15f);
      leftAxis.setAxisMinimum(0);
      YAxis rightAxis = chart.getAxisRight();
      rightAxis.setDrawGridLines(false);
      rightAxis.setLabelCount(8, false);
      rightAxis.setSpaceTop(15f);
```

```
        rightAxis.setAxisMinimum(0);
        // база даних та слухач бази
        AppDatabase database = App.getInstance().getDatabase();
        LiveData<List<DataLine>> liveData =
database.dataLineDao().getByNumOrderByLast(dataViewModel.getPetition());
        liveData.observe(this, new Observer<List<DataLine>>() {
          // якщо були отримані дані
          @Override
          public void onChanged(@Nullable List<DataLine> value) {
            // формат мітки дати
            data.addAll(value);
            SimpleDateFormat dateFormat = new SimpleDateFormat("dd.MM"); // день місяць
            String[] days = new String[value.size()]; //масив міток
if (value != null){
              ArrayList<BarEntry> values = new ArrayList<>();
              for (int i=0;i<value.size();i++)  {
                values.add(new BarEntry(i+1, value.get(i).getVotes()));
                days[i]=dateFormat.format(value.get(i).getLastUpd()); }
              BarDataSet set = new BarDataSet(values, num);
              DayAxisValueFormatter dayAxisValueFormatter = new DayAxisValueFormatter(days);
              XAxis xAxis = chart.getXAxis();
              xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
              xAxis.setDrawGridLines(false);
              xAxis.setGranularity(1f); // only intervals of 1 day
              xAxis.setLabelCount(7);
              xAxis.setValueFormatter(dayAxisValueFormatter);
              BarData data = new BarData(set);
              data.setBarWidth(0.9f); // ширина бара
              chart.setData(data); // встановлення даних
              chart.setFitBars(true); // make the x-axis fit exactly all bars
              chart.invalidate(); // оновлення
 }  }  });
@Override
  public void onBackPressed() { // обробник повернення назад на ту активність, з якою викликали
        Log.d("MyLogs","onBack "+caller);
        Intent intent = null;
          intent= new Intent(this, MainActivity.class);
          intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_SINGLE_TOP);
        startActivity(intent);
        finish(); }
```

Додаток Е

MainActivity.java

```
// основное вікно
public class MainActivity extends AppCompatActivity {
   // ім'я параметра для збереження активної вкладки
   private final static String ACTIVE_TAB="ACTIVE_TAB";
   private ActivityMainBinding binding;
   private Toolbar toolbar;
   // модель даних
   private PetitionViewModel model;
   private  TabLayout tabs;
   private EditText fVotes;
   private EditText fName;
   // кнопки фільтру та сортування
   private ImageButton btFilterVotes;
   private Button btPublicDate;
   private Button btSortVotes;
   private ImageButton btFilterName;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      binding = ActivityMainBinding.inflate(getLayoutInflater());
      setContentView(binding.getRoot());
      // отримання моделі
      PetitionViewModelFactory singletonNameViewModelFactory = new
PetitionViewModelFactory(PetitionViewModel.getInstance());
      model = new ViewModelProvider(this,singletonNameViewModelFactory).get(PetitionViewModel.class);
      SectionsPagerAdapter sectionsPagerAdapter = new SectionsPagerAdapter(this,
getSupportFragmentManager());
      ViewPager viewPager = binding.viewPager;
      viewPager.setAdapter(sectionsPagerAdapter);
      tabs = binding.tabs;
      toolbar = binding.toolbar;
      toolbar.inflateMenu(R.menu.main);
      toolbar.setTitle("Президент");
      tabs.setupWithViewPager(viewPager);
      // відновлення активної вкладки
      SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
```

```
int active = sharedPreferences.getInt(ACTIVE_TAB,0);
tabs.getTabAt(active).select();
setSupportActionBar(toolbar);
fVotes = binding.fVotes;
fVotes.setText("1000");
model.setActiveVotes(1000);
fName = binding.fName;
fName.setText(model.filter.getValue());
btFilterName = binding.btFilterName;
// при натисканні кнопки задається фільтр
btFilterName.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      model.filter.setValue(fName.getText().toString());  }  });
btFilterVotes = binding.btFilterVotes;
// при натисканні кнопки задається фільтр голосів
btFilterVotes.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
       int votes = 0;
       try {
          votes = Integer.parseInt(fVotes.getText().toString());
       }catch (Exception e){
       }
      model.setActiveVotes(votes);
      model.setInProcessVotes(votes);
      model.setProcessedVotes(votes); }  });
btSortVotes = binding.btSortVotes;
// при натисканні кнопки сортування за кількістю голосів
btSortVotes.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v)  {
      model.fieldFirst.setValue(Field.VOTE);
      model.setField(Field.VOTE);
     SortOrder value =  model.fieldVotes.getValue();
     if (value==SortOrder.ASC) {
        btSortVotes.setText(R.string.voteDESC);
        value=SortOrder.DESC; }
     else {value =SortOrder.ASC;
```

```
            btSortVotes.setText(R.string.voteASC); }
          model.fieldVotes.setValue(value);
          model.setVotesSort(value); } });
    btPublicDate= binding.btPublicDate;
    // при натисканні кнопки відсортується за датою створення
    btPublicDate.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        model.fieldFirst.setValue(Field.DATE);
        model.setField(Field.DATE);
        SortOrder value =  model.fieldPublicDate.getValue();
        if (value==SortOrder.ASC) {
          value=SortOrder.DESC;
          btPublicDate.setText(R.string.dateDESC);  }
        else {
          value =SortOrder.ASC;
          btPublicDate.setText(R.string.dateASC);  }
        model.fieldPublicDate.setValue(value);
        model.setPublicDateSort(value); }  });    }
  // створення меню
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main,menu);
    if(menu.getClass().getSimpleName()
          .equals("MenuBuilder")){
      try{
        Method m = menu.getClass()
              .getDeclaredMethod (
                    "setOptionalIconsVisible",
                    Boolean.TYPE);
        m.setAccessible(true);
        m.invoke(menu, true); }
      catch(NoSuchMethodException e){
        System.err.println("onCreateOptionsMenu");  }
      catch(Exception e){
        throw new RuntimeException(e); }  }
    for(int i=0;i<menu.size();i++){
      MenuItem item = menu.getItem(i);
      SpannableString title = new SpannableString(item.getTitle().toString());
      title.setSpan(new ForegroundColorSpan(Color.parseColor("#FF3700B3")),0,title.length(),0);
```

```
            item.setTitle(title); }
        return true;    }
    // обробник меню
    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item) {
        System.out.println("select "+item.getTitle());
        switch (item.getItemId()){
            case R.id.rada:{
                // оголошення що сайт рада
                item.setChecked(true);
                model.setSite(Site.RADA);
                toolbar.setTitle("Рада");
                return true; }
            case R.id.president:{
                // оголошення що сайт президент
                model.setSite(Site.PRESIDENT);
                toolbar.setTitle("Президент");
                item.setChecked(true);
                return true;}
            case R.id.mAbout:{
                // Про програму
                Intent intent = new Intent(this, AboutActivity.class);
                startActivity(intent); // виклик активіті
                return true;  }
            default:
                super.onOptionsItemSelected(item); }
        return true;  }
    // збереження активної вкладки
    @Override
    protected void onSaveInstanceState(@NonNull @NotNull Bundle outState) {
        int active = tabs.getSelectedTabPosition();
        outState.putInt(ACTIVE_TAB,active);
        Log.v("SAVE_INS",String.format("save %d ",active));
        SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        SharedPreferences.Editor ed = sharedPreferences.edit();
        ed.putInt(ACTIVE_TAB,active);
        ed.commit();
        super.onSaveInstanceState(outState);  } }
```

Додаток Ж

PetitionsWorker.java

```
// завдання сканування петицій
public class PetitionsWorker extends Worker {
   private static  String TAG = App.PRESIDENT_ACTIVE;
   private  static  String URL;
   AppDatabase database = App.getInstance().getDatabase();
   PetitionDao dao = database.petitionDao();
   Parser parser = null;
   Site site=null;
   Status status =null;
   Date now = new Date();
   int size =0;
public Result doWork() {
      // отримання сайту який сканується
      site = Site.valueOf(getInputData().getString("SITE"));
      switch (site){
         case RADA: parser = new RadaParser(); break;
         case PRESIDENT:parser = new PresidentParser();break; }
      // отримання статусу за яким сканується
      status = Status.valueOfName(getInputData().getString("STATUS"));
      switch (status){
         case ACTIVE:URL = parser.getActiveUrl();break;
         case INPROCESS: URL =parser.getInProcessUrl();break;
         case PROCESSED: URL =parser.getAnswerUrl();break;
         default:parser.getActiveUrl();break; }
      TAG = getInputData().getString("TAG");
      try {
         parse();
      } catch (ParseError parseError) {
         Log.v(TAG,"ERROR "+parseError.getMessage());
          // якщо помилка сканування - повторити
          return Result.retry();
      } catch (EmptyException e) {  }
      Date current = new Date();
      long duration = (current.getTime()-now.getTime())/1000;
      Log.v(TAG,"SUCCESS");
      return Result.success();  }
```

```
// сканування петиції
   private void parse() throws ParseError, EmptyException {
      // отримання списку
      List<Petition> listPetition  = parser.parsePetitions(URL);
      Log.v(TAG,String.format("get %d",listPetition.size()));
      // прохід за списком
      for(Petition petition:listPetition){
         // зчитування петиції з бази
         Petitions p =  dao.findByKey(petition.getId(),site);
         // якщо такої немає - створи
         if (p==null){
            p = new Petitions(petition);
            if (status!=Status.ACTIVE) p.setIsActive(0);
            Log.v(TAG,String.format("New petition %s %d %s",p.getUuid(),p.getId(),p.getSite()));
            // зберегти
            dao.insert(p);
         }else {
            // якщо є – оновити
            p.setVotes(petition.getVotes());
            Log.v(TAG,String.format("Update petition %s %d %s",p.getUuid(),p.getId(),p.getSite()));
            dao.update(p);   }   }
      size = listPetition.size();
      // сформувати рядок виводу
      String info = String.format("Скановано %d петицій з сайту %s ",size,site);
      String title = String.format("Сканування петицій %s",status.getName());
      Log.v(TAG,title+" ** "+info+" ** "+App.count.get());
      // виведення повідомлення
      displayNotification(title,info);
}
```

Додаток 3

SignersWorker.java

```
// завдання сканування голосів за петиціями
public class SignersWorker extends Worker {
   private static  String TAG = App.PRESIDENT_SIGNER;
   AppDatabase database = App.getInstance().getDatabase();
   PetitionDao dao = database.petitionDao();
   SignatureDao signerDao = database.signatureDao();
   Parser parser = null;
   Site site=null;
   Date now = new Date();
   final int nid = App.count.getAndIncrement();
   // максимальна кількість голосів для сканування для формування списку сканування
   int maxSigners = 100000;
   int size=0;
private void parse() throws ParseError, EmptyException {
      List<Petitions> listAll =  dao.getAllBySiteToScan(site);
      Log.v(TAG,"get list "+listAll.size());
      final  long minTime = 3*60*60;
      Log.v(TAG,"new list "+listAll.size());
      // сортування, спочатку тих, за яким не було оновлення з великою кількістю головос
      // потім за датою оновлення
      Collections.sort(listAll,
      new Comparator<Petitions>() {
         @Override
         public int compare(Petitions o1, Petitions o2) {
            if (o1.getUpdateDate()==null) {
               Integer v1 = o1.getVotes();
               Integer v2 = o2.getVotes();
               return v2.compareTo(v1);  }
            if (o2.getUpdateDate()==null) return -1;
            return o1.getUpdateDate().compareTo(o2.getUpdateDate());} });
      // формується список петицій для сканування
      List<Petitions> toScan = new ArrayList<>();
      Log.v(TAG,"toScan list "+listAll.size());
      int sum=0;
      for (Petitions p:listAll){
         sum+=p.getVotes();
```

```
            toScan.add(p);
            // якщо набралося достатньої кількості голосів - припинимо сканування
            if (sum>maxSigners) break;  }
        // оголошення що процес довгий і вивід повідомлення
        setForegroundAsync(createForegroundInfo(String.format("До сканування відібрано %d
петицій",toScan.size())));
        Log.v(TAG,"size "+toScan.size()+" voices "+sum);
        final Date now = new Date();
        int k=0;
        for (Petitions petition:toScan){
            Petition p = new Petition(petition);
            Log.v(TAG,"parse "+p.getUrl());
            Log.v(TAG,"scan "+p.getNum()+" "+p.getSite());
            List<Signer> signers =  parser.parseSingers(p);
            Log.v(TAG,"scaned signers "+signers.size());
            p.setSigners(signers.size());
            Map<Date,Integer> map = new HashMap<>();
            for(Signer signer:signers){
                if (!map.containsKey(signer.getSignDate())) map.put(signer.getSignDate(),0);
                int c = map.get(signer.getSignDate())+1;
                map.put(signer.getSignDate(),c); }
            // пошук максимальної кількості голосів
            int max = 0;
            for(Map.Entry<Date,Integer> entry: map.entrySet()){
                if (max<entry.getValue()) max =entry.getValue();   }
            Petitions ps = new Petitions(p);
            ps.setMaxDelta((int) max);
            ps.setUpdateDate(now);
            if (ps.getStatus()!=Status.ACTIVE) ps.setIsActive(0);
            List<Signature> signersList = new ArrayList<>();
            for(Signer s: signers){
                Signature ss = new Signature(s,p);
                signersList.add(ss); }
            signerDao.deleteByUUID(p.getUuid());
            signerDao.insertMany(signersList);
            dao.update(ps);
            ArrayList<DataLine> lines = new ArrayList<>(); // голоса
            Map<Date, Integer> nameCount = new HashMap<>();
             // підрахунок голосів по днях
            for(Signature signer:signersList){
```

```
            if (!nameCount.containsKey(signer.getVoteDate())) nameCount.put(signer.getVoteDate(),0);
            int c = nameCount.get(signer.getVoteDate())+1;
            nameCount.put(signer.getVoteDate(),c); }
          for (Map.Entry<Date,Integer> entry:nameCount.entrySet()){
            DataLine line = new DataLine(p.getUuid(),p.getNum(),entry.getKey(),entry.getValue());
            lines.add(line);  }
          // збереження результату
          database.dataLineDao().deleteByUUID(p.getUuid());
          database.dataLineDao().insertMany(lines);
          String info  = "Сканована петицій - %d";
          k++;
          // виведення повідомлення про кількість сканованих петицій
          setForegroundAsync(createForegroundInfo(String.format(info,k))); }
        size = toScan.size();
        String info = String.format("Скановано %d петицій з сайту %s ",toScan.size(),site);
        String title = "Сканування голосів ";
        Log.v(TAG,title+" ** "+info+" ** "+nid);
        // завершили сканування
        displayNotification(title,info); }
    private ForegroundInfo createForegroundInfo(@NonNull String progress) {
        Context context = getApplicationContext();
        String id = String.format("%d",nid);
        String title = "Сканування голосів";
        String cancel = "Відмінити";
        // можливість скасувати сканування
        PendingIntent intent = WorkManager.getInstance(context).createCancelPendingIntent(getId());
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
          NotificationChannel channel = new NotificationChannel("calcVoice", "calcVoice",
    NotificationManager.IMPORTANCE_DEFAULT);
          notificationManager.createNotificationChannel(channel); }
        Notification notification = new NotificationCompat.Builder(context, "calcVoice")
            .setContentTitle(title)
            .setContentText(progress)
            .setTicker(title)
            .setSmallIcon(R.mipmap.ic_launcher)
            .setOngoing(true)
            .addAction(android.R.drawable.ic_delete, cancel, intent)
            .build();
        return new ForegroundInfo(nid,notification);   } }
```

Додаток И

ManualWorker.java

```
// завдання для ручного сканування петиції
public class ManualWorker extends Worker {
   private static String TAG = App.MANUAL_PARSER;
   AppDatabase database = App.getInstance().getDatabase();
   PetitionDao dao = database.petitionDao();
   SignatureDao signerDao = database.signatureDao();
   Parser parser = null;
   Site site = null;
   Status status = null;
   Date now = new Date();
   final int nid = App.count.getAndIncrement();
   Petitions petition =null;
NotificationManager notificationManager;
   public ManualWorker(@NonNull Context context, @NonNull WorkerParameters workerParams) {
      super(context, workerParams);
      notificationManager = (NotificationManager)
            context.getSystemService(NOTIFICATION_SERVICE);    }
   public Result doWork() {
      // отримання UUID петиці
      String uuid = getInputData().getString("UUID");
      // якщо такої петиції немає - повернення помилки
      if (uuid == null || uuid.isEmpty()) return Result.failure();
      petition = dao.findByUUID(uuid);
      if (petition == null) return Result.failure();
      Site site = petition.getSite();
      switch (site) {
         case PRESIDENT:
            parser = new PresidentParser();
            break;
         case RADA:
            parser = new RadaParser();
            break;    }
      try {
         // оголошення що процес довгий та виведення повідомлення про сканування з кнопкою скасування
         setForegroundAsync(createForegroundInfo("Сканую "+petition.getNum()));
         Log.v(TAG, "start scan "+petition.getUuid());
         petition.setWorker(true);
```

```
        dao.update(petition);
        petition = dao.findByUUID(uuid);
        parse(petition);
    } catch (ParseError parseError) {
        // якщо сталася помилка сканування
        parseError.printStackTrace();
        Log.v(TAG,"ERORR "+parseError.getMessage());
        // відновлення статусу
        if (petition!=null) {
            Log.v(TAG,"set worker false ");
            petition.setWorker(false);
            dao.update(petition);
            petition=null;      }
        // вказівка планувальнику, що потрібно повторити
        return Result.retry();    }
    return Result.success();  }
 // парсер голоів
public void parse(Petitions petition) throws ParseError {
    Petition p = new Petition(petition);
    List<Signer> signers = parser.parseSingers(p);
    Log.v(TAG,"singers "+signers.size());
    p.setSigners(signers.size());
    Map<Date, Integer> map = new HashMap<>();
    int max = 0;
    // максимальна кількість голосів по днях
    for(Signer signer:signers){
        if (!map.containsKey(signer.getSignDate())) map.put(signer.getSignDate(),0);
        int c = map.get(signer.getSignDate())+1;
        if (c>max) max=c;
        map.put(signer.getSignDate(),c);     }
    petition.setMaxDelta(max);
    petition.setUpdateDate(now);
    petition.setVotes(p.getVotes());
    petition.setSignatories(signers.size());
    // якщо петиція не активна встановить статус isActive
    if (petition.getStatus() != Status.ACTIVE) petition.setIsActive(0);
    List<Signature> signersList = new ArrayList<>();
    // для dao держав сформуватио список
    for (Signer s : signers) {
        Signature ss = new Signature(s, p);
```

```
   signersList.add(ss);      }
// підрахунок голосів по днях
ArrayList<DataLine> lines = new ArrayList<>(); // голоса
Map<Date, Integer> nameCount = new HashMap<>();//
for(Signature signer:signersList){
   if (!nameCount.containsKey(signer.getVoteDate())) nameCount.put(signer.getVoteDate(),0);
   int c = nameCount.get(signer.getVoteDate())+1;
   nameCount.put(signer.getVoteDate(),c);      }
// сформувати список голосів по днях
for (Map.Entry<Date,Integer> entry:nameCount.entrySet()){
   DataLine line = new DataLine(p.getUuid(),p.getNum(),entry.getKey(),entry.getValue());
   lines.add(line);      }
// видаллення голосів за петиціями так як позиція у списку змінюється
signerDao.deleteByUUID(petition.getUuid());
// збереження голосів
signerDao.insertMany(signersList);
// видалення днів
database.dataLineDao().deleteByUUID(p.getUuid());
// збереження днів
database.dataLineDao().insertMany(lines);
petition.setWorker(false);
// збереження петиції
dao.update(petition);
petition  =null;
// виведення повідомлення
setForegroundAsync(createForegroundInfo(String.format("Скановано %d голосів",signersList.size()))); }
```

Додаток К

PresidentParser.java

```
// парсер сайту президент
public class PresidentParser implements Parser {
   private boolean terminate = false;
   final static String MainURL = "https://petition.president.gov.ua";
   public final static String activeUrl =
"https://petition.president.gov.ua/?status=active&sort=date&order=desc";
   final static String answerUrl =
"https://petition.president.gov.ua/?status=processed&sort=votes&order=desc";
   final static String inProcessUrl =
"https://petition.president.gov.ua/?status=in_process&sort=votes&order=desc";
   final static String archiveUrl = "https://petition.president.gov.ua/archive&sort=votes&order=desc";
   final static Site site = Site.PRESIDENT;
   public String getPage(String url, int page) {
      return String.format("%s&page=%d", url, page);    }
public Status getStatus(String param) {
      // розшифровка статусу за текстовим значенням
      Status status = null;
      switch (param) {
         case "Очікує на розгляд":
            status=Status.ACTIVE; break;
         case "Триває збір підписів":
            status = Status.ACTIVE;
            break;
         case "З відповіддю":
            status = Status.PROCESSED;
            break;
         case "На розгляді":
            status = Status.INPROCESS;
            break;
         case "Не підтримано":
            status = Status.NOTSUPORTED;
            break;
         case "Успішно підтримано":
            status = Status.PROCESSED;
            break;
         case "Успішно підтримано З відповіддю":
```

```
            status = Status.PROCESSED;
            break;      }
    if (status == null) System.out.println(param);
    return status;   }
@Override
public Petition parsePetition(String url) throws ParseError {
    try {
        // отримання сторінки
        Document document = Parser.getDocument(url);
        Petition pet = new Petition(Site.PRESIDENT);
        pet.setUrl(url);
        // парсинг даних по селекторам
        pet.setNum(document.selectFirst(".pet_number").html());
        Element el = document.selectFirst(".pet_date_container");
        Elements els = el.select(".pet_date");
        pet.setAuthor(els.get(0).text().replace("Автор (ініціатор): ", ""));
        pet.setPublicDate(parseLocalDate(els.get(1).text().replace("Дата оприлюднення: ", "")));
        pet.setName(document.select(".page_left").select("h1").first().text());
        pet.setStatus(getStatus(document.select(".petition_votes_status").text().replace("Статус: ", "")));
        String days = document.selectFirst(".votes_progress_label").text();
        //Збір підписів завершено
        if (days.equalsIgnoreCase("Збір підписів завершено")) pet.setLeftDays(0);
        else
            pet.setLeftDays(Integer.parseInt(days.replaceAll("[^0-9]", "")));
        pet.setVotes(Integer.parseInt(document.select(".petition_votes_txt").
                select("span").first().text().replaceAll("[^0-9]", "")));
        return pet;
    } catch (IOException e) {
        e.printStackTrace();
        throw new ParseError();
    }
}
@Override
public List<Petition> parsePetitions(String url) throws ParseError {
    List<Petition> list = new ArrayList<>();
    // кількість сторінок
    int max = 1;
    try {
        // отримання сторінки
        Document document = Parser.getDocument(url);
```

```
String t = "div.page_settings > div.pag > ul>li";
Elements ul = document.select(t);
// перевірка списку петицій на сторінці, якщо ні - повернення помилки
if (ul == null) {
   throw new ParseError();   }
// пошук кількості сторінок
for (int i = 0; i < ul.size() && terminate == false; i++) {
   Element e = ul.get(i);
   String text = e.getElementsByTag("a").text();
   int k = getInt(text);
   if (k > max) {
      max = k;    }    }
// початок парсингу по сторінках
for (int p = 1; p <= max && terminate == false; p++) {
   String urlPage = String.format("%s&page=%d", url, p);
   // кількість спроб прочитати сторінку
   int k = 0;
   document = null;
   // зчитування не більше 3 разів
   while (k < 2) {
      try {
         document = Parser.getDocument(urlPage);
         break;
      } catch (SocketException e) {
         k++;
         document = null;
         // якщо була помилка – зачекати 1 секунду
         try {
            Thread.sleep(1000);
         } catch (InterruptedException ex) {
            ex.printStackTrace();     }    }    }
   // якщо нічого не отримали - помилка
   if (document == null) throw new ParseError();
   // отримання списку петицій за селектором
   Elements pets = document.select(".pet_content");
   // якщо нічого немає - повернути до порожнього списку
   if (pets == null && pets.isEmpty()) {
      return list;    }
   // розбирати петиції поки вони є у списку або поки не прерветься завдання
   for (int i = 0; i < pets.size() && terminate == false; i++) {
```

```
            Element pet = pets.get(i);
            Petition petition = new Petition(Site.PRESIDENT);
            String num = pet.selectFirst(".pet_number").text().trim();
            petition.setNum(num);
            String url_ = pet.selectFirst(".pet_link").attr("href");
            petition.setUrl(MainURL + url_);
            int id=0;
            try {
              id = Integer.parseInt(url_.replaceAll("[^0-9]",""));
            }catch (Exception e){   }
            petition.setId(id);
            petition.setName(pet.selectFirst(".pet_link").text().trim());
            petition.setTag(pet.selectFirst(".pet_tag").text().trim());
            petition.setPublicDate(parseDate(pet.selectFirst(".pet_date").text()));
            petition.setStatus(getStatus(pet.selectFirst(".pet_status").text()));
            petition.setVotes(getInt(pet.selectFirst(".pet_counts strong").text()));
            try{
              petition.setLeftDays(parseLeftDay(pet.selectFirst(".pet_timer").text()));}
            catch (Exception e){
              petition.setLeftDays(0);              }
            list.add(petition);  }
          try {
            Thread.sleep(100);
          } catch (InterruptedException ex) {
            ex.printStackTrace();    }  }
    } catch (IOException e) {
      e.printStackTrace();
      throw new ParseError();   }
    return list;   }

// парсинг голосів
@Override
public List<Signer> parseSingers(Petition petition) throws ParseError {
    List<Signer> list = new ArrayList<>();
    try {
      // отримання сторінки
      Document document = Parser.getDocument(petition.getUrl());
      // проходження по селекторам
      Element el = document.selectFirst(".pet_date_container");
      Elements els = el.select(".pet_date");
```

```
petition.setAuthor(els.get(0).text().replace("Автор (ініціатор): ", ""));
petition.setStatus(getStatus(document.select(".petition_votes_status").text().replace("Статус: ", "")));
String days = document.selectFirst(".votes_progress_label").text();
int ld = 0;
if (days.equalsIgnoreCase("Збір підписів завершено")) petition.setLeftDays(ld);
else{
    try{
       ld  = Integer.parseInt(days.replaceAll("[^0-9]", ""));
    }catch (Exception e){
       ld=0;  }
   petition.setLeftDays(ld);    }
petition.setVotes(Integer.parseInt(document.select(".petition_votes_txt").
      select("span").first().text().replaceAll("[^0-9]", "")));
int page = 0;
final String urlEnd = "/votes/%d";
boolean isRun = true;
// парситься поки є підписники на сторінці
while (isRun) {
   page++;
   try {
      // затримка щоб севрер вспі віддати данні
      Thread.sleep(50);
   } catch (InterruptedException ex) {    }
   String url = new String().concat(petition.getUrl()).concat(urlEnd);
   String json = null;
   // кількість спроб
   int k = 0;
   // якщо меньше 4
   while (k < 3) {
      try {
         // отримання json пакету
         json = Parser.getBody(String.format(url, page));
         break;
      } catch (SocketException e) {
         // якщо помилка
         json = null;
         k++;
         // зачекати
         try {
            Thread.sleep(100);
```

```
            } catch (InterruptedException ex) {
                ex.printStackTrace();   }    }    }
        // якщо все одно нічого не отримали
        if (json==null) {
          // викликати помилку
          throw  new ParseError();  }
        String data = StringEscapeUtils.unescapeJava(json).trim();
        if (data.startsWith("<!DOCTYPE html>")) {
          isRun = false;
          break;   }
        Document doc = Jsoup.parseBodyFragment(data);
        try {
          Thread.sleep(10);
        } catch (InterruptedException e) {
          e.printStackTrace();   }
        Element body = doc.body();
        // зчитування таблиці
        Elements pets = body.getElementsByClass("table_row");
        if (pets == null || pets.isEmpty()) {
          break;  }
        //  прохід по рядках
        for (Element pet : pets) {
          String num = pet.getElementsByClass("table_cell number").html().replace(".", "");
          String elDate = pet.getElementsByClass("table_cell date").html();
          String name = pet.getElementsByClass("table_cell name").html();
          if (name==null || name.isEmpty()) {
            Log.d("SIGNER",pet.html());
              name= UUID.randomUUID().toString();   }
          if (num==null ||num.isEmpty()) {throw new ParseError();}
          if (elDate==null || elDate.isEmpty()) {throw new ParseError();}
          Signer signer = new Signer(petition.getUuid());
          signer.setNum(getInt(num));
          signer.setName(name);
          signer.setSignDate(parseLocalDate(elDate));
          list.add(signer);   }   }
    } catch (IOException e) {
      e.printStackTrace();
      // якщо щось пішло не так - викликати помилку
      throw new ParseError();       }
    return list;   }
```

```
  // рядок у дату
  public static Date parseLocalDate(String in) {
     Locale locale = new Locale("uk", "UA");
     SimpleDateFormat format = new SimpleDateFormat("dd MMMM yyyy", locale);
     Date ret = null;
     try {
        ret = format.parse(in);
     } catch (ParseException ex) {
        return null;       }
     return ret;    }
  // рядок у число
  public static int parseLeftDay(String str) {
     return Integer.parseInt(str.split(" ")[1]);    }
  // рядок у дату (різні поля)
  public static Date parseDate(String in) {
     String str = in.replace("Дата оприлюднення: ", "");
     Locale locale = new Locale("uk", "UA");
     DateFormat dateFormat = DateFormat.getDateInstance(
           DateFormat.DEFAULT, locale);
     SimpleDateFormat fsql = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
     SimpleDateFormat format = new SimpleDateFormat("dd MMMM yyyy", locale);
     Date ret = null;
     try {
        ret = format.parse(str);
     } catch (ParseException ex) {       }
     return ret;    }
  // рядок у число
  public static int getInt(String text) {
     int k = -1;
     try {
        k = Integer.parseInt(text);
     } catch (Exception e) {       }
     return k;
  }
}
```

Додаток Л

CompareFragment.java

```
public class CompareFragment extends Fragment {
   private ArrayAdapter<DataLine> adapterPetitions;
   private ArrayAdapter<String> adapterNames;
   ListView listViewPetitions;
   ListView listViewNames;
     // модель де зберігаються дані
   PetitionViewModel petitionModel;
   public static Fragment getInstance(){
     return new CompareFragment();   }
   public CompareFragment() {   }
   @Override
   public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable
Bundle savedInstanceState) {
      View root = inflater.inflate(R.layout.compare_fragment, container, false);
      PetitionViewModelFactory singletonNameViewModelFactory = new
PetitionViewModelFactory(PetitionViewModel.getInstance());
      // завантаження моделі
      petitionModel = new
ViewModelProvider(getActivity(),singletonNameViewModelFactory).get(PetitionViewModel.class);
      adapterPetitions = new ArrayAdapter<>(this.getContext(),
           android.R.layout.simple_list_item_1, android.R.id.text1, new ArrayList<>());
      adapterNames = new ArrayAdapter<String>(this.getContext(), android.R.layout.simple_list_item_1,
           android.R.id.text1,new ArrayList<>());
      //  створення слухача зміни списку петицій для порівняння
petitionModel.getCompareList().observe(getViewLifecycleOwner(),dataLines -> {
        adapterPetitions.clear();
        if (dataLines!=null && dataLines.size()>0)
          adapterPetitions.addAll(dataLines);
        adapterPetitions.notifyDataSetChanged();
        getListViewSize(listViewPetitions);       });
      // створення слухача зміни списку підписників яких порівняли
      petitionModel.getNameList().observe(getViewLifecycleOwner(),names->{
        adapterNames.clear();
        if (names!=null && names.size()>0)
          adapterNames.addAll(names);
        adapterNames.notifyDataSetChanged();
        getListViewSize(listViewNames);       });
```

```
listViewPetitions = root.findViewById(R.id.listItem);
listViewNames = root.findViewById(R.id.compareList);
listViewPetitions.setAdapter(adapterPetitions);
listViewNames.setAdapter(adapterNames);
listViewPetitions.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
  @Override
  public boolean onItemLongClick(AdapterView<?> adapterView, View view, int i, long l) {
    if (i>=0){
      //при довгому натисканні видаалити позицію
      DataLine d = (DataLine) adapterPetitions.getItem(i);
      String num = d.getNum();
      petitionModel.removedDataLine(d);
      Toast.makeText(getActivity(), String.format("%s видалено", num ), Toast.LENGTH_SHORT).show();;
      adapterPetitions.notifyDataSetChanged();
      getListViewSize(listViewPetitions);    }
    return false;    }  });
// кнопка порівняння
ImageButton btCompare = root.findViewById(R.id.btCompare);
btCompare.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    MyAsyncTask my = new MyAsyncTask(getActivity());
    my.execute();    }    });
// кнопка експорту
ImageButton btExport = root.findViewById(R.id.btExport);
btExport.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    if (checkW()){ //  если есть права на запись
      if (petitionModel.getNameList().getValue().size()>0){  //и есть что сохранять
        SaveTask saveTask = new SaveTask(); //создадим задачу и запустим
        saveTask.execute();}
      else {
        Toast.makeText(getActivity(), "Немає чого зберігати", Toast.LENGTH_SHORT).show();;  }  } }
});
// кнопка очищення
ImageButton btClear =root.findViewById(R.id.btClear);
btClear.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
```

```
        clearMessages(); }    });
   return root;   }
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);   }
// очищення списків
public  void clearMessages() {
   petitionModel.clearNames();
   petitionModel.clearCompareList();   }

// пошук збігів
private class MyAsyncTask extends AsyncTask<Void, Void, Void> {
   ProgressDialog pd;
   public MyAsyncTask(Context ctx) {
      this.pd = new ProgressDialog(ctx);      }
   @Override
   protected void onPreExecute() {
      super.onPreExecute();
      pd = new ProgressDialog(getActivity());
      pd.setTitle("Зачекайте");
      pd.setMessage("Порівнюю!");
      pd.setIndeterminate(false);
      pd.setProgressStyle(ProgressDialog.STYLE_SPINNER);
      pd.setCancelable(false);
      pd.show();      }
   @Override
   protected Void doInBackground(Void... voids) {
      final Set<DataLine> compareList = petitionModel.getCompareList().getValue();
      // створення списку унікальних петицій
      Set<String> pet =new HashSet<>();
      for(DataLine d:compareList){
         pet.add(d.getNum());         }
      // кількість петицій
      final int size = pet.size();
      String tag = "COMPARE_TASK";
      Log.v(tag,String.format("size %d",size));
      // якщо більше однох петиції – порвняти
      if (size>1){
         Log.v(tag,String.format("size %d",size));
         // карта для порівнянн – підписник та кількість збігів
```

```
        Map<String,Integer> maps = new HashMap<>();
        SignatureDao dao = App.getInstance().getDatabase().signatureDao();
        for(DataLine c : compareList){
          List<String> signatures =  dao.getSignatureName(c.getUuid(),c.getLastUpd());
          Log.v(tag,String.format("load %s %d",c.getNum(),signatures.size()));
          for(String  s:signatures){
            if (!maps.containsKey(s))
              maps.put(s,0);
            int count = maps.get(s);
            count++;
            maps.put(s,count);      }         }
        Log.v(tag,"map created");
        Set<String> names = new TreeSet<>();
        for(Map.Entry<String,Integer> entry: maps.entrySet()){
          // оберати тільки тих хто зустрічається стільки ж разів, скільки є петицій
if (entry.getValue()==size){
            names.add(entry.getKey());  }    }
        if (names.isEmpty()) names.add("Співпадінь немає");
        else  {
            int count = names.size();
            names.add(String.format(" Всього записів: %d",count));           }
          petitionModel.addNames(names);         }
        return null;      }
      @Override
      protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
        pd.dismiss();      }   }
    // втсановлення висоти списку
    public static void getListViewSize(ListView myListView) {
      ListAdapter myListAdapter = myListView.getAdapter();
      if (myListAdapter == null) {
        // якщо нема адаптера –-нічого не робити
        return;      }
      int totalHeight = 0;
      for (int size = 0; size < myListAdapter.getCount(); size++) {
        View listItem = myListAdapter.getView(size, null, myListView);
        listItem.measure(0, 0);
        totalHeight += listItem.getMeasuredHeight();      }
      ViewGroup.LayoutParams params = myListView.getLayoutParams();
      params.height = totalHeight + (myListView.getDividerHeight() * (myListAdapter.getCount() - 1));
```

```
    myListView.setLayoutParams(params);
    Log.i("height of listItem:", String.valueOf(totalHeight));    }
// завдання збереження результату порівняння
class SaveTask extends AsyncTask<Void, Void, Void> {
    SimpleDateFormat sdf;
    String fileName;
    File file;
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        sdf = new SimpleDateFormat("dd_MM_yyyy_HH_mm_ss");
        fileName = new String("backup_").concat(sdf.format(new Date())).concat(".csv");       }
    @Override
    protected Void doInBackground(Void... params) {
        try {
            // зберегти в папку завантажень
            file = new File(Environment.getExternalStoragePublicDirectory(
                    Environment.DIRECTORY_DOWNLOADS),fileName);
            StringBuilder str = new StringBuilder();
            Iterator it  = petitionModel.getCompareList().getValue().iterator();
            while (it.hasNext()){
                DataLine c = (DataLine) it.next();
                str.append(c.toString());
                str.append("\n");              }
            it=petitionModel.getNameList().getValue().iterator();
            while (it.hasNext()){
                String name = (String) it.next();
                str.append(name);
                str.append(";\n");           }
            FileOutputStream fos=new FileOutputStream(file);
            byte[] buffer = str.toString().getBytes();
            fos.write(buffer, 0, buffer.length);
            fos.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();          }
        return null;       }
    @Override
    protected void onPostExecute(Void result) {
```

```
        super.onPostExecute(result);
        Toast.makeText(getContext(),"save "+file.getName(),Toast.LENGTH_SHORT).show();    }   }
   // перевірка дозволу
   private boolean checkW()   {
      int permissionStatusW = ContextCompat.checkSelfPermission(getContext(),
Manifest.permission.WRITE_EXTERNAL_STORAGE);
      if (permissionStatusW == PackageManager.PERMISSION_GRANTED)
      {return true;}
      else {
         ActivityCompat.requestPermissions(getActivity(), new String[]
{Manifest.permission.WRITE_EXTERNAL_STORAGE},
               1);      }
      return false;
   }
}
```