МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Бакалаврська кваліфікаційна робота

на тему: Розробка мобільного додатку для прогнозу погоди

Виконав студент 4 курсу групи К-45
Спеціальності 122 комп'ютерні науки,
Дергачев Андрій Юрійович

Керівник к.т.н., доцент
Трегубова Ірина Анатоліївна

Консультант

Рецензент к.ф-м.н доцент
Вітавецька Лариса Анатоліївна

Одеса 2019

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

ART – Android Runtime – середа виконання Android-додатків, розробленная компанією Google як заміна Dalvik.

DAO – data access object – об'єкт що надає абстрактний інтерфейс до деяких видів баз даних.

HTML – HyperText Markup Language – стандартна мова розмітки для створення веб-сторінок і веб-додатків.

IDE – Integrated development environment – комплексне програмне рішення для розробки програмного забезпечення.

JVM – Java Virtual Machine – набір комп'ютерних програм та структур даних, що використовують модель віртуальної машини для виконання інших комп'ютерних програм.

MVP – Model – View – Presenter – шаблон проектування.

OHA – Open Handset Alliance – альянс 84 компаній по розробці відкритих стандартів для мобільних пристроїв.

REST – Representational State Transfer – підхід до архітектури мережевих протоколів.

UI – user interface – засіб взаємодії користувача з інформаційною системою.

UML – Unified Modeling Language – уніфікована мова моделювання.

URI – Uniform Resource Identifier – компактний рядок літер, який однозначно ідентифікує окремий абстрактний чи фізичний ресурс.

URL – Uniform Resource Locator – стандартизована адреса певного ресурсу

UX – User Experience – це те, що людина відчуває при користуванні продуктом.

WPF – Windows Presentation Foundation – графічна підсистема.

XML – eXtensible Markup Language – стандарт побудови мов розмітки ієрархічно структурованих даних

ВСТУП

Актуальність теми. Сьогодні мобільні телефони не є просто засобами зв'язку. Зараз ці пристрої замінюють собою всі гаджети, необхідні в повсякденному житті, наприклад: плеєр, навігатор, фотоапарат, відеокамера та інші. Смартфон надає величезну кількість функцій, які можуть бути корисні в будь–якій сфері діяльності людини.

З появою смартфонів, процес отримання доступу до необхідної інформації значно спростився. Тепер досить просто встановити додаток, що володіє необхідним набором функцій, отже, питання мобільних додатків є вкрай актуальним та має перспективи розвитку.

Метою роботи є розробка програми, яка буде надавати користувачеві інформацію про погодні умови в обраному регіоні, з такими функціональними можливостями:

- температура повітря протягом дня
- інформація про опади протягом дня
- швидкість вітру протягом дня
- вологість протягом дня
- хмарність впродовж дня
- атмосферний тиск протягом дня
- час світанку і заходу
- прогноз всієї цієї інформації протягом тижня

Для досягнення мети роботи необхідно вирішити наступні завдання:

- вивчити і описати предметну область;
- проаналізувати аналоги, які вже існують на ринку, взяти їх кращі якості;
- за допомогою UML–діаграм змоделювати додаток.

Вибрати такі інструменти:

- платформу для розробки;
- мова програмування;

- середовище розробки.

Розробити програму, що має наступний функціонал:

- температура повітря протягом дня;
- інформація про опади протягом дня;
- швидкість вітру протягом дня;
- вологість протягом дня;
- хмарність впродовж дня;
- атмосферний тиск протягом дня;
- час світанку і заходу;
- прогноз всієї цієї інформації протягом тижня.

Об'єктом роботи є розробка додатку для отримання користувачем актуальної інформації про стан погодних умов у потрібній для нього локації.

Предмет дослідження є теоретичні та практичні аспекти розробки мобільного додатку для прогнозування погоди.

Методи дослідження полягають в тому, що при виконанні цієї роботи були використані теоретичні, практичні та прикладні засоби розробки, з якими будуть пройдені всі етапи розробки програми.

Практична значущість отриманних результатів полягає у тому, що отримані висновки сприятимуть подальнішому розвитку мобільних додатків не лише в області прогнозування погоди, а також в інших галузях.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Історія спостереження за погодою

Метеорологія як наука виникла після винаходу в XVII столітті термометра Галілео Галілеєм і ртутного барометра Е. Торрічеллі. Пізніше в XVII столітті були винайдені гигрометр, дощомір, флюгер і анемометр.

Перше подобу мережі метеоспостережень виникло в Європі в 1654 році. Збір інформації здійснювався до 1667 року Академією дель Чіменто у Флоренції.

У Російській імперії, на відміну від Європи, тільки в кінці XVII століття почали замислюватися про будь – яких регулярних спостереженнях за погодою.

Регулярні спостереження за погодою першим спробував встановити цар Олексій Михайлович. За його повелінням із Європи привезли астрономічні інструменти і метеорологічні прилади, в тому числі винахід Еванджеліста Торрічеллі, учня Галілея – барометр. Однак призначений царем вести записи про погоду Афанасій Матюшкін, син дяка, інструментами не користувався і фіксував в «днювальний записках» в основному власні спостереження: коли почався дощ, коли закінчився, коли замерзла Москва–ріка, коли скресла крига.

Чималий внесок в зародження і подальший розвиток метеоспостережень вніс Петро I. За його наказом в кінці XVII століття почалися постійні спостереження за станом погоди. У 1715 році за його вказівкою було створено перший в Росії водомірний пост на Неві у Петропавлівській фортеці. 10 квітня 1722 року в Санкт–Петербурзі почалися систематичні спостереження за погодою. Записи вів віце –адмірал Корнеліус Крюйс.

Перший час запису були досить скупі на цікаву інформацію і виглядали приблизно так: «Апрель, 22, воскресенье. Поутру ветер норд – вест; вода також стоит, как выше упомянуто. Пасмурно и студено… в полдни ветр малый норд – вест и дождь после полудня. Тихо и красный день до самого вечера ».

Пізніше спостереження взяли понад науковий характер. Перші відомості про метеорологічних спостереженнях на Вятської землі відносяться до 1456 року «когда, … в весне, великий князь Московский послал рать на Вятку со князем Семеном Ряполовским и ничтоже успе воротились» ... «... тогда ж была буря велика, громна гроза, и солнце гинуло.». У Вятском Временнике (1905 р) подібні відомості є і за 1471, 1667, 1698 та інші роки. Справжні ж метеорологічні спостереження були розпочаті в 1786 році директором Вятського головного народного училища Ів. Стефановичем і проводилися по 1795 рік. Спочатку він проводив візуальні спостереження (відзначалися терміни випадання першого снігу, морози і ін.). У 1791р. він придбав термометри і зробив перші інструментальні спостереження за температурою повітря. На жаль спостереження ці були нерегулярними.

Після чималої перерви в Європі в 1723 році секретар Лондонського королівського товариства Джеймс Джурин розробив інструкцію зі спостереження за погодою, в якій наводилася форма стандартних вимірів, перелік необхідних приладів і опис методик вимірювання температури, атмосферного тиску, швидкості і напряму вітру. За його участю була організована друга мережа метеостанцій в Європі, яка проіснувала до 1735 року.

Приблизно в цей же час в Росії з'явилося перше подобу мережі метеостанцій для спостережень за погодою. Це було обумовлено розгорнулася в той час Великої Північної експедицією.

Інструкцію для спостерігачів написав Данило Бернуллі. За період з 1733 по 1744 рік по всієї Сибірі було створено 24 метеостанції.

У 1724 році була утворена перша в Російській імперії метеорологічна станція, а з грудня 1725 року за Академії наук стали проводитися спостереження за допомогою барометра і термометра.

У 1781 році в Мангеймі було засновано перше в світі метеорологічне суспільство. Воно забезпечувало спостерігачів в різних країнах світу однаковими приладами. За його програмі діяло 39 метеостанцій, розташованих від Кембриджа до Уралу. Їм було запропоновано встановити чотири моменти

проведення вимірювань в день: в 7, 11, 14 і 21 годину.

У 1802 році, незалежно один від одного, Жан – Батіст Ламарк і Люк Говард запропонували свої системи класифікації хмар. Однак термінологія Ламарка не ввійшла в науковий обіг, так як він використовував для її написання французьку мову. Говард ж використовував у своїй класифікації латинську мову. Саме Говард дав хмарам їх загальноприйняті назви, що використовуються і до цього дня.

Регулярні метеорологічні спостереження на В'ятці почалися з 1830 року (в м Слобідської (Никанор Кулёв, штатний наглядач повітового училища), в м Котельнич (учитель повітового училища Афанасій Суворов), м Вятка (старший учитель фізики і математики Вятської гімназії І. Наумов).

У 1835 році на сході Європейської території Росії з ініціативи професора Казанського університету Е. А. Кнорра, з дозволу академії наук і за підтримки А. Я. Купфера почали відкриватися перші метеорологічні станції. У підсумку в 1835. в В'ятці була відкрита метеостанція, першим спостерігачем якої був учитель математики А. П. Габов.

Спостереження проводилися в терміни 9, 12, 15 і 21 годин за тиском повітря, температурою по Реомюр, станом неба, опадами, по флюгеру визначався вітер.

Таким чином в метеорологічну історію Вятки золотими літерами вписується 1835 рік, так як спостереження на Вятської метеостанції проводилися вже за Інструкцією Академії наук систематично, в єдині терміни і за єдиними приладами.

Матеріали спостережень регулярно висилалися в Казанський університет і Головну фізичну обсерваторію в Петербурзі, де з 1860 року стали регулярно друкуватися в її «Записках».

У 1877 році відкривається перший водомірний пост на річці Вятка (м Вятка), організовані інструментальні гідрологічні спостереження. До 1900 року на р. Вятка було організовано ще дві посади (Слобідської та Котельнич) і два на р. Кама (Сарапул і Каракулино). Перші водомірні пости на великих

річках були відкриті для потреб судноплавства і належали в ті роки Міністерству шляхів сполучення.

У 1853 році було покладено початок першого в історії державного метеорологічного відомства – метеослужбі Великобританії. Відтепер всі капітани англійських судів повинні були вести спостереження за погодою з занесенням даних в спеціально розроблені таблиці. На узбережжі Великобританії, а також в деяких європейських країнах були створені 24 метеорологічні станції. Станції були з'єднані з центром служби погоди недавно винайденим телеграфом Морзе.

Поступове накопичення відомостей про погоду і клімат різних широт призвело до необхідності подальшої обробки метеоданих.

Перші синоптичні карти були опубліковані в Німеччині Брандесом ще в 1826 році. На цих вельми недосконалих картах ще не було ні контурів материків, ні будь–яких изолиний. В подальшому карти погоди епізодично складалися в багатьох країнах і поступово удосконалювалися.

Після знаменитої балаклавській бурі, що вибухнула на Чорному морі 14 листопада 1854 року, і потопивши 60 кораблів англо–французького флоту, який діяв проти Росії в період Кримської війни, директор Паризької обсерваторії Урбен Левер'є звернувся з проханням до знайомих європейським ученим надіслати йому зведення про стан погоди в період з 12 по 16 листопада. Коли зведення були отримані і дані нанесли на карту, стало ясно, що ураган, потопив кораблі в Чорному морі, можна було передбачити заздалегідь. У лютому 1855 р Леверье підготував доповідь Наполеону III про перспективи створення централізованої метеорологічної мережі спостережень. Цей висновок послужив поштовхом для організації збору метеорологічних даних і створення служби погоди в ряді країн.

В організації служби погоди, перш за все, був зацікавлений морський флот. Тому спочатку служба погоди створювалася в приморських країнах і першими синоптиками були моряки.

Офіційною датою початку служби погоди в Росії вважати 1 січня 1872

рік, коли в Головній фізичній обсерваторії, заснованої 1 квітня 1849 в Санкт–Петербурзі (нині «Головна геофізична обсерваторія» ім. А. І. Воєйкова (ГГО), почався регулярний випуск щоденного бюлетеня погоди. Однак в ГФО ще в 1856 році було розпочато прийом метеорологічних телеграм від 13 російських і 5 зарубіжних станцій. у 1864 році було опубліковано дослідження Ф. Міллера «про попередження бур, особливо про бурях, лютували з 1 по 4 грудня 1863 р . », а в 1867 році сел ано перший штормове оповіщення. Перше штормове попередження було дано в 1874 році. У 1889 році видано перший посібник з синоптичної метеорології М. М. Поморцева (1851–1916 рр.). Починаючи з 1890 р налагоджено регулярне попередження управлінь залізниць про заметілі і снігових заметах, що в кліматичних умовах Росії мало особливо важливе значення.

У 1873 р у Відні відбувся перший міжнародний метеорологічний конгрес, на якому були вироблені єдині терміни вимірювань, єдиний телеграфний код передачі метеосведеній.

Згідно архівних матеріалів за станом на 01.04.1898 року в Вятської губернії працювали 33 метеостанції. До кінця 1903 року – 40. Спостерігачам виплачували по 2–3 рубля на місяць, потім їх позбавили матеріальної підтримки, і станції стали закриватися. У 1913 році їх залишилося 19, а через 5–6 років, через революційних подій, – одна (Вятка).

У цей період цікавий факт заснування метеостанції Малковской Котельнічского повіту в 1913 році на кошти бідного селянина В. Краєва, «який віддав для цього все». Спостереження велися їм же. У 1919 році Країв був покликаний на службу в Червону Армію, але через 5 місяців від служби звільнений, як незамінний фахівець-метеоролог.

Під час першої світової війни 1914-1918 рр. обмін метеорологічною інформацією між країнами був порушений. Однак в невоєвавший Скандинавських країнах в цей період була створена досить густа мережа метеорологічних станцій, що дозволило складати більш докладні карти погоди. За цими картами вченим вдалося виявити фронтальні розділи між повітряними маса-

ми, а також зв'язати виникнення і розвиток циклонів з фронтами.

У Росії найбільш видатні дослідження циклонів, антициклонів, синоптичних умов небезпечних явищ і розробка прийомів прогнозу погоди були виконані П. І. Броуновим, Б. І.Срезневським і М. А. Рикачевим. Багато з цих дослідників зберегли своє значення до теперішнього часу.

Завдання, поставлені декретом Ради Народних Комісарів про організацію метеорологічної служби, підписаним В. І. Леніним в 1921 році, в цей період були істотно розширені.

У 1929 році організована єдина Гідрометеорологічна служба країни, організовані нові метеорологічні станції і підрозділи служби погоди.

Початок і розвиток аерологічних спостережень на В'ятці тісно пов'язане з діяльністю Вятської опорної метеорологічної станції, відкритої 1 жовтня 1921 року. З 1 вересня 1923 на В'ятці стали вестися регулярні аерологічні спостереження.

Винахід П. А. Молчановим радіозонда в 1930 році відкрило нову епоху в розвитку синоптичної метеорології. Вивчення вертикального будови атмосфери стало можливим не непрямими методами (за даними наземних спостережень), а за результатами радиозондирования атмосфери.

Була створена мережа аерологічних станцій і почалося складання перших карт баричної топографії в наукових цілях. В оперативних цілях, в СРСР і ряді інших країн, карти баричної топографії стали застосовувати з 1937 року. Однак досить часту світову мережу аерологічних станцій, з яких вони запускалися, вдалося створити лише після Другої світової війни.

1 січня 1930 року в Москві було відкрито Центральне бюро погоди СРСР (ЦПП), перетворене згодом в Центральний інститут погоди (нині Гідрометцентр Росії). Прогнози погоди стали більш конкретними, детальними. Широко розгорнулося метеорологічне забезпечення авіації.

У цей період почалося систематичне вивчення Арктики. У 1937 році була створена перша дрейфу станція «Північний полюс».

У 1933 році при Вятської метеостанції була організована гідрологічна

станція, почалося інтенсивне вивчення режиму малих річок, в основному для будівництва гідроелектростанцій в сільській місцевості. До 1941 року було відкрито 32 водомірних поста. З 1935 року на всіх метеостанціях введені снігозйомки. Станом на 19.11.1939 року мережа метеорологічних станцій Кіровської області налічувала 68 підрозділів.

У 1939 році в Кірові для потреб авіації була створена метеорологічна станція, перетворена в 1941 році в авіаметстанцію. Першими начальниками авіаметстанції був А.С.Флегонтов і Ананьін.

В період Великої Вітчизняної війни служба погоди була воєнізована. У 1943 році в Кірові був організований пункт вертикального радиозондирования атмосфери. Випуски радіозондов почалися 13 липня 1943 року.

Незважаючи на важкі наслідки війни в СРСР синоптичні дослідження атмосферних процесів, успішно розпочаті в 30–і роки, активно тривали. Великий розвиток отримала регіональна синоптика та авіаційна метеорологія.

Запуск в Радянському Союзі першого штучного супутника Землі 4 жовтня 1957 року відкрив виняткові принципові можливості отримання різного роду нової інформації, в тому числі метеорологічної.

В 50-ті 60-ті роки активно розвивалася мережа пунктів метеорологічних спостережень не тільки в країнах Європи, а й в Росії. У 1966 році запроваджуються єдині восьмісрочние спостереження за погодою (00, 03, 06, 09, 12, 15, 18, 21 годині).

У 70-ті роки – почалося масове розвиток мережі пунктів гідрологічних спостережень на великих річках і озерах[1][1)].

В кінці 60-х років в Радянському Союзі і США були створені метеорологічні космічні системи. Це дозволило більш об'єктивно проводити синоптичний аналіз, особливо на території, слабо освітленій метеорологічними даними, своєчасно виявляти особливо небезпечні тропічні циклони і т.д.

Широке застосування отримали метеорологічні радіолокатори. З поча-

---

[1)] [1] Метеорологія і кліматологія як науки. URL: http://www.geograf.com.ua (дата звернення 13.04.2019)

тком їх застосування дослідникам вдалося більш детально вивчити фізичні процеси, що відбуваються в атмосфері.

Всі ці досягнення дозволили поліпшити якість короткострокових прогнозів погоди і підвищити їх виправданість.

1.2 Аналіз існуючих програмних засобів

Додаток, що розробляється, не є унікальним у своєму роді – існують аналоги, які є фундаментом для вдосконалення та виправлення існуючих недоліків.

1.3 Програмний аналог «AccuWeather»

«AccuWeather» – найпопулярніший додаток в Google Play , що надає інформацію про погоду[2][1)], рис. 1 Він володіє всім функціоналом, який може бути корисним користувачеві, однак має ряд істотних недоліків.

Рисунок 1 – Сторінка додатка в Google Play

[1)] [2] Сторінка додатка AccuWeather в Google Playstore. URL: https://play.google.com/details?id=com.accuweather (дата звернення 15.04.2019)

Зовнішній вигляд програмного засобу приведений на рис. 2 – 4.

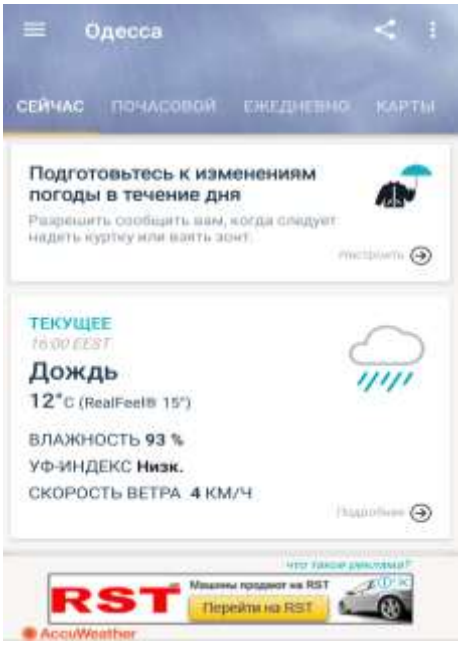

Рисунок 2 – Зовнішній вигляд додатку «AccuWeather».

Погода зараз

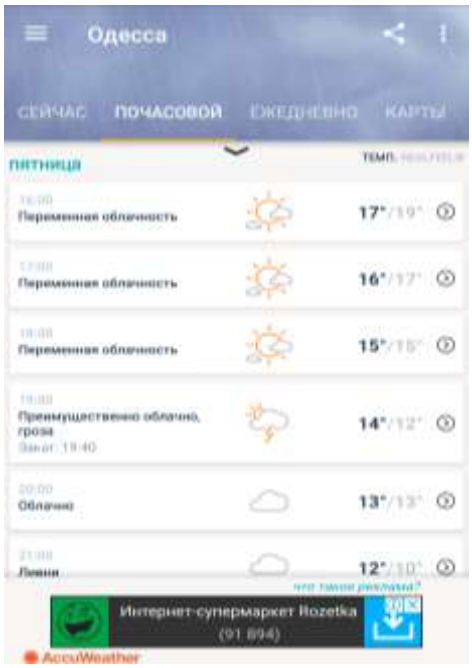

Рисунок 3 – Зовнішній вигляд додатку «AccuWeather».

Погода сьогодні

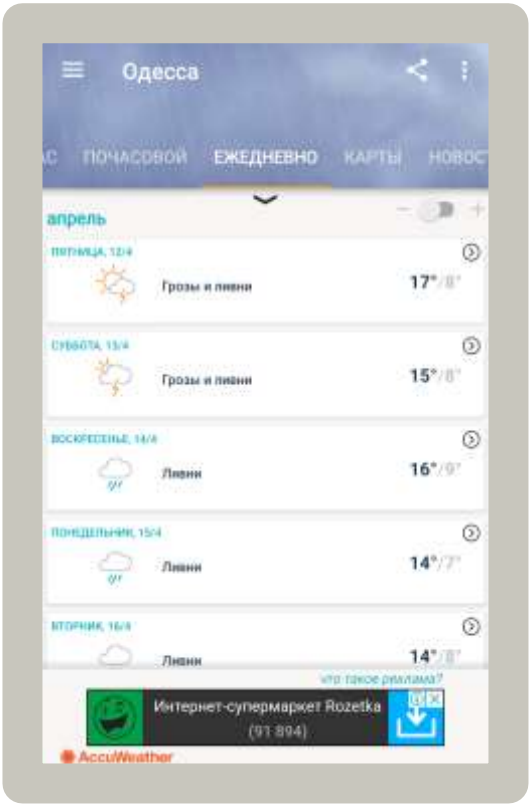

Рисунок 4 – Зовнішній вигляд додатку «AccuWeather».
Погода на тиждень

1.4 Недоліки додатка «AccuWeather»

Незважаючи на те, що «AccuWeather» є найпопулярнішим додатком для прогнозу погоди, він не позбавлений недоліків, а саме:

- додаток займає багато пам'яті на пристрої користувача, близько 100 мб;
- інтерфейс програми дуже перевантажений, в ньому важко розібратися і він не завжди працює швидко і гладко;
- на всіх екранах програми показується реклама.

Додаток розробляється з акцентом на виправлення перерахованих недоліків.

2 ОПИС ПРОГРАМНИХ ЗАСОБІВ РОЗРОБКИ

Android (вимов.Андроїд) – операційна система і платформа для мобільних телефонів та планшетних комп'ютерів, створена компанією Google на базі ядра Linux. Підтримується альянсом Open Handset Alliance (OHA).Хоча Android базується на ядрі Linux, він стоїть дещо осторонь Linux–спільноти та Linux–інфраструктури. Базовим елементом цієї операційної системи є реалізація Dalvik віртуальної машини Java, і все програмне забезпечення і застосування спираються на цю реалізацію Java.

У 84 % смартфонів, проданих у 3-ому кварталі 2017 року, була встановлена операційна система Android.

У березні 2017 року ОС Android стала найпопулярнішою ОС, з якої виходили в інтернет. Так з 37,93 % користувачів заходили в інтернет із Android'a, а з Windows лише 37,91 % користувачів. В Азії показники ще вищі – 52,2 % і 29,2 % відповідно.

Додатки під операційну систему Android є програмами в нестандартному байт–коді для віртуальної машини Dalvik, для них був розроблений формат настановних пакетів .APK. Для роботи над додатками є безліч бібліотек: Bionic (бібліотека стандартних функцій, несумісна з glibc); мультимедійні бібліотеки на базі PacketVideo OpenCORE (підтримують такі формати, як MPEG–4, H.264, MP3, AAC, AMR, JPEG і); SGL (движок двомірної графіки); OpenGL ES 1.0 ES 2.0 (движок тривимірної графіки); Surface Manager (забезпечує для додатків доступ до 2D / 3D); WebKit (готовий движок для веб–браузера; обробляє HTML, JavaScript); FreeType (движок обробки шрифтів); SQLite (легка СКБД, доступна для всіх додатків); SSL (протокол, що забезпечує безпечну передачу даних по мережі).

У порівнянні зі звичайними додатками Linux, додатки Android підкоряються додатковим правилам : Content Providers – обмін даними між додатками; Resource Manager – доступ до таких ресурсів, як файли XML, PNG, JPEG; Notification Manager – доступ до рядка стану; Activity Manager – уп-

равління активними додатками[3][1].

Google пропонує для вільного скачування інструментарій для розробки (Software Development Kit), який призначений для x86–машин під операційними системами Linux, macOS (10.4.8 або вище), Windows XP, Windows Vista і Windows 7. Для розробки потрібно JDK 5 або більше новий.

Розробку додатків для Android можна вести на мові Java (не нижче Java 1.5). Існує плагін для Eclipse – Android Development Tools (ADT), призначений для Eclipse версій 3.3–3.7. Також існує плагін для IntelliJ IDEA, який полегшує розробку Android–додатків, і для середовища розробки NetBeans IDE, який, починаючи з версії NetBeans 7.0, перестав бути експериментальним, хоч поки і не є офіційним. Крім того, існує Motodev Studio for Android – комплексна середовище розробки на базі Eclipse, що дозволяє працювати безпосередньо з Google SDK.

У 2009 році на додаток до ADT був опублікований Android Native Development Kit (NDK) – пакет інструментаріїв і бібліотек, що дозволяє реалізувати частину програми на мові C / C ++. NDK рекомендується використовувати для розробки ділянок коду, критичних до швидкості.

У 2013 році Google представила нову середу розробки Android Studio, засновану на IntelliJ IDEA від JetBrains.

У 2013 році відбувся реліз Embarcadero RAD Studio – XE5. Можливість розробки нативних додатків для платформи Android.

Процес створення Android програми не вимагає додаткових пристроїв, крім, власне, Android–пристрої (в принципі, можна обійтися і емулятором).

В Android 4.4 з'явилася можливість змінити віртуальну машину Dalvik на ART (Android Runtime). ART відрізняється підвищеною швидкістю завантаження програми. Опрацьовано механізм оптимізації пам'яті.

В Android 5 перемальований дизайн, який базується на концепції

---

[1] [3] Гріффітс Дон, Гріффітс Девід Head First. Програмування для Android: СПб.: Питер, 2016.704 с.

Material Design, доданий режим енергозбереження Project Volta, вибір машини пропав, замість Dalvik стала використовуватися ART.

В Android 6 з'явився інтелектуальний режим витрати енергії Doze і заборона виходу в Інтернет і роботи в тлі давно не використовуваних додатків App Standby.

2.1 Особливості мови програмування Java

Java (вимовляється Джава) – об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт–код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи[4][1)].

«Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано.

Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно–орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині.

Передусім Java розроблялась як платформо–незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

---

1) [4] Шилдт Гебрерт. Java 8 полное руководство:М:Вильямс, 2016. 720 с.

Java вплинула на розвиток J++, що розроблялась компанією «Microsoft». Роботу над J++ було зупинено через судовий позов «Sun Microsystems», оскільки ця мова програмування була модифікацією Java. Пізніше в новій платформі «Microsoft» .NET випустили J#, щоб полегшити міграцію програмістів J++ або Java на нову платформу. З часом нова мова програмування C# стала основною мовою платформи, перейнявши багато чого з Java. J# востаннє включався в версію Microsoft Visual Studio 2005. Мова сценаріїв JavaScript має схожу із Java назву і синтаксис, але не пов'язана із Java.

У створенні мови програмування Java було п'ять початкових цілей:

- синтаксис мови повинен бути «простим, об'єктно–орієнтовним та звичним»;
- реалізація має бути «безвідмовною та безпечною»;
- повинна зберегтися «незалежність від архітектури та переносність»;
- висока продуктивність виконання;
- мова має бути «інтерпретованою, багатонитевою, із динамічним зв'язуванням модулів».

Платформа Java: під «незалежністю від архітектури» мається на увазі те, що програма, написана на мові Java, працюватиме на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції.Цього можна досягти, компілюючи початковий Java код у байт-код, який є спрощеними машинними командами.

Потім програму можна виконати на будь-якій платформі, що має встановлену віртуальну машину Java, яка інтерпретує байткод у код, пристосований до специфіки конкретної операційної системи і процесора. Зараз віртуальні машини Java існують для більшості процесорів і операційних систем.

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та роботу з мережами. У деяких версіях задля збільшення продуктивності JVM байт–код можна компілювати у машинний код до або під час виконання програми.

Основна перевага використання байт-коду – це портативність. Тим не менш, додаткові витрати на інтерпретацію означають, що інтерпретовані програми будуть майже завжди працювати повільніше, ніж скомпільовані у машинний код, і саме тому Java одержала репутацію «повільної» мови. Проте, цей розрив суттєво скоротився після введення декількох методів оптимізації у сучасних реалізаціях JVM.

Одним із таких методів є just–in–time компіляція (JIT, що перетворює байт–код Java у машинний під час першого запуску програми, а потім кешує його. У результаті така програма запускається і виконується швидше, ніж простий інтерпретований код, але ціною додаткових витрат на компіляцію під час виконання.

Складніші віртуальні машини також використовують динамічну рекомпіляцію, яка полягає в тому, що віртуальна машина аналізує поведінку запущеної програми й вибірково рекомпілює та оптимізує певні її частини. З використанням динамічної рекомпіляції можна досягти більшого рівня оптимізації, ніж за статичної компіляції, оскільки динамічний компілятор може робити оптимізації на базі знань про довкілля періоду виконання та про завантажені класи.

До того ж він може виявляти так звані гарячі точки – частини програми, найчастіше внутрішні цикли, які займають найбільше часу при виконанні. JIT–компіляція та динамічна рекомпіляція збільшує швидкість Java-програм, не втрачаючи при цьому портативності.

Існує ще одна технологія оптимізації байткоду, широко відома як статична компіляція, або компіляція ahead–of–time (AOT). Цей метод передбачає, як і традиційні компілятори, безпосередню компіляцію у машинний код.

Це забезпечує хороші показники в порівнянні з інтерпретацією, але за рахунок втрати переносності: скомпільовану таким способом програму можна запустити тільки на одній, цільовій платформі.

Швидкість офіційної віртуальної машини Java значно покращилася з моменту випуску ранніх версій, до того ж, деякі випробування показали, що

продуктивність JIT-компіляторів у порівнянні зі звичайними компіляторами у машинний код майже однакова. Проте ефективність компіляторів не завжди свідчить про швидкість виконання скомпільованого коду, тільки ретельне тестування може виявити справжню ефективність у даній системі.

Об'єктність: на противагу C++, Java є більш об'єктно-орієнтованою. Всі дані і дії групуються в класи об'єктів. Виключенням з повної об'єктності (як скажімо в Smalltalk) є примітивні типи (int, float тощо).

Це було свідомим рішенням проектувальників мови задля збільшення швидкості. Через це Java не вважається повністю об'єктно-орієнтовною мовою.

У Java всі об'єкти є похідними від головного об'єкта (він називається просто Object), з якого вони успадковують базову поведінку і властивості[5][1)].

Хоча у C++ вперше стало доступне множинне успадкування, але у Java можливе тільки одинарне успадкування, завдяки чому виключається можливість конфліктів між членами класу (методи і змінні), які успадковуються від базових класів.

Безпека: у намірах проектувальників Java мала замінити C++ – об'єктного наступника мови C. Проектувальники почали з аналізу властивостей C++, які є причиною найбільшого числа помилок, щоби створити просту, безпечну і безвідмовну мову програмування.

В Java існує система винятків або ситуацій, коли програма зустрічається з неочікуваними труднощами, наприклад:

- операції над елементом масиву поза його межами або над порожнім елементом;
- читання з недоступного каталогу або неправильної адреси URL;
- ввід недопустимих даних користувачем.

Одна з особливостей концепції віртуальної машини полягає в тому, що по-

---

[1)] [5] Екель Брюс. Філософія Java. 4-е полное изд.: СПб.: Питер, 2015. 1168 с.

милки (виключення) не призводять до повного краху системи.

Крім того, існують інструменти, які «приєднуються» до середовища періоду виконання і кожен раз, коли сталося певне виключення, записують інформацію з пам'яті для зневадження програми.

Ці інструменти автоматизованої обробки виключень надають основну інформацію щодо виключень в програмах на Java.

Проте мову програмування Java не рекомендується використовувати в системах, збій в роботі яких може призвести до смерті, травм чи значних фізичних ушкоджень (наприклад, програмне забезпечення для керування атомними електростанціями, польотами, систем життєзабезпечення чи систем озброєння) через ненадійність програм, написаних на мові програмування Java (пункт ліцензії Microsoft 7.7.h.).

Автоматичне керування пам'яттю: java використовує автоматичний збирач сміття для керування пам'яттю під час життєвого циклу об'єкта. Програміст вирішує, коли створювати об'єкти, а віртуальна машина відповідальна за звільнення пам'яті після того, як об'єкт стає непотрібним. Коли до певного об'єкта вже не залишається посилань, збирач сміття може автоматично прибирати його із пам'яті.

Проте, витік пам'яті все ж може статися, якщо код, написаний програмістом, має посилання на вже непотрібні об'єкти, наприклад на об'єкти, що зберігаються у діючих контейнерах.

Збирання сміття дозволене у будь-який час. В ідеалі воно відбувається під час бездіяльності програми. Збірка сміття автоматично форсується при нестачі вільної пам'яті в купі для розміщення нового об'єкта, що може призводити до кілька секундного зависання.

Тому існують реалізації віртуальної машини Java з прибиральником сміття, спеціально створеним для програмування систем реального часу.

Java не має підтримки вказівників у стилі C/C++.

Це зроблено задля безпеки й надійності, аби дозволити збирачу сміття переміщувати вказівникові об'єкти.

Програми на Java утворені з визначень класів та інтерфейсів. Класи містять змінні та константи, які утримують дані, методи, які виконують дії, та конструктори, які створюють екземпляри класів – об'єкти.

Дані можуть мати простий тип (наприклад байт, ціле число, символ) або бути посиланням на об'єкт. Мова Java є статично типізованою.

2.2 Інструменти Android SDK

Android SDK – універсальний засіб розробки мобільних додатків для операційної системи Android.

Відмінною рисою від звичайних редакторів для написання кодів є наявність широких функціональних можливостей, що дозволяють запускати тестування і налагодження вихідних кодів, оцінювати роботу програми в режимі сумісності з різними версіями ОС Android і спостерігати результат в реальному часі (опціонально). Підтримує велику кількість мобільних пристроїв, серед яких виділяють: мобільні телефони, планшетні комп'ютери, розумні окуляри (в тому числі Google Glass), сучасні автомобілі з бортовими комп'ютерами на ОС Андроїд, телевізори з розширеним функціоналом, особливі види наручних годинників і багато інших мобільні гаджети, габаритні технічні пристосування.

2.3 Середовище розробки «Android Studio»

Android Studio – інтегроване середовище розробки (IDE) для платформи Android, представлене 16 травня 2013 року на конференції Google I/O менеджером по продукції корпорації Google – Еллі Паверс (англ. Ellie Powers). 8 грудня 2014 року компанія Google випустила перший стабільний реліз Android Studio 1.0.Android Studio прийшло на зміну плагіну ADT для платформи Eclipse.

Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA

Community Edition, що розвивається компанією JetBrains.

Android Studio розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0.

Бінарні складання підготовлені для Linux (для тестування використаний Ubuntu), Mac OS X і Windows. Середовище надає засоби для розробки застосунків не тільки для смартфонів і планшетів, але і для носимих пристроїв на базі Android Wear, телевізорів (Android TV), окулярів Google Glass і автомобільних інформаційно–розважальних систем (Android Auto).

Для застосунків, спочатку розроблених з використанням Eclipse і ADT Plugin, підготовлений інструмент для автоматичного імпорту існуючого проекту в Android Studio.

Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android.

У тому числі у середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування застосунків, що працюють на пристроях з екранами різної роздільності (планшети, смартфони, ноутбуки, годинники, окуляри тощо). Крім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька додаткових функцій, таких як нова уніфікована підсистема складання, тестування і розгортання застосунків, заснована на складальному інструментарії Gradle і підтримуюча використання засобів безперервної інтеграції.

Для прискорення розробки застосунків представлена колекція типових елементів інтерфейсу і візуальний редактор для їхнього компонування, що надає зручний попередній перегляд різних станів інтерфейсу застосунку (наприклад, можна подивитися як інтерфейс буде виглядати для різних версій Android і для різних розмірів екрану)[6][1)].

Для створення нестандартних інтерфейсів присутній майстер створення власних елементів оформлення, що підтримує використання шаблонів.

---

1) [6] Гріффітс Дон, Гріффітс Девід Head First. Програмування для Android. СПб.: Питер, 2016. 704 с.

У середовище вбудовані функції завантаження типових прикладів коду з GitHub.

Робоча область середовища розробки Android Studio приведена на рис. 5:

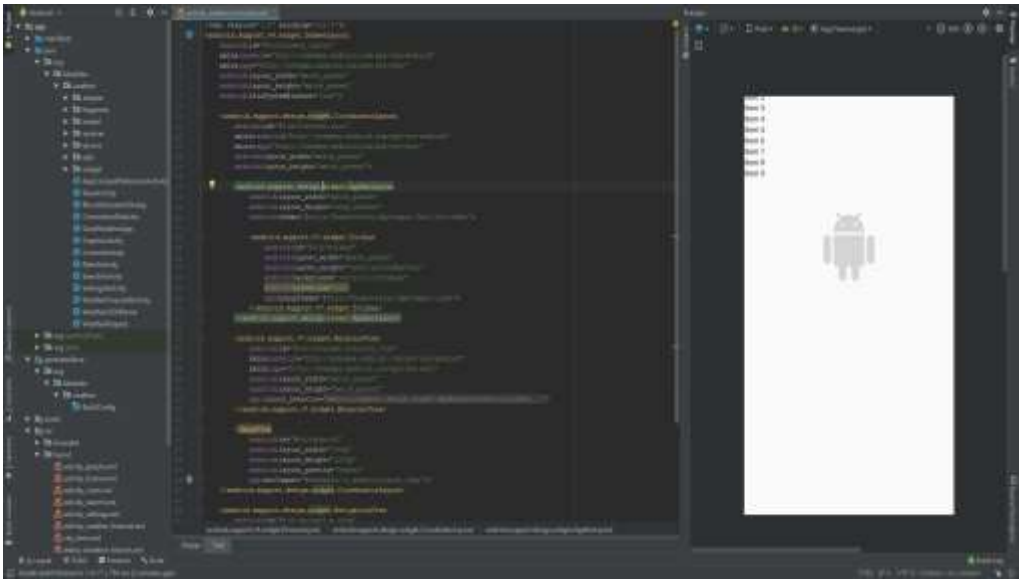

Рисунок 5 – Робоча область середовища розробки Android Studio

До складу також включені пристосовані під особливості платформи Android розширені інструменти рефакторингу, перевірки сумісності з минулими випусками, виявлення проблем з продуктивністю, моніторингу споживання пам'яті та оцінки зручності використання.

У редактор доданий режим швидкого внесення правок. Система підсвічування, статичного аналізу та виявлення помилок розширена підтримкою Android API. Інтегрована підтримка оптимізатора коду ProGuard. Вбудовані засоби генерації цифрових підписів.

Надано інтерфейс для управління перекладами на інші мови.

Деякі особливості будуть пізніше розгорнуті для користувачів так як програмне забезпечення розвивається; наразі, передбачені такі функції:

– живі макети (layout): редагувальник WYSIWYG – живе кодування подання (rendering) програми в реальному часі;
– консоль розробника: підказки по оптимізації, допомога по перек-

ладу, стеження за напрямком, агітації та акції – метрики Google аналітики;

- резерви бета релізів та покрокові релізи;
- базування на Gradle;
- android-орієнтований рефакторинг та швидкі виправлення;
- lint утиліти для охоплення продуктивності, юзабіліті, сумісності версій та інших проблем;
- використання можливостей ProGuard та підписів до програм;
- шаблони для створення поширених Android дизайнів та компонентів;
- багатий редактор макетів (layouts) що дозволяє користувачам перетягнути і покласти (drag–and–drop) компоненти користувацького інтерфейсу, як варіант, переглянути одночасно макети (layouts) на різних конфігураціях екранів.

2.4 Емулятор пристрою

Емуляція дозволяє виконувати комп'ютерну програму на платформі (комп'ютерній архітектури та/або операційній системі), відмінній від тієї, для якої вона була написана в оригіналі. Емуляцією також називають сам процес цього виконання. На відміну від симуляції, яка лише відтворює поведінку програми, при емуляції ставиться мета точного моделювання стану імітованої системи, для виконання оригінального машинного коду.

При використанні мов високого рівня, іноді в цілях збереження швидкодії виконуваної програми, замість емуляції роблять портування програм в нове середовище. У цьому випадку проводиться переписування заново апаратнозалежних ділянок коду.

Одне з популярних застосувань емуляції – виконання на персональному комп'ютері ігор, написаних для гральних автоматів або ігрових приставок.

Теоретично, згідно з тезисом Черча-Тюринга, будь-яке операційне се-

редовище може бути емульоване в будь-якому іншому середовищі.

На практиці, однак, зустрічається ряд труднощів; зокрема, точна поведінка емульованої системи часто не документована і має бути досліджена і визначена за допомогою зворотної розробки.

Достатньо повна емуляція деякої апаратної платформи вимагає граничної точності, до рівня окремих тактових циклів, недокументованих особливостей і навіть помилок реалізації. Це особливо важливо для таких моделей класичних домашніх машин, як Commodore 64, програмне забезпечення яких значною мірою залежить від програмістських рішень. Вибір конкретного рішення відбувається з метою оптимізації (за розміром або швидкістю виконання програми), застосовуваної, наприклад програмістами ігор. Такі програми досить часто бувають засновані на недокументовані можливості процесора або операційної системи.

На противагу цьому, на деяких інших платформах досить мало використовувався прямий доступ до апаратного забезпечення. У цьому випадку виявляється достатнім забезпечити певний рівень сумісності, що забезпечує трансляцію системних викликів емуліруемой системи в виклики працюючої системи.

Зазвичай, емулятор складається з декількох модулів, що відповідають за різні підсистеми емульованого комп'ютера.

Частіше за все, емулятор складається з:

– емулятора центрального процесора;

– модуля підсистеми пам'яті що емулює постійну і оперативну пам'ять;

– модуля або модулів емуляції різних пристроїв вводу/виводу.

Системна шина, як правило, не емулюєтеся, з причин спрощення або підвищення продуктивності, та віртуальна периферія звертається безпосередньо до модуля ЦП та модуля пам'яті. Робоча область емуляторна Android Studio приведена на рис. 6:

Рисунок 6 – Робоча область середовища розробки Android Studio

2.5 Сервіс який надає інформацію про погоду

Для отримання інформації про погоду, я буду використовувати сервіс, який надає інформацію про погоду посрведством API.

Прикладний програмний інтерфе́йс (інтерфейс програмування застосунків, інтерфейс прикладного програмування) (англ. Application Programming Interface, API) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено – це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб–базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

Для доступу до сервісу я буду використовувати http запроси за допомогою Rest архітектури.

HTTP – протокол передачі даних, що використовується в комп'ютерних

мережах. Назва скорочена від Hyper Text Transfer Protocol, протокол передачі гіпер–текстових документів

HTTP належить до протоколів моделі OSI 7-го прикладного рівня. Основним призначенням протоколу HTTP є передача веб–сторінок (текстових файлів з розміткою HTML), хоча за допомогою нього успішно передаються і інші файли, які пов'язані з веб–сторінками (зображення і застосунки), так і не пов'язані з ними (у цьому HTTP конкурує з складнішим FTP).

HTTP припускає, що клієнтська програма – веб-браузер – здатна відображати гіпертекстові веб-сторінки та файли інших типів у зручній для користувача формі. Для правильного відображення HTTP дозволяє клієнтові дізнатися мову та кодування символів веб-сторінки й/або запитати версію сторінки в потрібних мові/кодуванні, використовуючи позначення із стандарту MIME.

HTTP – протокол прикладного рівня, схожими на нього є FTP і SMTP.

Обмін повідомленнями йде за звичайною схемою «запит–відповідь». Для ідентифікації ресурсів HTTP використовує глобальні URI. На відміну від багатьох інших протоколів, HTTP не зберігає свого стану. Це означає відсутність збереження проміжного стану між парами «запит–відповідь». Компоненти, що використовують HTTP, можуть самостійно здійснювати збереження інформації про стан, пов'язаний з останніми запитами та відповідями. Браузер, котрий посилає запити, може відстежувати затримки відповідей. Сервер може зберігати IP-адреси та заголовки запитів останніх клієнтів. Проте, згідно з протоколом, клієнт та сервер не мають бути обізнаними з попередніми запитами та відповідями, у протоколі не передбачена внутрішня підтримка стану й він не ставить таких вимог до клієнта та сервера.

Кожен запит/відповідь складається з трьох частин:

– стартовий рядок;
– заголовки;
– тіло повідомлення, що містить дані запиту, запитаний ресурс або опис проблеми, якщо запит не виконано.

Стартові рядки розрізняються для запиту й відповіді. Рядок запиту виглядає так:

‹Метод› ‹URI› HTTP/‹Версія› , де ‹Метод› можливо:

GET – запитує вміст вказаного ресурсу. Запитаний ресурс може приймати параметри (наприклад, пошукова система може приймати як параметр шуканий рядок). Вони передаються в рядку URI (наприклад: http://www.example.net/resource?param1=value1&param2=value2).

Згідно зі стандартом HTTP, запити типу GET вважаються ідемпотентними – багатократне повторення одного і того ж запиту GET повинне приводити до однакових результатів (за умови, що сам ресурс не змінився за час між запитами). Це дозволяє кешувати відповіді на запити GET.

Якщо назва ресурсу не вказана (у URI наявні лише схема та доменне ім'я), то веб-сервер повертає індекс директорії веб-сервера.

POST – передає призначені для користувача дані (наприклад, з HTML форми) заданому ресурсу. Наприклад, в блогах відвідувачі зазвичай можуть вводити свої коментарі до записів в HTML-форму, після чого вони передаються серверу методом POST, і він поміщає їх на сторінку. При цьому передані дані (у прикладі з блогами – текст коментаря) включаються в тіло запиту.

На відміну від методу GET, метод POST не вважається ідемпотентним, тобто багатократне повторення одних і тих же запитів POST може повертати різні результати (наприклад, після кожного відправлення коментаря з'являтиметься одна копія цього коментаря). PUT – завантажує вказаний ресурс на сервер. DELETE – видаляє вказаний ресурс.

REST (скор. англ. Representational State Transfer, «передача репрезентативного стану») – підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP. В основі REST закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на

попередньому протоколі HTTP 1.0 [7][1)].

Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON).

Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит–відповідь». Стверджується, що такий підхід забезпечує масштабовність системи і дозволяє їй еволюціонувати з новими вимогами.

Існує кілька сервісів, які надають інформацію про погоду, кожен з них має свої плюси і мінуси. Далі буду розглянуті найпопулярніші сервіси і буде зроблений вибір з користь самого оптимального.

Yahoo! Weather – сервіс надає інформацію про погоду по всьому світу, проте інформацію не можна назвати точною, швидше за наближеною.

Weather Underground – сервіс надає дуже багато точної інформації про погоду, наприклад: циклони, фази місяця, магнітні бурі, проте сервіс занадто "комерціалізован" – він надає лише 100 запитів на добу, щоб збільшити їх кількість – потрібно заплатити.

Forecast.Io – хороший сервіс, пропонує досить точний і великий прогноз. Безкоштовних запитів на добу до 1000, чого повинно вистачити. Але в сервісі немає великої кількості середніх і дрібних міст, наприклад – Овідіополя.

OpenWeather – сервіс надає базову інформацію про погоду з хорошою точністю і невеликими обмеженнями (1000 запитів на добу), має гарне покриття по містах.

Порівнюючи всі запропоновані варіантти було вирішено зупинитися на OpenWeather, тому що його точності, покриття і обмежень цілком вистачить для моєї програми і мені не доведеться йти на компроміси.

[1)] [7] Створення Android додатків. URL: http://developer.alexanderklimov.ru. (дата звернення 16.04.2019)

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Моделювання додатку

UML (англ. Unified Modeling Language) – уніфікована мова моделювання, використовується у парадигмі об'єктно–орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, яка називається UML-моделлю[8][1)].

UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

Перша версія (1.0) UML вийшла 13 січня 1997, вона була створена за запитом Object Management Group (OMG) – організації, відповідальної за прийняття стандартів в галузі об'єктних технологій і баз даних. Після обговорення, у вересні 1997 року, версія 1.1 UML була представлена на голосування в OMG. Розробку UML підтримали і вже тоді використовували як стандарт такі гранди ринку інформаційних технологій, як Microsoft, IBM, Hewlett–Packard, Oracle, DEC, Sybase, Logic Works й інші. Поточна версія – 2.0.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес–систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

Діаграми дозволяють уявити систему (як ділову, так і програмну) в такому вигляді, щоб її можна було легко перевести в програмний код.

Основною причиною використання мови UML є спілкування розробників між собою.

---

1) [8] Моделювання на UML. URL: https://book.uml3.ru. (дата звернення 20.04.2019).

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації в кілька разів і значно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі на мові UML у вихідний код об'єктно-орієнтованих мов програмування, ще більш прискорює процес розробки.

Практично всі CASE-засобу (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і спрямувати зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

- керівникам проектів, які керують розподілом завдань і контролем за проектом;
- проектувальникам інформаційних систем які розробляють технічні завдання для програмістів;
- бізнес–аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії;
- програмістам які реалізують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності.

Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

3.1.1 Проектування діаграми прецедентів

Прецеденти – це текстовий документ, що описує послідовність подій, пов'язаних з використанням системи.

Діаграма прецедентів – в UML, діаграма, на якій зображений відношення між акторами і прецедентами в системі.

Актор – це набір ролей, який виконує користувач в ході взаємодії з деякою сутністю (система, підсистема, клас).

На діаграмі прецедентів ілюструється набір прецедентів системи та виконавці, а також взаємозв'язку між ними. Прецеденти визначають, як виконавці взаємодіють з програмною системою. У процесі цієї взаємодії виконавцем генеруються події, які передаються системі, які представляють собою запити на виконання деякої операції. Як правило, окремі кроки або види діяльності у вигляді прецеденту не представляються.

При створенні діаграм використання спочатку визначаються виконавці (ролі, користувачі). Виконавець (actor) є зовнішнім по відношенню до системи поняттям, яке певним чином участь в процесі, описуваному прецедентом. Актор є якийсь ідеалізацією намірів користувача, а не самого користувача. Один реальний користувач може використовуватися декількома акторам, а один актор може представляти один і той же намір відразу декількох користувачів.

Поведінка, розробляється (тобто функціональність, що забезпечується системою) описується за допомогою моделі прецедентів, яка відображає системні прецеденти (use cases), системне оточення (дійових осіб або акторів – actors) і зв'язку між прецедентами і акторами (діаграми прецедентів – use cases diagrams) .

Головною метою даної моделі є реалізація собою єдиного засобу для взаємодії розробника з замовником і кінцевим користувачем для обговорення функціональної складової та поведінки системи.

На основі цього типу діаграм проводиться ітераційний цикл загальної

постановки завдання разом із замовником.

Оскільки замовник ніколи не буде точно знати чого він хоче, то діаграми прецедентів якраз і є основою для досягнення взаєморозуміння між програмістами-професіоналами, які розробляють проект, і "бізнесменами"-замовниками проекту.

Ці діаграми описують функціональність ІС, яка буде видна користувачам системи, основні функції, які повинні бути включені в систему (use case), їх оточення (actors). "Кожна функціональність" зображується у вигляді "прецедентів використання" (use case) або просто прецедентів.

Прецедент – це типова взаємодія користувача з системою, при цьому:

– описує видиму користувачем функцію;

– може представляти різні рівні деталізації;

– забезпечує досягнення конкретної мети, важливої для користувача.

Актори не є частиною системи, вони використовують систему (або використовуються системою) в даному прецеденті.

Актори можуть:

– тільки постачати інформацією систему;

– тільки отримувати інформацію з системи;

– забезпечувати інформацією і отримувати інформацію з системи.

Актор, який представляє людини–користувача, характеризується роллю в даному прецеденті. На діаграмі зображується тільки один актор, проте, реальних користувачів, які виступають в цій ролі по відношенню до ІС, може бути багато.

Актори визначаються на основі складеного завдання або в процесі обговорення з замовником.

Спроектована діаграма прецедентів згідно додатку, що розроблюється, зображена на рис. 7:

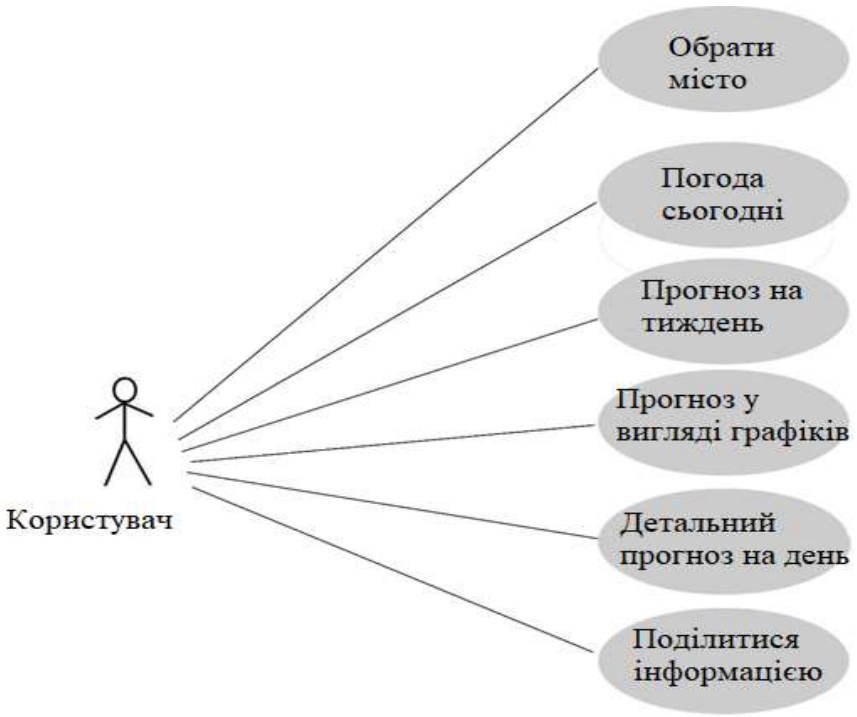

Рисунок 7 – Діаграма прецендентів

У даному випадку у додатку є один рівень доступу для користувача оскільки безглуздно розмежовувати дану інформацію, адже вона знаходиться у відкритому доступі. Кожен користувач може скористуватися усіма можливостями додатку.

3.1.2 Проектування діаграми потоків данних

Діаграма потоків даних (англ. Data Flow Diagram) – модель проектування, графічне представлення «потоків» даних в інформаційній системі. Діаграма потоків даних також може використовуватись для візуалізації процесів обробки даних (структурне проектування).

Для розробника вважається звичним спочатку креслити діаграму потоків даних рівня контексту, завдяки чому буде показано взаємодію системи із

зовнішніми модулями. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати розлого розроблювану систему.

Діаграми потоків даних містять чотири типи графічних елементів:

– сховища даних (репозиторії);
– зовнішні по відношенню до системи сутності;
– потоки даних між елементами трьох попередніх типів.
– процеси – являють собою трансформацію даних в рамках описуваної системи.

Спроектована діаграма потоків даних наведена на рис. 8:

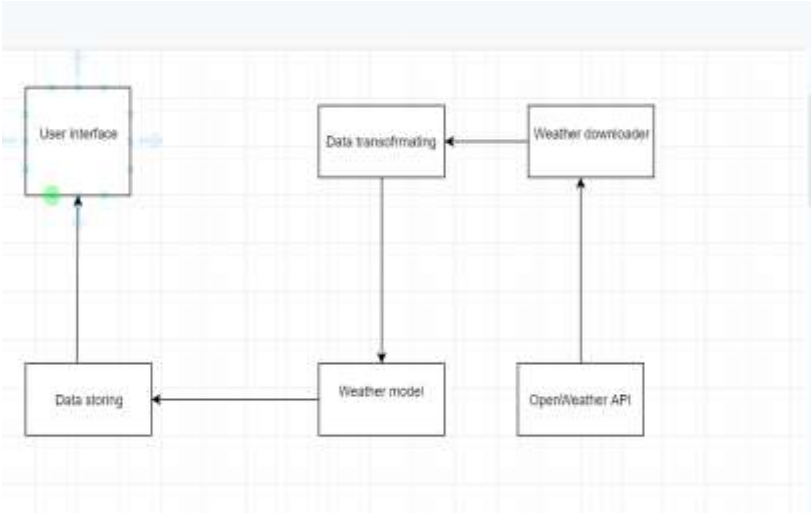

Рисунок 8– Діаграма потоків даних

3.2 Функціональна схема додатку

Компоненти програми можна віднести до одного з чотирьох типів. Компоненти кожного типу призначені для певної мети, вони мають власний життєвий цикл, який визначає спосіб створення і припинення існування компонента.

Існують чотири стандартні блоки додатки Android:

– Activity (активність);

– Intent Receiver;
– Service (служба);
– Content Provider.

Не кожен додатку повинен містити всі чотири блоки, але він буде використовувати деяку їх комбінацію.

Коли розробник вирішує які компоненти необхідно створювати для додатка, їх необхідно записати в файлі на ім'я AndroidManifest.xml. Це – файл XML, де оголошуються елементи додатка, а також їх можливості і вимоги.

Activity (активність) – основний з перерахованих вище блоків Андроїд. Активність являє собою один повноцінний вікні керування, зазвичай – єдиний. Кожна активність здійснена як єдиний клас, який розширює базовий клас Activity. Клас відображає призначений для користувача інтерфейс, складений з Views на її макеті, і відповідає на події ініційовані користувачем. Більшість додатків складається з декількох екранів. Переміщення в інший екран досягається стартом нової активності. Коли новий екран відкривається, попередній екран припиняється і поміщається в стек хронології. Користувач може перейти на попередню активність перемістившись назад через раніше відкриті активності в хронології. Андроїд зберігає стеки хронології для кожної програми.

Незважаючи на те що Activity спільно формують чітку взаємодію користувача з додатком, кожна з них не залежить від інших операцій.

Будь-які з цих операцій можуть бути запущені іншим додатком. Для організації переходу від одного екрана до іншого Андроїд використовує спеціальний клас, називається Intent. Він описує додаток збирається зробити.

Дві найважливіших частині структури Intent – дія і дані до дії. Intent може містити наступні значення для дії MAIN (головний екран додатка), VIEW, PICK, EDIT, і т.д. Дані виражені у вигляді Uniform Resource Indicator (URI) – послідовність символів, що ідентифікує логічний або фізичний ресурс.

Приклади ресурсів включають електронні документи, датчики дверей

ліфта, простір імен XML, веб-сторінки та ідентифікаційні мікрочіпи для домашніх тварин.

Існує пов'язаний клас під назвою IntentFilter. Якщо Intent є запит на якусь дію, то IntentFilter є описом того, що Intent Activity, здатний обробити.

Активність, яка в змозі відобразити інформацію для користувача, видає IntentFilter, який повідомляє, що може зробити VIEW і містить інструкції для обробки. IntentFilters задається у файлі AndroidManifest.xml до відповідної активності.

Як перейти від однієї екранами додатка досягається за допомогою Intent. Щоб переміститися вперед, активність викликає startActivity (myIntent). У такому випадку система звертається до IntentFilter всіх встановлених додатків і вибирає активність, Intent якої відповідає myIntent. Викликаної активності повідомляється про Intent, який змусив її початися. Процес виконання Intent відбувається, в той момент, коли викликається метод startActivity.

IntentReceiver використовується, коли необхідно, щоб код в своєму додатку виповнився у відповідь на зовнішнє подія, наприклад, в разі надходження телефонного виклику, або коли розрядився акумулятор.

Intent Receiver не буде доступний широкому в інтерфейсі, проте вони здатні відобразити повідомлення. Додаток не повинен працювати для його Intent Receiver; система запустить додаток, в разі необхідності, коли Intent Receiver буде викликаний. Додатки можуть також послати свої власні Intent Receiver з іншим Context.broadcastIntent ().

Service (служба) являє собою компонент, який працює у фоновому режимі і виконує тривалі операції, пов'язані з роботою віддалених процесів. Служба не призначеного для користувача інтерфейсу. Треба відзначити, що можна з'єднатися з Service (і запустити його, якщо він вже не працює) з методом Context.bindService (). У разі підключення до Service, з ним можна взаємодіяти через інтерфейс, виставлений Service.

Додатки можуть зберігати свої дані в файлах, базі даних SQLite, персо-

нальних налаштуваннях або будь-якому іншому механізмі, який має сенс. Content Provider, однак, корисний, якщо необхідно, щоб дані додатки були розділені з іншими додатками.

Content Provider – клас, який здійснює стандартний набір методів, щоб дозволити іншим додаткам зберігати і відновлювати тип даних, які оброблені іншим (that) Content Provider.

Призначені для користувача інтерфейси в Андроїд можуть бути створені двома шляхами, через XML–код або в java–коді. Створення структури графічного інтерфейсу користувача в XML переважно, так як за принципом зразкового управління коштами перегляду, призначений для користувача інтерфейс повинен завжди бути відділений від логіки програми[9][1)].

Основний функціональний модуль додатки Android – Activity – об'єкт класу android.app.activity. Activity відповідає за функціональну складову програми, але не має окремої присутності на екрані. Щоб дати Activity присутність на екрані і проектувати його призначений для користувача інтерфейс, необхідно працювати з Views і Viewgroups – основними одиницями графічного представлення візуальних компонентів для користувача інтерфейсу на платформі Андроїд.

View – об'єкт, який розширює базовий клас android.view.view. View є структурою даних і його властивості знаходяться в макеті. View безпосередньо відповідає за інформаційне наповнення для певного контейнера макета.

Об'єкт View обробляє вимір, його схему розміщення, малюнок, зміни центру, прокрутку, і клавіші / знаки для області екрану, яку він представляє. Клас View слугує базовим класом для всіх графічних фрагментів – ряд підкласів, які малюють інтерактивні елементи екрану.

Серед дочірніх графічних об'єктів класу View знаходяться такі як: текстові уявлення у вигляді статичного тексту TextView і рядку для введення тексту EditText, звичайна кнопка Button, кнопка–перемикач, яка дозволяє ко-

---

1) [9] Створення Android додатків. Структура Android додатку. URL: http://4pda.biz/stati/495–sozdanie–android–prilozhenij–struktura–android–prilozheniya. (дата звернення 22.04.2019)

ристувачеві вибрати один з пунктів (опцій) з певного набору RadioButton, кнопка–прапорець Checkbox, ScrollView і т.д.

Viewgroup – об'єкт класу android.view.viewgroup. Viewgroup – спеціальний тип об'єкта View, функція якого – утримувати набір View і Viewgroup і управляти ними. Viewgroups дозволяють додавати структуру до призначеного для користувача інтерфейсу і створювати складні елементи екрану, до яких можна звернутися як до єдиного об'єкту. Клас Viewgroup слугує базовим класом для Layouts – ряду повністю здійснених підкласів, що забезпечує загальні типи Layouts екрану. Layouts дають можливість вбудувати структуру для ряду View.

На платформі Андроїд визначається призначений для користувача інтерфейс Activity використання дерева View і Viewgroup вузлів, як показано на рис. 9:

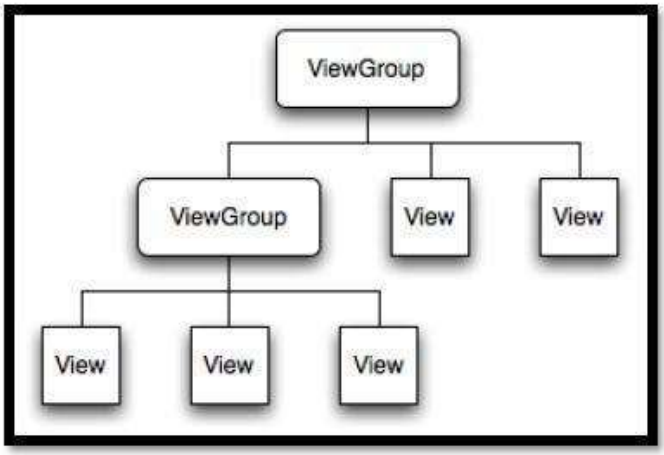

Рисунок 9 – Деревоподібна структура користувальницького інтерфейсу операційної системи «Android»

Дерево може бути настільки ж простим або складним, як зробить розробник. Можна його побудувати, використовуючи набори обумовлених графічних фрагментів і Layouts Андроїда, або замовних типів View .

Щоб прикріпити дерево до екрану і візуалізувати його, Activity викли-

кає свій метод setContentView () і передає інформацію на кореневої об'єкт вузла. Як тільки система Андроїд отримує інформацію на кореневої об'єкт вузла, вона починає працювати безпосередньо з вузлом, щоб виміряти, і визначити дерево.

Коли Activity стає активним і отримує пріоритет, система реєструє його і звертається до кореневого вузла для вимірювання і визначення дерева.

Тоді кореневої вузол надає запит, щоб його дочірні вершини позначили себе – в свою чергу, кожен Viewgroup вузол в дереві відповідальний за визначення його прямих дочірніх вузлів.,

Як згадано раніше, у кожної групи View є відповідальність вимірювання її доступного простору, розташування її дочірніх вузлів, і виклик draw () на кожному дочірньому вузлі, щоб дозволити їм візуалізувати себе. Дочірні вузли можуть запрошувати розмір і місце розташування в батьківської, але у батьківського об'єкта є кінцеве рішення, де і як великий кожен спадкоємець може бути.

Даний додаток використовує активності:

– «MainActivity» – головна активність, на яку потрапляє користувач коли відкриває додаток, на ній відображається погода на поточний момент;
– «SearchActivity» – активність, на якій користувач шукає місто, для якого його цікавить прогноз погоди;
– «WeatherForecastActivity» – активність, на якій користувачеві представлений прогноз погоди на тиждень;
– «GraphsActivity» – історія температури, опадів і вітру, представлена у вигляді граіфков.

До цих активностей додаток містить відповідні макети:

– «activity_main» – макет, що відповідає за формування головної сторінки;
– «activity_search» – макет, що відповідає за формування сторінки «SearchActivity »;

- «activity_weather_forecast» – макет, що відповідає за формування сторінки « WeatherForecastActivity »;
- «activity_graphs» – макет, що відповідає за формування сторінки « GraphsActivity ».

Також додаток містить 1 фрагмент, який поміщуються до активності « WeatherForecastActivity » – «ForecastBottomSheetFragment» – фрагмент, який показує прогноз погоди на обраний день

Додаток налічує два сервіса:

- «LocationService» – сервіс, який визначає геолокацію користувача, щоб користувачеві не доводилося вибирати місто вручну;
- «CurrentWeatherService» – сервіс, який займається отриманням інформації про погоду.

3.3 Алгоритм роботи додатку

При запуску програми відкривається активність, яка показує погоду на поточний момент, а саме: температуру повітря, опади, швидкість вітру, вологість, атмосферний тиск, час світанку і заходу.

З цього екрану користувач може поділитися інформацією про погоду чи обрати інше місто. На цьому екрані можна відкрити бічне меню і перейти на інші екрани: екран графіків і тижневого прогнозу.

Екран графіків – відображає інформацію про температуру, швидкості вітру і опадах протягом тижня у вигляді графіка MPAndroidChart. MPAndroidChart – це бібліотека, яка надає можливість додати в додаток різноманітні діаграми у вигляді звичайних графіків, графіків–пирогів і інших.

Екран з прогнозом погоди на тиждень – на цьому екрані користувачеві відображається прогноз погоди на тиждень у вигляді списку ListVIew. ListVIew – це спосіб реалізації списків при розробці андроїд додатків.

При розробці списку потрібно намалювати один елемент списку як макет, який потім буде наповнюватися даними. Для отримання детальної інфо-

рмації про обраний день користувачеві досить клікнути по ньому і знизу з'явиться картка CardView з детальним прогнозом на обраний день.

CardView – це елемент у вигляді картки, дизайн якого розробник створює сам, як макет для фрагмента або активності.

З огляду на мету зменшення трудовитрат на розробку складного програмного забезпечення, припустимо, що необхідно використовувати готові уніфіковані рішення.

Адже шаблонність дій полегшує комунікацію між розробниками, дозволяє посилатися на відомі конструкції, знижує кількість помилок.

Патерн (англ. Design pattern) – архітектурна конструкція, що представляє собою рішення проблеми проектування в рамках деякого часто виникає контексту.

Почнемо з першого головного – Model-View-Presenter. MVP – це фундаментальний патерн, який знайшов застосування в багатьох технологіях, дав розвиток нових технологій і кожен день полегшує життя розробникам[10][1)].

Вперше патерн MVP з'явився в мові SmallTalk. Розробники повинні були придумати архітектурне рішення, яке дозволяло б відокремити графічний інтерфейс від бізнес логіки, а бізнес логіку від даних.

Під моделлю, зазвичай розуміється частина містить в собі функціональну бізнес–логіку програми. Модель повинна бути повністю незалежна від інших частин продукту. Модельний шар нічого не повинен знати про елементи дизайну, і яким чином він буде відображатися.

Досягається результат, що дозволяє змінювати подання даних, то як вони відображаються, не чіпаючи саму модель.

Таким чином, в класичному варіанті, MVP складається з трьох частин, які і дали йому назву, вони наведені на рис. 10:

---

1) [10] Документація до створення Android додатків. URL: developer.android.com. (дата звернення 24.04.2019).

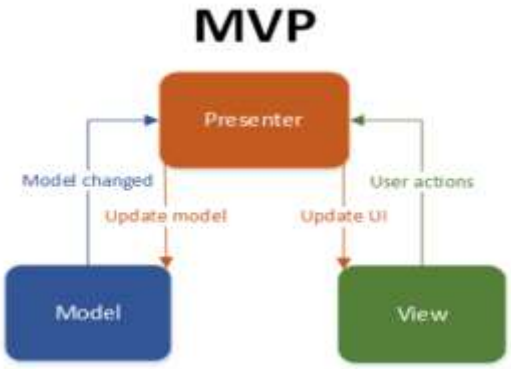

Рисунок 10 – структура шаблону MVP

Модель володіє наступними ознаками:

– модель – це бізнес–логіка програми;
– модель володіє знаннями про себе саму і не знає про контролерах і уявленнях;
– для деяких проектів модель – це просто шар даних (dao, база даних, xml–файл);
– для інших проектів модель – це менеджер бази даних, набір об'єктів або просто логіка додатка.

В обов'язки уявлення входить відображення даних отриманих від моделі. Однак, уявлення не може безпосередньо впливати на модель.

Можна говорити, що уявлення отримує доступ «тільки на читання» до даних.

Подання володіє наступними ознаками:

– у поданні реалізується відображення даних, які виходять від моделі будь–яким способом;
– у деяких випадках, уявлення може мати код, який реалізує деяку бізнес–логіку.

Приклади подання: HTML–сторінка, WPF форма, Windows Form.

Даний підхід дозволяє створювати абстракцію уявлення. Для цього необхідно виділити інтерфейс уявлення з певним набором властивостей і мето-

дів. Презентер, в свою чергу, отримує посилання на реалізацію інтерфейсу, підписується на події вистави і за запитом змінює модель.

Ознаки презентера:

- двостороння комунікація з поданням;
- подання взаємодіє безпосередньо з презентери, шляхом виклику відповідних функцій або подій примірника презентера;
- презентер взаємодіє з view шляхом використання спеціального інтерфейсу, реалізованого поданням;
- один екземпляр презентера пов'язаний з одним відображенням.

Кожна вистава має реалізовувати відповідний інтерфейс. Інтерфейс подання визначає набір функцій і подій, необхідних для взаємодії з користувачем (наприклад, IView.ShowErrorMessage (string msg)).

Презентер повинен мати посилання на реалізацію відповідного інтерфейсу, яку зазвичай передають в конструкторі.

Логіка уявлення повинна мати посилання на екземпляр презентера.

Всі події вистави передаються для обробки в презентер і практично ніколи не обробляються логікою представлення (в т.ч. створення інших уявлень).

3.4 Розробка зовнішнього вигляду додатку

Розробка зовнішнього вигляду додатка відбувалася з використанням сучасних стандартів і принципів побудови призначеного для користувача інтерфейсу. В першу чергу орієнтованість спрямована на Material Design.

Це мова візуальних образів, який створила корпорація Google для уніфікації інтерфейсів її продуктів і сервісів.

Матеріальний дизайн (англ. Material design) – принципи дизайну сайтів, програмного забезпечення і додатків, а також правила дизайну інтерфейсів для операційної системи Android від компанії Google.

Вперше представлений на конференції Google I / O 25 червня 2014 року

Ідея дизайну полягає в інтерфейсі, поведінку і вигляд якого слідують правила поведінки і виду паперових карт в реальному житті.

Дослідження, зроблене дизайнерами Google змінює підхід до інтерфейсу і робить його реальним. Інтерфейс перетворюється в живу приємну людині взаємодія.

Основні компоненти матеріального дизайну наведені на рис. 11:

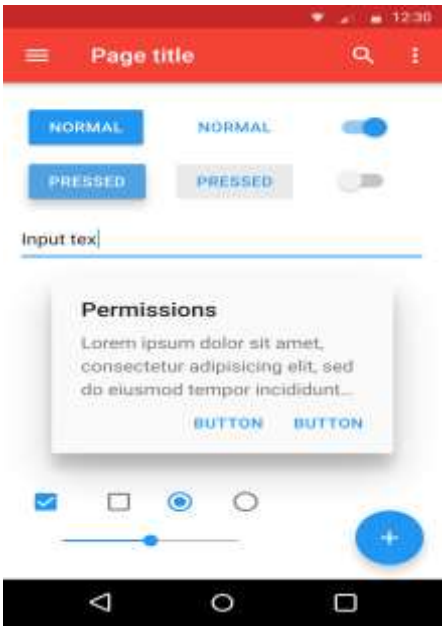

Рисунок 11 – Основні компоненти Material Design

Матеріальний дизайн за допомогою системи управління тінями створює візуальну ілюзію простору між додатком і екраном пристрою.

Основні принципи матеріального дизайну:

– була збільшена реалістичність анімації за допомогою прискорення і пригальмовування рухів, пружного стрибки об'єктів;
– розвинулися інтерактивні іконки;
– з'явилася об'єктна хореографія, при якій елементи здатні впливати на сусідні елементи.

Material design ґрунтується на принципах друкованого дизайну. І не тільки для краси, а й для розстановки акцентів і фокусування уваги користу-

вача на потрібному елементі, для спрощення навігації серед ієрархії конструкцій інтерфейсу, для інтуїтивної передачі їх сенсу. На діях користувача сфокусовано основну увагу. Взаємодією з дизайном керує призначений для користувача досвід, а не навпаки. Всі дії відбуваються в одному оточенні, інтерактивні об'єкти без переривання переходять з одного середовища в іншу.

Крім того додаток розроблявся відповідно до вимог зручності та використовуємих призначеного для користувача інтерфейсу, грунтуючись на UX–дизайні і естетику зовнішнього вигляду.

Нижче будуть наведені скріншоти екранів додатка.

На рис. 12 зображена головна активність, на яку потрапляє користувач, коли запускає додаток. На ньому відображено стан погодних умов на поточний момент.

Рисунок 12 – Головний екран додатка

На рис. 13 показано вікно пошуку міста, який буде обраний для відображення погоди на головному екрані, користувачеві потрібно ввести назву міста в рядок пошуку:

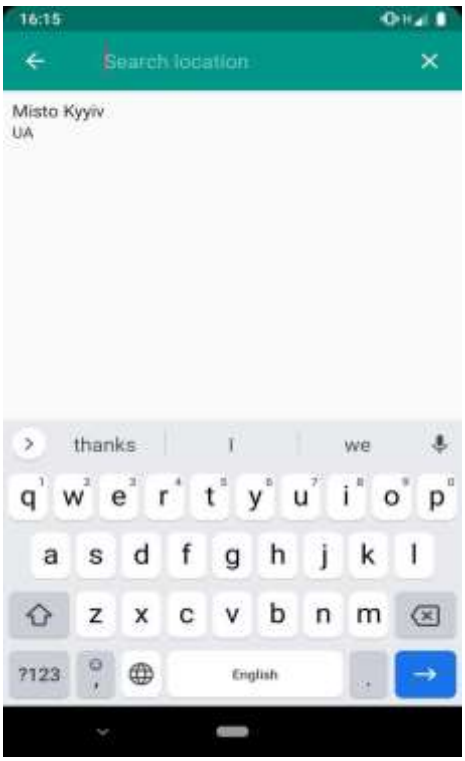

Рисунок 13 – Екран вибору міста

На рис. 14 зображено бічне меню програми, через яке відбувається навігація по екранам додатку:

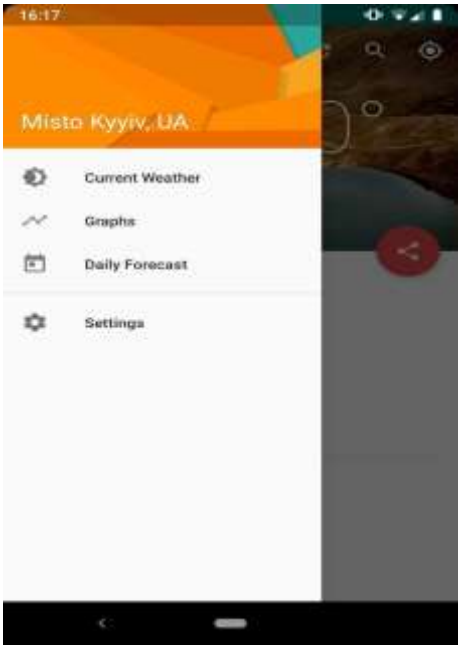

Рисунок 14 – Бокове меню в якому представлена навігація по додатку

На рис. 15 показаний список, на якому представлений прогноз погоди на наступний тиждень:

Рисунок 15 – Прогноз погоди на тиждень

На рис. 16 представлено вікно, через яке користувач може поділитися інформацією про погоду через месенджери або соціальні мережі:

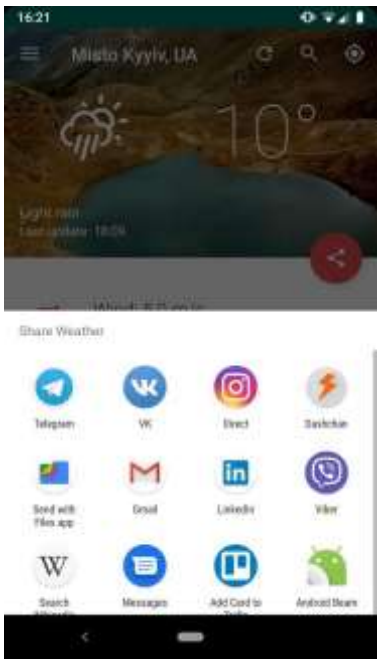

Рисунок 16 – Вікно через яке користувач ділиться інформацією про погоду

3.5 Розробка логіки додатку

В останню чергу потрібно розробити логіку для роботи додатка: для відображення інтерфейсу користувача, для отримання данних с веб–сервісу, для взаємодії з користувачем.

В першу чергу буде розглядатись розробка логіки для отримання інформації з веб–сервису, бо без додаток без з інтерфейсом, але без логіки не має сенсу.

Для роботи додатка потрібно створити 3 модельних класу: для зберігання погоди, прогнозу погоди та міста. Клас WeatherModel зберігає інформацію про стан погоди, його реалізація представлена нижче.

```
public class WeatherModel {

    public CitySearch mlocation;
    public Temperature mtemperature = new Temperature();
    public Wind mwind = new Wind();
    public CurrentWeather mcurrentWeather = new
 CurrentWeather();
    public CurrentCondition mcurrentCondition = new
 CurrentCondition();
    public Cloud mcloud = new Cloud();
    public Sys msys = new Sys();
}
```

Клас SearchModel зберігає інформацію про місто, за яким хоче спостерігати користувач, цей клас містить у собі назву міста, країну а також географічні координати.

Реалізація цього класу представлена нижче.

```
public class SearchModel
{
    private String cityName, country, latitude, longitude,
    countryCode;
```

```
    public CitySearch(String cityName, String countryCode,
    String latitude, String longitude)
    {
        this.cityName = cityName;
        this.countryCode = countryCode;
        this.atitude = latitude;
        this.longitude = longitude;
    }
    public String getCityName()
    public void setCityName(String cityName)
    public String getCountry()
    public void setCountry(String country)
    public String getLatitude()
    public void setLatitude(String latitude)
    public String getLongitude()
    public void setLongitude(String longitude)
    public String getCountryCode()
    public void setCountryCode(String countryCode)
}
```

Далі буде наведена реалізація класу Forecast, цей клас зберігає у собі інформацію про прогнозовану погоду. Він зберігає у собі значення температури, тиску, швидкості вітру та опадів.

```
import java.io.Serializable;

public class Forecast implements Serializable {

    private long mdateTime;

    private float mtemperatureMin;
    private float mtemperatureMax;
    private float mtemperatureMorning;
    private float mtemperatureDay;
    private float mtemperatureEvening;
    private float mtemperatureNight;
```

```
    private String mpressure;
    private String mhumidity;
    private String micon;
    private String mdescription;

    private String mwindSpeed;
    private String mwindDegree;

    private String mcloudiness;
    private String mrain;
    private String msnow;
}
```

Нижче будуть представлені класи, які отримують інформацію про погоду з сервису OpenWeather.

Далі наведена реалізація класу WeatherObtainerService, цей клас займається отправкою http запросів до сервісу OpenWeather, отриманням відповідей та парсінгом відповідей до стану модельних класів.

```
public class WeatherObtainerService(){
public void get() {
        String requestResult = "";
        HttpURLConnection connection = null;
            URL url =
 Utils.getWeatherForecastUrl(Constants.WEATHER_ENDPOINT, latitude,
 longitude, units, locale);
            connection = (HttpURLConnection) url.openConnection();

if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {
ByteArrayOutputStream byteArray = new ByteArrayOutputStream();
InputStream inputStream = connection.getInputStream();

                int bytesRead;
                byte[] buffer = new byte[1024];
                while ((bytesRead = inputStream.read(buffer)) > 0)
 {
                    byteArray.write(buffer, 0, bytesRead);
                }
                byteArray.close();
                requestResult = byteArray.toString();
```

```
                }
            }
            parseWeather(requestResult);
        }
}
```

Далі було б логічно навести реалізацію класів активностей, але логіка інтерфейсу користувача є досить простою, тому буде достатньо навести реалізацію класу MainActivity, а інші класи ітнерфейсу спадкуються від нього і не мають много відмін.

```
Public class MainActivity extends Ativity{
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        weather = new Weather();
        citySearch = new CitySearch();
        weatherConditionsIcons();
        initializeUserInterface();
        initializeWeatherOntainer();

setTitle(Utils.getCityAndCountry(this));
swipeRefresh = (SwipeRefreshLayout)findViewById(R.id.swipe);
swipeRefresh.setProgressViewOffset(false, 0, top_to_padding);
        swipeRefresh.setColorSchemeResources(R.color.swipe_red,
R.color.swipe_green, R.color.swipe_blue);
        swipeRefresh.setOnRefreshListener(swipeRefreshListener);
FloatingActionButton fab = (FloatingActionButton)
findViewById(R.id.fab);
        this.storedContext = this;
        fab.setOnClickListener(fabListener)
```

ВИСНОВКИ

У ході створення додатку були виконані наступні дії:

- проаналізована та описана предметна область;
- розглянуті програмні аналоги, що існують на ринку;
- підтверджена актуальність додатку;
- встановлено мету;
- змодельована робота системи через UML діаграму прецедентів
- розглянуті, проаналізовані та обрані програмні засоби для реалізації додатку;
- розроблен функціональна схема додатку
- розроблен алгоритм роботи додатку
- розроблена візуальна частина додатку;
- цілком розроблена функціональна частина додатку;
- протестовано роботу додатку;
- розроблено керівництво користувача.

Для реалізації програмного забезпечення було проведено об’ємну роботу по взаємодії між елементами середовища Android Studio. Для цього будо використана допоміжна література. За допомогою інструментів, що надаються Android Studio, був разроблен дизайн, та структура додатку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Шилдт Гебрерт. Java 8 полное руководство.:Вильямс, 2016.720 с.
2. Гріффітс Дон, Гріффітс Девід Head First. Програмування для Android. СПб.: Питер, 2016. 704 с.
3. Екель Брюс. Філософія Java. 4–е полное изд.: СПб.: Питер, 2015. 1168 с.
4. Документація до створення Android додатків.URL: developer.android.com(дата звернення 16.04.2019)
5. Гріффітс Дон, Гріффітс Девід Head First. Програмування для Android.:СПб.: Питер, 2016.704 с.: ил.
6. Створення Android додатків. URL: http://developer.alexanderklimov.ru/ (дата звернення 20.04.2019).
7. Створення Android додатків. Структура Android додатку.URL: http://4pda.biz/stati/495–sozdanie–android–prilozhenij–struktura–android–prilozheniya (дата звернення 22.04.2019)
8. Моделювання на UML. URL: https://book.uml3.ru (дата звернення 24.04.2019)
9. Метеорологія і кліматологія як науки. URL: http://www.geograf.com.ua (дата звернення 13.04.2019)
10. Сторінка додатка AccuWeather в Google Playstore. URL: https://play.google.com/store/apps/details?id=com.accuweather (дата звернення 15.04.2019).