Міністерство освіти і науки України
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Каштелян Іван Володимирович

«Інформаційна модель планування задач на основі алгоритмів машинного навчання / The information model of task scheduling based on machine learning algorithms»

Студент групи КІм – 21
Каштелян Іван Володимирович
_____

Науковий керівник
к.т.н., викладач, О.Й. Піцун
_____

Тернопіль – 2020

РЕЗЮМЕ

Кваліфікаціна робота на тему «Інформаційна модель планування задач на основі алгоритмів машинного навчання» зі спеціальності 123 «Комп'ютерна інженерія» освітнього ступеня магістр має обсяг 91 сторінки, та містить 41 рисунків, 2 таблиці, 2 додатки, та 50 джерел.

Метою випускної кваліфікаційної роботи є створення інформаційної моделі планування задач на основі алгоритмів машинного навчання.

Методи дослідження: методи та алгоритми машинного навчання.

Було досліджено важливість плавильного планування задач задля збільшення ефективності та уникнення вигорань.

Було проаналізовано найпопулярніші алгоритми машинного навчання, та обраний найбільш оптимальний для створення на його основі інформаційної моделі.

Було проведено проектування бази даних, та розроблено інформаційну модель планування задач. В ході розробки та проектування було проаналізовано аналоги, з метою знаходження оптимального способу реалізації того чи іншого функціоналу інформаційної планування задач.

Можливими напрямками подальших досліджень є розробка ефективних підходів, з використанням машинного навчання, для покращення планування задач.

КЛЮЧОВІ СЛОВА: ЕФЕКТИВНІСТЬ ПЛАНУВАННЯ, МАШИННЕ НАВЧАННЯ, АЛГОРИТМИ ШТУЧНОГО ІНТЕЛЕКТУ, ІНФОРМАЦІЙНА МОДЕЛЬ.

RESUME

Qualification work on "The information model of task scheduling based on machine learning algorithms" in the specialty 123 "Computer Engineering" which has a volume of 89 pages and contains 41 figures, 2 tables, 2 appendices and 50 sources.

The purpose of the final qualification work is to create an information model of task planning based on machine learning algorithms.

Research methods: methods and algorithms of machine learning.

The importance of planned task planning to increase efficiency and burnout was investigated. The most popular algorithms of machine learning were analyzed, and the best optimal information model for its creation was chosen. The database was designed and an information model of task planning was developed. In the course of development and design, analogues were analyzed in order to find the optimal way to implement a particular functional information planning.

Possible areas for further research are the development of effective approaches, using machine learning, to improve task planning.

KEYWORDS: EFFICIENCY OF PLANNING, MACHINE LEARNING, ARTIFICIAL INTELLIGENCE ALGORITHMS, INFORMATION MODEL.

ЗМІСТ

ВСТУП

Актуальність роботи. Завжди була актуальною проблема нехватки часу, проте зважаючи на постійно зростаючий рівень навантаження людини професійними, соціальними задачами доцільним є розроблення інформаційної інтелектуальної системи для формування оптимального з точки зору активності людини графіку задач. Використання сучасних технологій підвищує комфорт планування задач, так як смартфон завжди поруч та має багато корисних функцій які збільшують ефективність планування. Використовуючи засоби нейронних мереж які аналізують дані використання смартфона можна визначити періоди активності користувача на основі чого надавати рекомендації по встановленні часу виконання поставлених задач планувальника[1].

Аналіз предметної області показав, що для розробки алгоритмів машинного навчання існує багато підходів. Основою інформаційної моделі було вирішено обрати метод k найближчих сусідів як найбільш оптимальний для вирішення поставленої задачі.

Аналіз обраного методу машинного навчання «k найближчих сусідів» показав, що його недоліком є зберігання всієї навчальної вибірки.

Для покращення точності роботи алгоритму перспективним є використання декількох моделей результати яких будуть поєднуватись в спільний з найбільшою точністю, та з меншою кількістю помилок[2].

Мета і завдання дослідження. Метою кваліфікаційної роботи є створення інформаційної моделі планування задач на основі алгоритмів штучного навчання.

Об’єкт дослідження. Процес планування часу виконання специфічного завдання.

Предметом дослідження є алгоритми та методи машинного навчання, котрі використовують вхідні дані застосовуючи статистичний аналіз для прогнозування результатів.

Для виконання поставленої мети необхідно розв’язати наступні задачі:

− відповідно до завдання, необхідно визначити оптимальний алгоритм машинного навчання;

− спроектувати архітектуру інформаційної системи;

− спроектувати архітектуру бази даних;

− порівняти розроблену систему з наявними аналогами;

− опублікувати систему в публічний доступ для проведення тестування та збору зворотнього зв’язку користувачів;

Методи дослідження. Методи та алгоритми машинного навчання.

Наукова новизна одержаних результатів полягає у доповненні та покращенні існуючої інформаційної технології планування часу людини з врахуванням специфіки діяльності людини в поєднанні з даними використання смартфона.

Практичне значення отриманих результатів. Спроектована та розроблена інформаційна система планування завдань відповідно до шаблону проектування MVVM, з використанням алгоритму машинного навчання «k – найближчих сусідів».

Публікації та апробації кваліфікаційної роботи. Отримані результати апробовані в межах III науково-практичної конференції «Інтелектуальні комп’ютерні системи та мережі» Західноукраїнського національного університету, та опубліковано дві тези доповіді по темі роботи [1,2].

Кваліфікаційна робота складається із трьох розділів, висновків, списку використаної літератури та додатків. У першому розділі було проаналізовано особливості та важливість планування часу, а також проаналізовано доступні аналоги на ринку та програмні засоби для реалізації програмного продукту.

У другому розділі було проаналізовано доступні алгоритми машинного навчання, обрано найбільш оптимальний, спроектовано архітектуру програми та бази даних.

У третьому розділі було реалізовано спроектовану архітектуру програми та бази даних. Описано роботу розробленого продукту, та покроково описано реєстрацію та налаштування аккаунта розробника в Google Play після чого є можливість опублікувати додаток для широкого кола користувачів.

1 АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ ПЛАНУВАННЯ ЗАДАЧ ТА МОЖЛИВОСТІ ВИКОРИСТАННЯ В НИХ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ

1.1 Програмні засоби планування задач

Управління часом - це громіздкий розділ науки, у якому вивчаються проблеми і методи оптимізації витрат часу в різних сферах людської діяльності. Тайм-менеджмент має широкі зв'язки з безліччю інших наук: філософією, психологією, соціологією, біологією, фізикою – у всіх перечислених науках, а навіть багатьох інших міститься безліч знань та порад про час.

Керування часом, тайм-менеджмент (від англ. time management) – сукупність методик оптимальної організації часу для виконання поточних задач, проектів та календарних подій. Типовими підходами в керуванні часом є постановка пріоритетів, розбиття великих завдань та проектів на окремі дії та делегування іншим людям. До керування часом належать також методи впливу на мотивацію та контролю результатів. По темі менеджменту часу часто проводяться психологічні тренінги. Головними допоміжними інструментами для керування часом є особистий календар, список поточних завдань та список проектів. Механізми для керування часом (календар та список задач з можливістю їх пріоритизації та категоризації) реалізовані в комп'ютерних програмах таких як Microsoft Outlook, iCal а також у сучасних мобільних додатках[3]. Планування надає такі переваги :

− планування дисциплінує;

− планування спрощує робочий процес;

− планування робить людину більш ефективною;

− планування знижує рівень стресу;

− планування розвиває пунктуальність;

− планування вивільняє вільний час;

− пданування впорядковує справи;

− планування звільняє мозок;

− планування сприяє натхненню;

− планування сприяє досягненню цілей [4].

Процес планування часу слід починати з постановки задач, для яких на майбутній період складається перелік справ і можливих перешкод, на подолання яких піде визначений час. Згодом цей перелік регулярно доповнюється, обновляється, коректується шляхом виключення з нього того, що на справді є несуттєвим. [5]

Зазвичай люди використовують для планування свого часу такі додатки як Google Keeps, Google Calendar, MS Outlook. Дані додатки виступають як місце для зберігання задач та відповідно часу їх проведення. Людина самостійно обирає час або місце виконання задач та контролює їх кількість.

Доповнюючи традиційні інструменти планування задач створюються додатки, що на основі методів штучного інтелекту розширюючи основний функціонал попередніх. Сучасні системи здатні самостійно формувати розклад людини, запам’ятовувати звички, аналізувати попередні задачі, що дає змогу назвати тип даних додатків особистим асистентом. Суміжним типом додатків є системи тайм менеджменту для команд, що додатково надають змогу контролювати співробітників та команду в цілому.

Далі представлений опис додатків, пов’язаних за тематикою з керуванням часом та задачами, що використовують у своєму функціоналі методи штучного інтелекту.

Timeful – Додаток, розроблений під керівництвом вчених в області штучного інтелекту Йоава Шохама та Якоба Бенка та відомого вченого в області психології та поведінкової економіки Дена Аріелі, що функціонує як традиційний календар, проте також рекомендує оптимальний час для конкретних подій.

Якщо перед людиною стоїть задача піти в продуктовий магазин, та вона має вільний час між 18:00. і о 20:00 у понеділок, додаток може запропонувати їй саме цей вільний час. Запропонований час може бути прийнятий, у такому

випадку він відображатиметься у календарі, змінено або відхилено, у такому випадку з'явиться альтернативний запропонований часовий інтервал.

Коли ви вперше починаєте роботу над додатком, Timeful робить пропозиції, що базуються на середніх показниках, наприклад, що статистично, люди найбільш продуктивні вранці і відповідно вирішують справи вранці. У той час як користувач вводить більше даних у систему, особисті моделі домінують у рекомендаціях та система дізнається, що людина прийняла і відкинула, і з часом зробить кращі пропозиції[4].

Додаток має зручний інтерфейс для перегляду добового списку задач (рис.1.1) та контролю діяльності впродовж місяця(1.2).

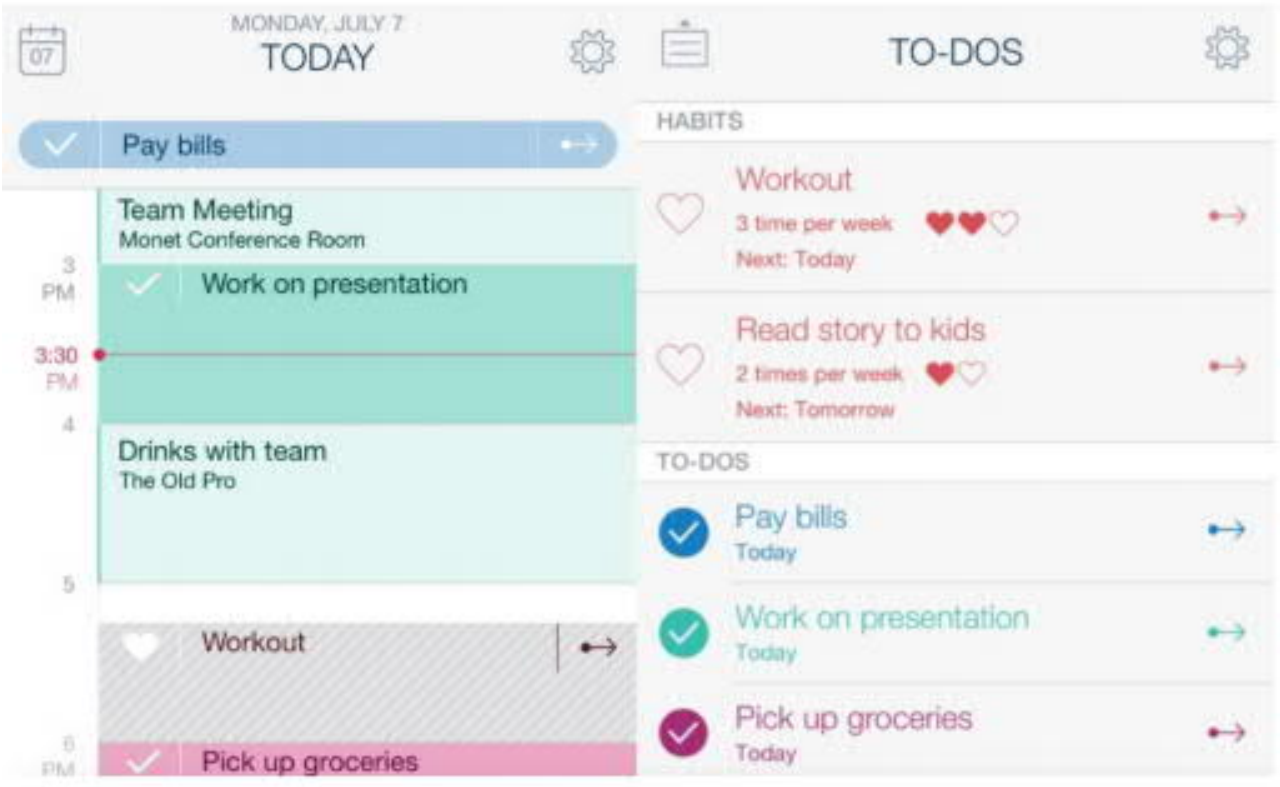


Рисунок 1.1 – Вікно додатку Timeful (Перегляд добового списку справ)

Основною функцією Timeful є «розумні» пропозиції планування: вказавши тимчасові рамки для виконання завдання, використовуючи алгоритми оптимізації та поведінкової науки, Timeful пропонує інтервали для завдань, щоб забезпечити його виконання[5].

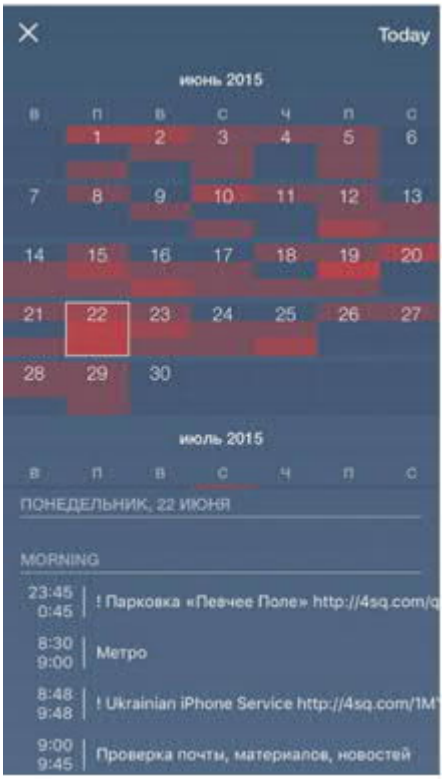


Рисунок 1.2 – Відображення місячного навантаження в додатку Timeful

SELFPLANNER – інтелектуальний додаток-календар, який допомагає користувачу планувати в часі та просторі свої індивідуальні завдання. На відміну від інших асистентів, які концентруються на автоматизації планування зустрічей, SELFPLANNER підкреслює планування індивідуальних завдань і подій.

SELFPLANNER підтримує прості, переривчасті та гнучкі періодичні завдання, часові проміжки, обмеження над частинами перериваного завдання, двійкові обмеження між завданнями, перевагами часових проміжків над іншими, посилання на місцезнаходження, класи розташувань, часові пояси тощо.

SELFPLANNER інтегрується з Календарем Google і додатком на основі Карт Google (рис.1.3). Він запроваджує інноваційний спосіб визначення часових проміжків на основі визначених користувачем правил. Основою системи є Squeaky Wheel Optimization алгоритм поєднаний з ефективною евристикою. SELFPLANNER є кроком на шляху до наступного покоління інтелектуальних додатків календаря[6].

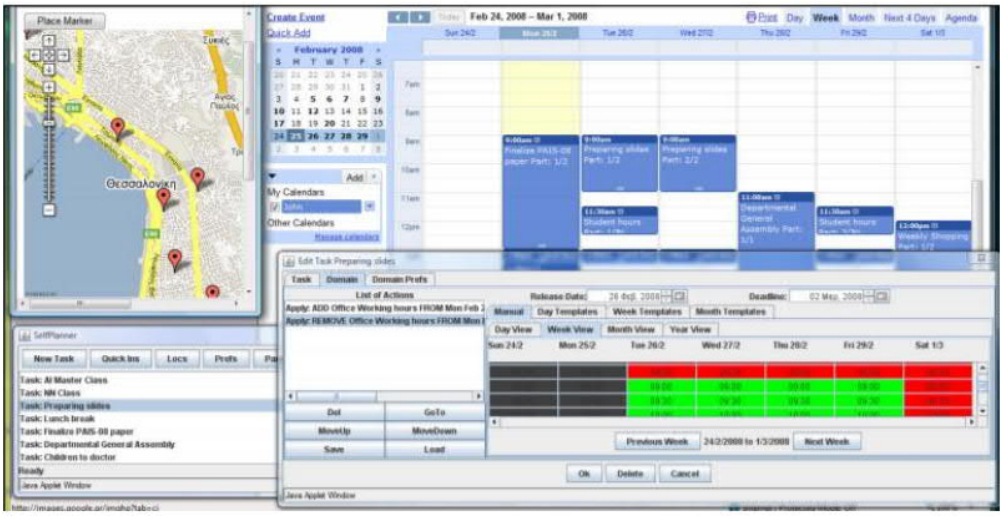


Рисунок 1.3– Вікно додатку SELFPLANNER

Time – додаток за допомогою якого користувач може нав'язливо контролювати завдання під рукою з кольоровим таймером, який змінюється від зеленого до жовтого та до червоного, коли виділений час для даного завдання закінчується.

На перший погляд програма не виглядає дуже складною, хоча приховує технології штучного інтелекту. Чим більше користувач використовує Time, тим розумнішим він стає. Підсумковий перегляд покаже всі завдання, що були зроблені, а технології запропонують пропозиції щодо продуктивності на основі вашого попереднього використання.

Аналіз діяльності користувача починається після того, як виконується певне завдання кілька разів. Алгоритми використовують лінгвістичне розпізнавання, щоб зв'язати назви завдань разом, навіть якщо вони не написані однаково. Наприклад, «code app» and «work on app» можна вважати тим самим елементом[7].

AI програми починає працювати з часом, дізнаючись, як ви працюєте, і пропонуєте індивідуальні пропозиції щодо продуктивності. Користувач може отримати візуалізацію того, скільки часу він працював та додавав кожному завданню. Це допоможе краще зрозуміти завдання, які були виконані швидко, і ті, які потребували додаткового часу[8]. Якщо користувач завжди відставав

від своїх цілей у часі, програма може запропонувати додати певну кількість додаткових хвилин до завдання наступного разу (рис. 1.4).

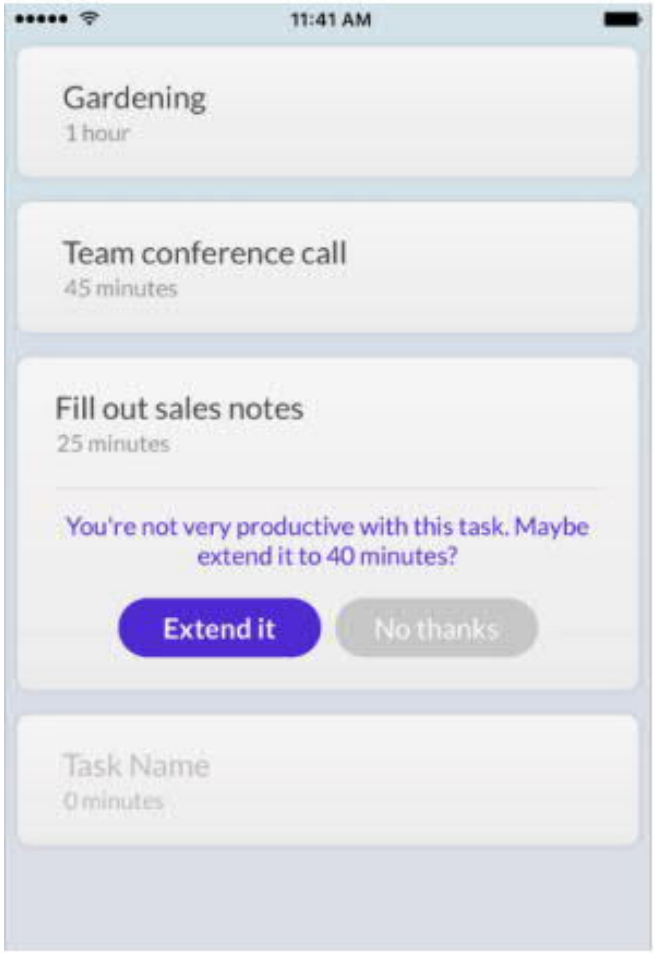


Рисунок 1.4 – Додаток Time, пропозиція збільшити час

Todoist – планувальник завдань з функцією штучного інтелекту, що називається Smart Schedule.

Smart Schedule використовує інтелектуальне моделювання, щоб допомогти користувачу легко спланувати свої завдання на день і тиждень. Вона вивчає особисті звички та враховує моделі всіх користувачів Todoist, щоб передбачити найкращі терміни виконання завдань.

Це означає, що завдання, які планує користувач, можуть бути швидко перепланові в масовому порядку, тоді як нові та позапланові завдання можуть бути легко призначені для кращих термінів.

При пошуку ідеальних термінів, Smart Schedule враховує:

– Звички – Smart Schedule ознайомиться з звичками та відповідно запропонує дати. Усі особисті дані обробляються автоматично алгоритмом інтелектуального розкладу.

– Терміновість завдання – користувач при створенні завдання може вказати його пріорітет

– Робочі дні в порівнянні з вихідними - Smart Schedule дізнається, які типи завдань можна виконувати у вихідні дні, і які завжди повинні бути заплановані протягом тижня

– Баланс – Smart Schedule спробує збалансувати навантаження на завдання протягом наступних 7 днів відповідно, та не перевантажувати одни день в порівнянні з іншим (рисунок 1.5).

– Щоденні та щотижневі цілі - Todoist дозволяє встановлювати та відстежувати цілі за кількістю завдань, які потрібно виконувати кожен день і тиждень. Розумний графік рекомендуватиме терміни, які допоможуть вам досягти конкретних цілей.

Користувач завжди матимете змогу приймати, редагувати або відхиляти пропозиції Smart Schedule. Як і для всіх функцій, що працюють на основі AI, які з часом «навчаються», прогнозування дат смарт-розкладу ставатиме більш точними, чим більше користувач використовує Todoist [9].

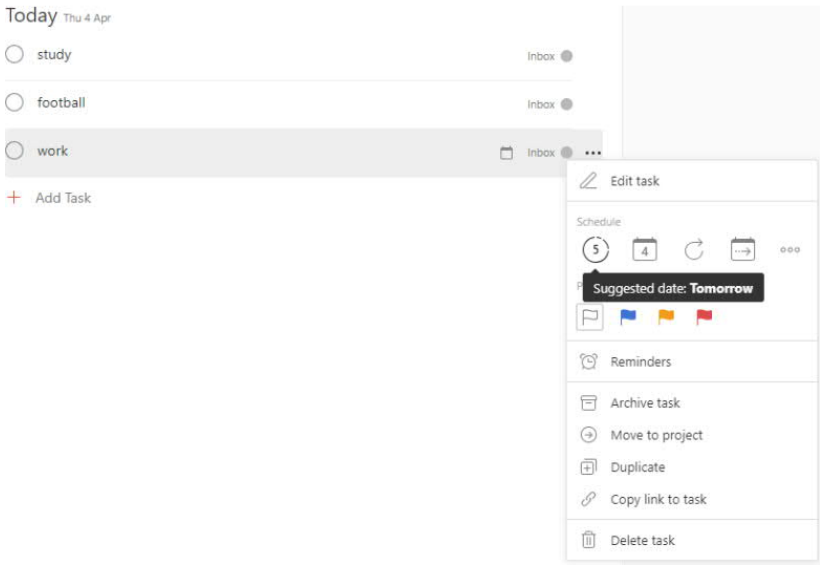


Рисунок 1.5 – Додаток todoist, пропозиція змінити день виконання

На ринку існує багато рішень з використанням штучного інтелекту, слід звернути увагу, що кожен з них по своєму унікальний та дає великий набір інструментів користувачу для управління часу, та для того щоб робити це ефективніше. При розробці продукту слід брати найкраще від конкурентів та покращувати це. Також не потрібно забувати про унікальність власного продукту, а саме дизайн, та різноманітний функціонал який триматиме користувача.

1.2 Існуючі методи машинного навчання

Машинне навчання – великий підрозділ штучного інтелекту, математична дисципліна, яка використовує розділи математичної статистики, чисельних методів оптимізації, теорії ймовірностей, дискретного аналізу, і витягати знання з даних [7]. Основною метою машинного навчання є повна або часткова автоматизація вирішення різноманітних нелінійних професійних завдань в різних областях людської діяльності.

Штучні нейронні мережі набули значної популярності, як інструмент прогнозування, за останні роки. Нейронна мережа - це система складається з безлічі простих процесних елементів діючих паралельно, чия функція визначається структурою мережі, силою зв'язків і процесом, що відбувається при комп'ютерних обчисленнях цих елементів. Нейронна мережа накопичує знання завдяки процесу навчання. Зв'язки між нейронами, які ще називають санаптичні ваги, зберігають накопичені знання. Завдяки такій характеристиці, нейронні мережі знайшли свою популярність в області прогнозування на фондовому ринку. Вони залишаються актуальними донині і застосовуються в безлічі програмних пакетів. Альтернативні підходи, такі як лінійні регресії і кластеризація, вважаються менш ефективними, таким чином, машинне навчання переважає по ефективності в аналізі над альтернативними підходами.

Наприклад компанія OpenAI Ілона Маска, яка розробляє штучний інтелект та тестує його на професійних гравцях комп'ютерної гри Dota 2, за декілька років штучний інтелект навчився не тільки грати в цю гру, а й зміг перемогти професійних гравців та команди[36].

Нейронна мережа складається з нейронів різних типів які обробляють інформацію, що надходить до них за допомогою каналів зв'язку, що визначаються при створенні проектної моделі системи. Видів нейронних мереж багато, принципово вони відрізняються:

− наявністю прихованих шарів нейронів, тобто тих нейронів, результат обробки інформації якими не видно;

− кількістю прихованих шарів, що впливає на можливість розшифрувати шлях, яким дана нейронна мережа дала саме такий результат;

− видом нейронів, що впливає на спосіб обробки (або зберігання) інформації;

− типами зв'яку між нейронами;

− наявністю чи відсутністю зворотнього зв'язку у нейронній мережі;

− кількістю нейронів у шарі.

Генетичний алгоритм – алгоритм поступового підбору, рекомбінування і ітерації деяких змінних.

Генетичний алгоритм являє собою евристичний метод випадкового пошуку, заснований на принципі імітації еволюції біологічної популяції. Спочатку вибирається сімейство покриттів, заданого обсягу S. Масив P називають населенням, а його елементи - особинами. З популяції P обираються дві особини, наприклад I, звані батьками I, з покриття, виділяється покриття J, зване їх нащадком. Нащадок піддається мутації (випадковому зміни), після чого він заміщає в P найгіршу особину. Цей цикл - вибір батьків, створення нащадка, його мутація і оновлення популяції - повторюється заданий число раз. Результатом роботи алгоритму є найкраще з покриттів, що виникли в ході розвитку популяції.

Так як у генетичних алгоритмів тобто не оцінюється чисельно характеристика, що описує їх пошукові здібності, вони залежать від всього, але в більшій мірі від стратегій селекції та генетичних операторів, отже можна зробити наступні висновки:

− використання більш агресивних варіантів відбору укупі з досить великою ймовірністю мутації в багатьох випадках дозволяє домогтися більш хороших результатів, в порівнянні з канонічним генетичним алгоритмом. Агресивними стратегіями відбору можна вважати відбір урізанням з досить великим порогом (тобто коли до відтворення допускається менша кількість особин), а також турнірний відбір з розміром турніру 4 і більше;

− двоточковий і однорідний оператори кросинговеру, як правило, працюють краще, ніж одноточковий;

− популяція більшого розміру працює стабільніше і часто краще. Якщо ж необхідно вкластися в кілька обчислень цільової функції, то краще пошукати оптимальний розмір, при якому і рішення може бути знайдено, і обчислювальні витрати цілком прийнятні;

− застосування стратегії елітарності - дозволяє гарантовано залишити в популяції найкращих особин;

− планомірне вистежування і ліквідація диверсійних елементів в особі дублікатів в популяції підвищують якість результатів і корисно проти передчасної збіжності;

− велика ймовірність мутації в деяких випадках здатна поліпшити роботу алгоритму (особливо для малих популяцій), але небажана, в силу внесення великий хаотичності в еволюційний процес, що може негативно позначитися на стабільності роботи алгоритму. [3]

Байєсова мережа – це модель яка описує стан певної састини даних та описує їх взаємозв’язок з ймовірністю. Модель може бути будинком, автомобілем, тілом, громадою, екосистемою, фондовою біржею тощо. Всі можливі стани мережі відображають всі можливі існуючі світи, тобто обробляться можливі шляхи, якими можна налаштовувати частини або стани.

Байєсовські мережі входять в категорію імовірнісних графічних моделей (ВГМ). ВГМ використовуються для обчислення мінливості для застосування в концепціях ймовірності.

Загальноприйнята назва Байєсова мереж - Глибокі мережі. З їх допомогою моделюються спрямовані ациклічні графи.

Байесові мережі використовуються в таких напрямах:

− менеджмент фінансових ризиків;

− моделювання екосистем;

− передбачення;

− діагностика.

Обмеження байесової мережі:

− складність обчислення;

− у випадку наявності великої кількості випадкових параметрів мережа намагається поставити їх у причинно-наслідкові зв’язки. Внаслідок чого справжні зв’язки втрачаються або втрачають значущість. Байєсова мережа кодує лише спрямовані зв'язки, а не двонаправлені. Байесова мережа не дає жодних гарантій щодо зображення причинно-наслідкових зв'язків[10];

− виходячи з попереднього пункту, якщо дані були згенеровані з моделі, в якій щонайменше три змінні співвідносяться між собою, байесова мережа не зможе змоделювати ці зв’язки;

− деякі зі складних функцій забивання потребують надійних пріорітетів, щоб знайти структуру, яка ближче до оригінальної моделі.

Зважаючи на наявні кращі альтернативи та серйозні обмеження, а також складність розробки системи з використанням байесової мережі, було прийнято рішення відмовитися від розробки інтелектуальної системи з використанням байесової мережі.

Навчання з підкріпленням – область машинного навчання, пов'язана з тим, як програмні агенти повинні робити дії в середовищі, щоб досягти деякого результату, який позначено як найкращий. Ця проблема, в силу своєї спільності, вивчається у багатьох інших дисциплінах, таких як теорія ігор,

теорія управління, дослідження операцій, теорія інформації, оптимізація на основі моделювання, статистика і генетичні алгоритми.

У навчання з підкріпленням є один великий плюс. У симуляторі можна створити спрощену модель світу. Так, для фігурки людини достатньо всього 17 ступенів свободи, замість 700 в живій людині (приблизну кількість м'язів). Тому в симуляторі можна вирішувати завдання в дуже маленькій розмірності.

Фізичний пристрій мозку і нервової системи налаштоване еволюцією під конкретний вид тваринного і його умови проживання. Так, у мухи в процесі еволюції розвинулася така нервова система і така робота нейромедіаторів в гангліях (аналог мозку), щоб швидко ухилятися від мухобойки. Ну добре, що не від мухобойки, а від птахів, які їх ловили 400 мільйонів років (жарт, птиці самі з'явилися 150 млн років тому, швидше за від жаб 360 млн років). А носорога досить такий нервової системи і мозку, щоб повільно повернутися в бік цілі і почати бігти. А там, як то кажуть, у носорога поганий зір, але це вже не його проблеми.

Але крім еволюції, у кожної конкретної особи, починаючи з народження і протягом усього життя, працює саме звичайний механізм навчання з підкріпленням. У разі ссавців, та й комах теж, цю роботу виконує дофаминовая система. Її робота сповнена таємниць і нюансів, але все зводиться до того, що в разі отримання нагороди, дофаминовая система, через механізми пам'яті, як-то закріплює зв'язку між нейронами, які були активні безпосередньо до цього. Так формується асоціативна пам'ять. Яка, в силу своєї асоціативності, потім використовується при прийнятті рішень. Простіше кажучи, якщо поточна ситуація (поточні активні нейрони в цій ситуації) по асоціативної пам'яті активують нейрони пам'яті про задоволення, то особина вибирає дії, які вона робила в схожій ситуації і які запам'ятала. "Вибирає дії" - це погане визначення. Вибору немає. Просто активовані нейрони пам'яті про задоволення, закріплені дофаминовой системою для даної ситуації, автоматично активують моторні нейрони, що призводять до скорочення м'язів. Це якщо необхідно негайне дію.

Навчання без вчителя є одним із методів машинного навчання, яке реалізує можливість навчання без зовнішнього втручання. Сутністю даної методики є припущення про те, що існують деякі приховані закономірності, які неможливо або дуже важко знайти і використати в навчанні з вчителем чи підкріпленням, тобто будь-яке зовнішнє втручання погіршить результат. Такий спосіб навчання можливий у випадках, коли правила існування системи, в якій необхідно знайти рішення деякої задачі, відомі, про неможливо знайти хоча б якість логічні зв’язки між окремими компонентами системи, тобто правила гри відомі, а стратегії ще ні.

Кластерний аналіз – задача розбиття заданої множини деяких об'єктів на підмножини так, щоб кожна підмножина складалася з найбільш схожими групами параметрів і відрізнялася від інших підмножин. Кластерний аналіз це один із прикладів застосування методу машинного навчання без вчителя.

Кластерний аналіз – задача, для розв'язання якої використовуються різні підходи. Відносно того, що можна вважати кластером, а зо ні, алгоритм кластеризації може бути побудованим різними способами. Для того, щоб пояснити логіку реалізація кластерного аналізу, візьмемо приклад з задачею розділення країн на групи по характеристикам, наприклад, кількістю людей з вищим рівнем освіти, рівнем праці, етапом індустріального розвитку. Виконавши кластерний аналіз, отримаємо декілька підмножин загальної множини країн, і в одній із цих підмножин будуть Японія, Німеччина, Франція, Англія. В той час Уганда, Киргистан та Сомалі будуть в іншій підмножині тому, що вони поширюють між собою інший набір характеристик включаючи низький рівень життя, нестабільні та недемократичні інститути правління та низький рівень технологічного розвитку[12].

Глибоке навчання – це техніка машинного навчання, а точніше сукупність технік машинного навчання, яка навчає комп'ютер робити те, що природно приходить людям: навчитися на прикладі.

При глибокому вивченні комп'ютерна модель навчається виконувати завдання класифікації безпосередньо з зображення, тексту або звуку. Моделі

глибокого навчання можуть досягти високої точності роботи, іноді перевищуючи продуктивність людини. Моделі навчаються за допомогою великого набору помічених даних та архітектур нейронних мереж, що містять багато шарів.

Декілька прикладів застосування машинного навчання:

− віртуальні ассистенти. Сірі, Кортана та Алекса використовують глибоке навчання для того, щоб навчитися розуміти мову людей;

− переклад. У схожий спосіб, алгоритми глибоко навчання можуть автоматично перекладати декілька мов. Це використовується та буде мати попит по всьому світі тому, що існують безліч людей, які не розуміють інших мов;

− чатботи та автовідповідачі. Вони використовуються для комунікації з клієнтами та не потребують людських ресурсів, тобто можуть майже безкоштовно замінити людей;

− додання кольору на фотографії. На даний момент існують безліч фотографій часів чорно-білої фотографії, які можливо перевести в колів за допомогою глибокого навчання;

− розпізнавання облич. Має попит у сфері безпеки та обслуговування, дозволяючи зменшити витрати на касирів (покупець буде здійснювати покупки за допомогою розпізнавання свого обличча);

− медицина та фармацевтика. Глибоке навчання дозволяє розпізнавати хвороби (наприклад, розпізнавання віку та рак за фотографію очей)

− персоналізація процесу покупок та дозвілля. Алгоритми глибокого навчання дозволяють персоналізувати асортимент товарів, які надаються клієнтові, що збільшує імовірність покупки або придбання послуги. Наприклад, Netflix використовує глибоке навчання для впровадження системи, яка надає глядачам фільми на спробу та має декілька сотень тисяч категорій глядачів, що дозволяє дуже ретельно підбирати товар для кожного клієнта.

Структура моделей глибокого навчання складається з нейронної мережі з декількох шарів, з яких деякі – приховані, тобто інформація обробляється

декілька разів всередині них. Одним із популярних видів нейронних мереж в такому випадку стають конволюційні мережі, приклад такої наведено на рисунку 1.6.

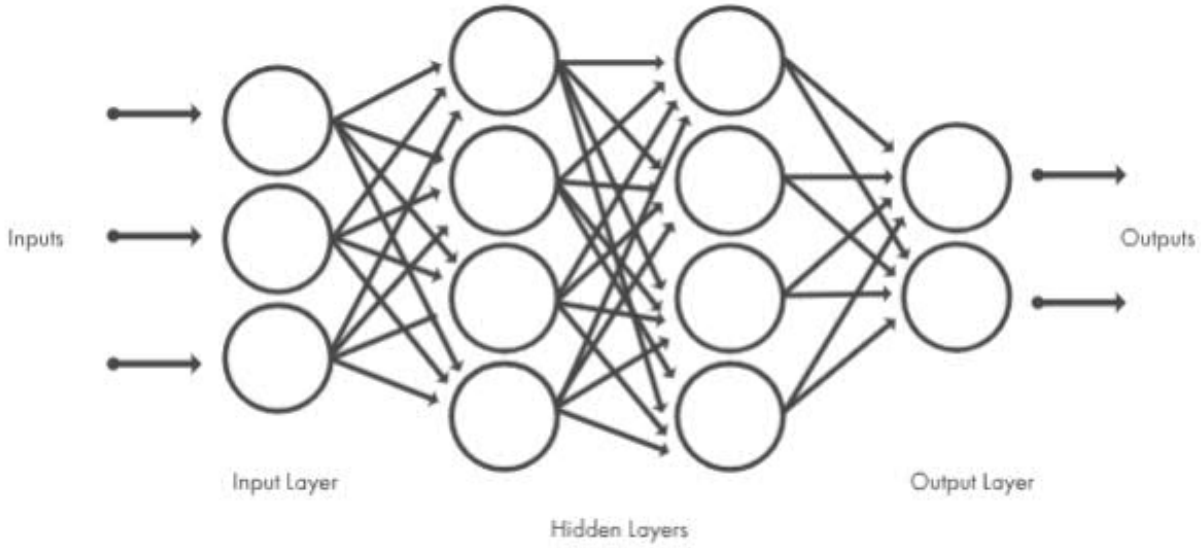


Рисунок 1.6 – загальна модель конволюційної нейронної мережі

Конволюційні нейронні мережі – це особлива архітектура штучних нейронних мереж, запропонована Яном Лекуном у 1988 році [5]. Одним з найпопулярніших застосувань цієї архітектури є класифікація зображень. Наприклад, Facebook використовує їх для автоматичного проставлення тегів, Amazon – для створення рекомендацій щодо продукту, а Google – для пошуку фотографій користувача.

Головним шаром згорткової нейронної мережі є конволюційний (згортковий), тому цей вид мережі і має таку назву. Шар складається з набору фільтрів, за допомогою яких відбувається прохід по вхідним даним і таким чином мережа навчається. В період перебору елементів кожен фільтр здійснює згортку за шириною та висотою вхідної ємності, обчислюючи скалярний добуток даних фільтру та вхідних, які представленні значеннями кожного з пікселів зображення, і формуючи двовимірну карту збудження цього фільтру (рисунок 1.7).

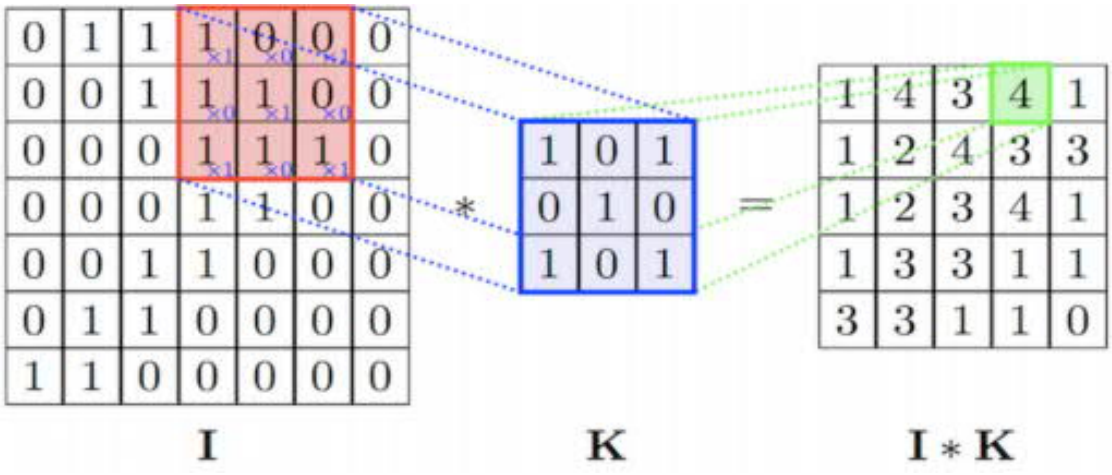


Рисунок 1.7 – схема роботи фільтру згорткового шару

Градієнтний спуск – це повторно застосовуваний алгоритм знаходження екстремуму дійсної функції першого порядку. Якщо здійснюються кроки пропорційно самому значенню градієнту, то відбувається наближення до локального максимуму цієї функції і ця процедура тоді відома як градієнтний підйом. Він може застосовуватися у будь-якій кількості вимірів. Може бути поєднаним разом із таким видом пошуку, як лінійним, що здійснює пошук локально оптимальний розмір кроку, проте це може бути неефективним у плані часу[13].

Градієнтний спуск відомий також як найшвидший спуск, або метод найшвидшого спуску.

Обмеженнями застосування градієнтного спуску є те, що для недиференційовних функцій градієнтні методи є недостатньо визначеними. Для локально ліпшицевих задач, та особливо для задач опуклої оптимізації, цілком визначеними є в'язкові методи спуску. Також можуть застосовуватися не-спускові методи, такі як методи субградієнтної проекції[37].

Було проаналізовано наступні основні методи вирішення задачі класифікації є:

– метод класифікації, заснований на наївному байєсівському класифікаторі, є алгоритмом навчання з учителем, в якому застосовується теорема Байеса із суворим (наївним) припущенням про незалежність між

кожними парами ознак [14]. Припущення про незалежність дозволяє позбутися від складної схеми оцінки параметрів класифікатора. Це дозволяє застосовувати алгоритм на великих вибірках. також класифікація виявляється досить точною: недостатньою для високоточних систем класифікації, однак задовільною для грубої оцінки та порівняння з іншими алгоритмами. Незважаючи на надмірно спрощені припущення, наївні класифікатори Байеса працювали досить добре у багатьох реальних ситуаціях таких як класифікація документів та фільтрація спаму.

− дерево ухвалення рішень (також можуть називатися деревами класифікацій або регресійними деревами) — використовується в галузі статистики та аналізу даних для прогнозних моделей. Структура дерева містить такі елементи: «листя» і «гілки». На ребрах («гілках») дерева ухвалення рішення записані атрибути, від яких залежить цільова функція, в «листі» записані значення цільової функції, а в інших вузлах — атрибути, за якими розрізняються випадки. Щоб класифікувати новий випадок, треба спуститися по дереву до листа і видати відповідне значення. Подібні дерева рішень широко використовуються в інтелектуальному аналізі даних. Мета полягає в тому, щоб створити модель, яка прогнозує значення цільової змінної на основі декількох змінних на вході[15].

− метод K-найближчого сусіда - один з методів вирішення задачі класифікації. Передбачається, що вже є якась кількість об'єктів з точною класифікацією (тобто для кожного них точно відомо, якого класу він належить). Потрібно виробити правило, що дозволяє віднести новий об'єкт до одного з можливих класів (тобто самі класи відомі заздалегідь) [16].

В основі метода k-найближчих сусідів лежить таке правило: об'єкт вважається належним того класу, до якого належить більшість його найближчих сусідів. Під «сусідами» тут розуміються об'єкти, близькі до досліджуваного в тому чи іншому сенсі. В якості метрики найчастіше обирається евклідова метрика через її простоту та зрозумілість. До недоліків метричних алгоритмів можна віднести зберігання всієї навчальної вибірки[17].

Оскільки вхідними даними для класифікації добового навантаження є числові значення, вибірка поповнюється даними – оцінками користувачів та необхідно приближувати дані класифікації до відгуків користувача, то було вирішено використовувати метод K-найближчого сусіда для задачі класифікації.

1.3 Інструменти реалізації

Сучасні програмні продукти можна поділити за категоріями – web-сайти, webдодатки, desktop-додатки, мобільні додатки. Оскільки основною ознакою розроблюваної системи є постійна доступність та широких аналіз активності користувача, то найкращим варіантом є використання смартфону для доступу. Web-сайти та web-додатки вимагають доступу до мережі, що може бути не постійним, також сповільнює роботу в порівнянні з offline мобільним додатком. Аналіз даних використання смартфона користувачем збільшить точність роботи нейронної мережі. Тому було вирішено реалізувати систему у вигляді мобільного додатку.

На сьогоднішній день iOS і Android є двома основними мобільними операційними системами які використовуються у повсякденному житті звичайних користувачів кожного дня[18].

Відомо, що єдиний виробник виробляє пристрої для iOS - Apple. Але є тисячі малих і великих компаній, які роблять пристрої для Android. Ця конкуренція знижує ціни, що призводить до збільшення частки ринку дешевими телефонами Android. Statcounter випустила графік, що відображає цю ситуацію на ринку (рисунок 1.8). В усьому світі близько 75% людей використовують Android, і лише 19% використовують iOS.

Рисунок 1.8 – Частка виробників на ринку

Нижче наведено карту, яка надасть вам більше інформації про налаштування платформи по всьому світу (рис 1.9).

Рисунок 1.9 – Демографічна карта платформ

Ця карта показує, що багато країн Європи, Південної Америки, Азії та Африки віддають перевагу Android. Деякі країни з вищими доходами, включаючи США, деякі європейські країни та Австралію, віддають перевагу iOS. Ці регіональні переваги можна частково пояснити низькою вартістю деяких мобільних телефонів Android. У середньому люди, які віддають

перевагу iOS, молодші за людей, які віддають перевагу Android, мають вищий рівень освіти і заробляють більше грошей.

Розробник з комп'ютером - може створити програму для Android. Linux, Windows, і навіть пристрої Mac можуть робити цю роботу. Розробка програми iOS вимагає від розробника наявність пристрою екосистеми Mac.

Немає сенсу вибирати старі технології для сучасних проектів. Нові мови програмування Kotlin і Swift поступово замінюють Java і Objective-C.

Ось коли на перше місце виникають питання сумісності. Kotlin повністю сумісний з Java. Ця повна сумісність означає, що ви можете використовувати всі численні фреймворки та бібліотеки для Java в проекті Котліна. Більше того, ви можете переключитися з однієї мови програмування на іншу з рядка в лінію.

З Swift речі не виглядають так добре. Objective-C та Swift не є повністю сумісними. Це створює безліч проблем і ускладнює розвиток. Крім того, кожна версія коду має різну сумісність, тому один фреймворк може бути більш сумісним з Swift 2, ніж з Swift 3.

Стисле порівняння платформ приведене на рисунку 1.9.

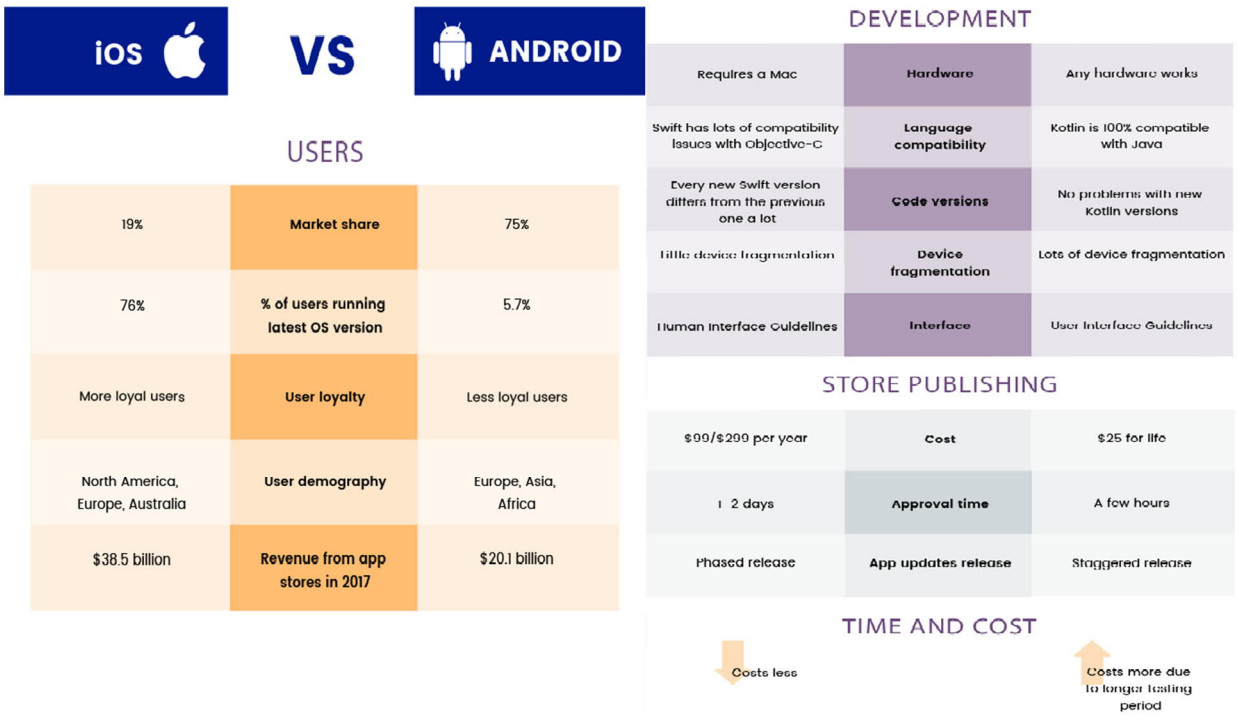


Рисунок 1.9 – Порівняння платформ

Оскільки Android є більш поширеною платформою, не потребує додаткової периферії та розробник має навички розробки на мові програмування Kotlin, Android було обрано платформою розробки.

1.4 Висновки до розділу

В даному розділі проведено аналітичний огляд підходів до створення системи телемедицини, що дозволило обрати найоптимальнішу мову програмування для отримання поставленої мети, визначити основні структурні елементи, провести порівняльний аналіз сучасних систем телемедицини та виділити їх переваги та недоліки.

Метою даного дипломного проекту є інформаційна модель планування задач на основі алгоритмів машинного навчання

Для досягнення поставленої мети потрібно виконати наступні задачі:

− провести огляд існуючих досліджень за темою створення планування розкладів та задач;

− провести огляд існуючих методів штучного, що застосовуються для планування;

− моделювання системи: проектування бази даних, створення алгоритмів, проектування інтерфейсу;

− розробка системи; тестування;

− введення в користування.

В даному розділі було досліджено аналоги та засоби для створення інформаційної моделі планування задач. Після проведення аналізу платформ було вибрано платформу Android як найпопулярнішу платформу для розробки мобільних додатків та мову Kotlin як найоптимальніший варіант для написання Android додатку. Проведено аналіз методів розробки системи штучного інтелекту та враховано їх переваги та недоліки.

2. МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ МОДЕЛІ ПЛАНУВАННЯ ЗАДАЧ

2.1 Архітектура системи

Діаграма прецедентів – в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (англ. use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою. Важливо розуміти, що дана діаграма являє собою погляд на систему з боку, тобто не потрібно шукати ніякої взаємодії між прецедентами та класами всередині системи.

В розроблюваній системі єдиним можливим актором є користувач додатку. Варіантами використання даної системи є її 4 основні функції. Дві з них, а саме Calendar та Notes, тобто використання сисеми, як звичайного календаря або ж записника справ є звичайними та поширинеми, мають достатньо аналогів.

Дві інші – «Scheduling» та «TodoList» є особливими прецедентами, що виділяють систему з поміж її аналогів. Система здатна аналізувати навантаження людини та планувати її завдання в оптимальний з точки зору навантаження час.

Проаналізувавши роботу програми та взаємодію користувача з нею можна зобразити наступну UML діаграму прецедентів зображену на рисунку 2.1.

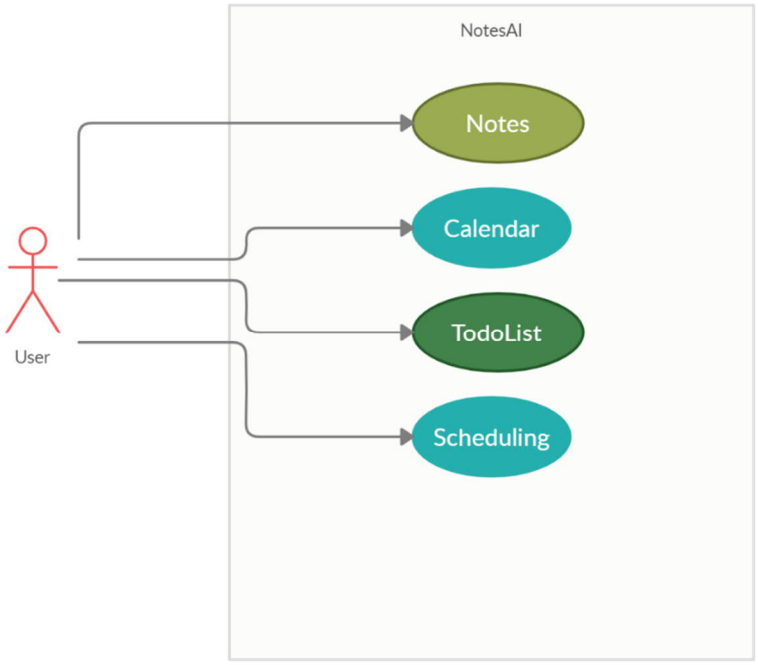


Рисунок 2.1 – Діаграма варіантів використання

Архітектура інформаційної системи – це концепція, що визначає модель, структуру, виконувані функції й взаємозв'язок компонентів інформаційної системи [19]. На даному етапі розроблена інтелектульна система має просту архітектуру, що складається з двох взаємодіючих компонентів – мобільного додатку, що включає в себе сторонні бібліотеки та sqlite бази даних.

Взаємодія бази даних з додатком здійснюється з використанням архітектурного шаблона MVVM (Model-View-ViewModel).

View – це абстракція для Activity, Fragment, або любого іншого UI елемента. View не має зберігати дані, а лише посилання на екземпляр ViewModel і всі дані які потрібні приходять саме звідти. Окрім того view повинна спостерігати за тими даними та мінятись відповідно до них.

ViewModel – це абстракція для класа який зберігає дані та логіку отримання та відображення цих даних. Також там зберігається посилання на один або декілька model класів звідки і отримує дані, не важливо це дані отримані з сервера чи з локальної бази даних. Також ViewModel не повинен знати про рівень абстракції view.

Model – це абстракція для класа який готує дані для ViewModel. Це клас в який програма отримує дані з сервера, одразу ж зберігаючи їх в локальну базу даних, або з бази даних якщо з'єднання з інтернетом відсутнє. Model не повинен містити ніяких згадок про ViewModel.

Даний паттерн дозволяє відокремити логіку додатку від візуальної частини (представлення). Даний шаблон є архітектурним рішенням, тобто він задає загальну архітектуру програми [26].

Кінцева архітектура програми зображена на рисунку 2.2.

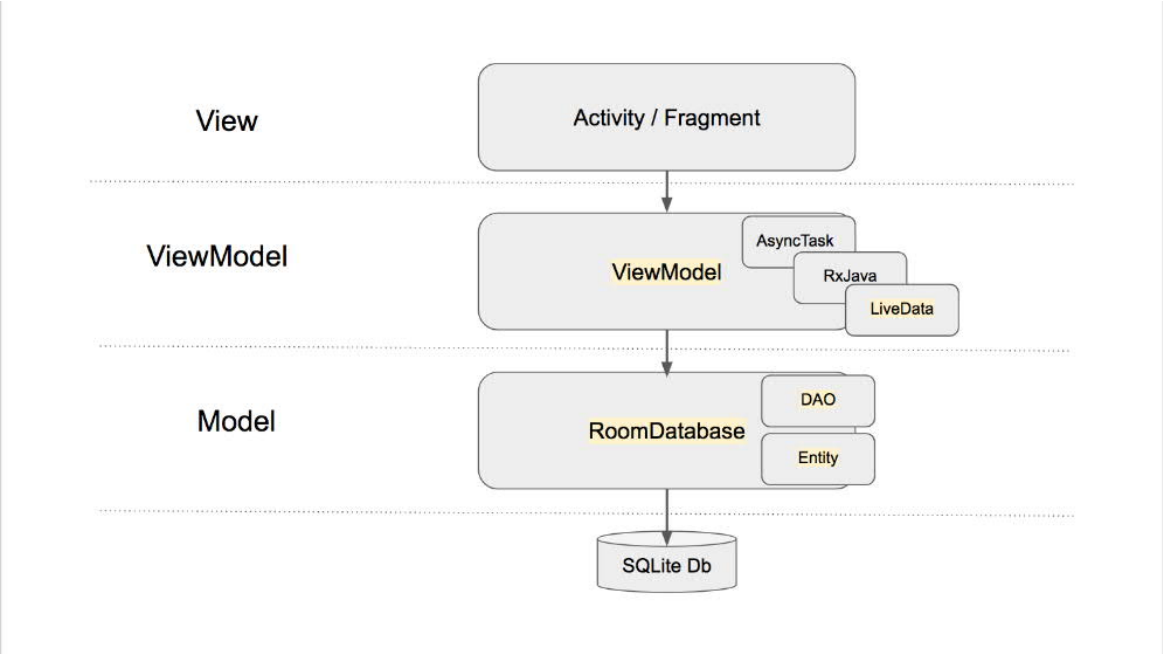


Рисунок 2.2 – архітектура програми

Даний підхід дозволить розділити програму на декілька логічних частин, що спрощує її подальшу розробку, тестування та підтримку старого коду.

Найкращим способом реалізації даного шаблону є створення декількох незалежних модулів програми які будуть взаємодіяти між собою, та модулі вищих рівнів не знатимуть про існування нижчих рівнів, що дозволить побудувати чисту архітектуру яка:

− буде легко покриватись тестам;

− не буде залежати від UI;

− не буде залежати від зовнішніх фреймворків і бібліотек.

Це все досягається поділенням логіки на шари як зображено на рисунку 2.3.

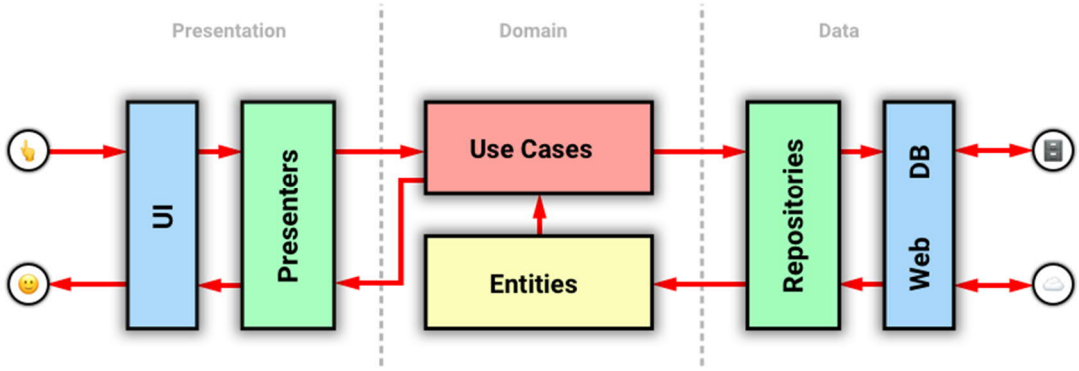


Рисунок 2.3 – схематичне зображення поділу мобільного додатку на шари

Також важливою частиною написання хорошого додатку є паттерн dependency injection(DI). DI – це шаблон проектування який забезпечує реалізацію принципа інверсії залежностей і реалізує правила створення об’єктів та незалежність реалізацій[29]. Принцип інверсії залежностей – один з п’яти SOLID-принципів об’єктно-орієнтованого проектування програм, суть якого полягає у розриві зв’язку між модулями нижчого та вищого рівнів за допомогою спільних абстракцій. Принцип формується наступним чином:

− модулі вищого рівня не повинні залежати від модулів нижчого рівня. Обидва типи модулів повинні залежати від абстракцій;

− абстракції не повинні залежати від деталей реалізації. Деталі реалізації повинні залежати від абстракцій.

Традиційні методи проектування програмного забезпечення схиляють до створення програмних структур, у яких модулі вищого рівня залежать від модулів нижчого та у яких абстракції залежать від деталей реалізації. Ці методи, крім всього іншого, мають на меті визначення ієрархії підпрограм, які описують, як модулі вищого рівня здійснюють виклики до модулів нижчого рівня. Тому структура добре спроектованої об'єктно-орієнтованої програми «інвертована» по відношенню до структури залежностей, яка є результатом традиційних процедурних методів проектування[30].

Бібліотека Dagger 2 володіє рядом переваг перед іншими Dependency injection бібліотеками. ЇЇ основна перевага це робота на принципі генерації коду без рефлекції, що означає всі помилки зв'язані з побудовою графа залежностей будуть виявлені та описані програмісту, ще на етапі компіляції проекту. А також генерація коду допомагає пришвидшити розробку проекту оскільки для ручної реалізації потрібно писати велику кількість стрічок коду який буде повторюватись додаючи лишню роботу програмісту. Схематичне представлення графа залежностей відображено на рисунку 2.4.

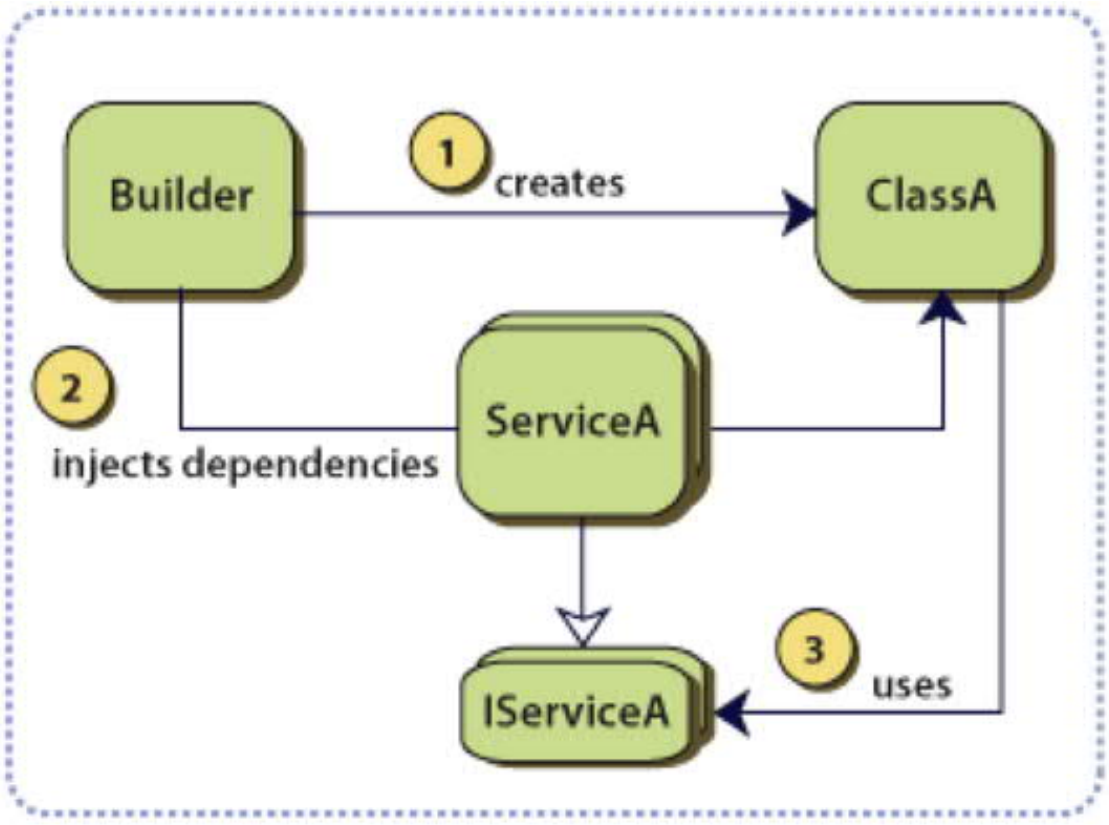


Рисунок 2.4 – Схематичне представлення залежностей

Як можна побачити на рисунку 2.4 залежності бувають різні. В додатку можуть бути залежні сервіси, активності, презентери та контролери. Всі ці сутності переплетені між собою зелжностями. Якщо спробувати вирази це графічно буде приблизно те, що зображено на рисунку 2.4.

Також бібліотека Dagger 2 володіє зручним та зрозумілим логуванням помилок. Приклад помилки яку вона повертає зображено на рисунку 2.5.

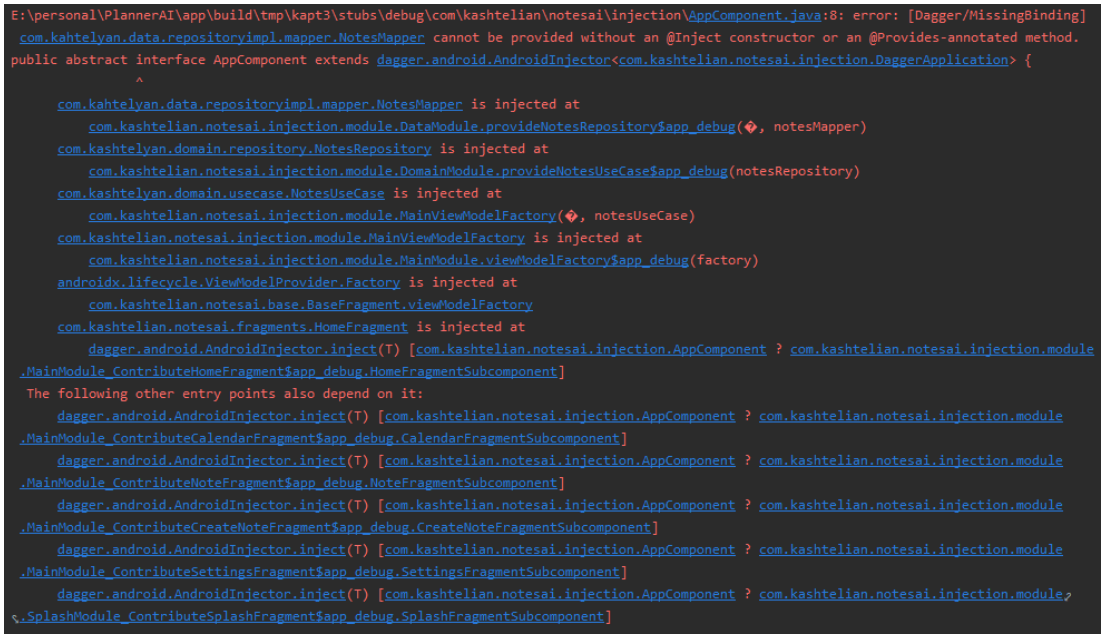


Рисунок 2.5 – приклад помилки залежностей в бібліотеці Dagger 2

Як зображено на рисунку 2.5 бібліотека на етапі компілювання проекту повідомляє користувачу, що він допустив помилки при розробці програми, а також вказує де саме вона та що потрібно виправити. Це набагато спрощує побудову архітектури проекту та добавляння нових компонентів оскільки користувач бачить помилку ще на етапі компілювання, що зменшує кількість помилок які можуть попасти в релізну версію програми.

2.2 Розробка бази даних

База даних (англ. database) – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; ця сукупність підтримує щонайменше одну з областей застосування [20]. В загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, крім саме даних, містить їх опис та може містити засоби для їх обробки.

Головним завданням бази даних – гарантоване збереження інформації та надання доступу до неї користувачеві або ж прикладній програмі.

Історично системи управління базами даних орієнтувалися на вирішення завдань, пов'язаних у першу чергу з транзакційною обробкою структурованої інформації. Безумовно, найкращим, перевіреним часом рішенням тут була і залишається реляційна модель СУБД. Однак в останні роки область застосування баз даних незмінно розширювалася. З одного боку, потрібно керувати більш широким набором форматів даних, переходячи до вирішення спільних проблем управління корпоративною інформацією. З іншого – саме СУБД(Database Management System) беруть на себе основні функції інтеграції даних і додатків корпоративних систем.

Ієрархічні бази даних можуть бути представлені як дерево, що складається з об'єктів різних рівнів. Верхній рівень займає один об'єкт, другий – об'єкти другого рівня. Мережеві бази даних подібні до ієрархічних, за винятком того, що в них є покажчики в обох напрямках, які з'єднують споріднену інформацію.

Незважаючи на те, що ця модель вирішує деякі проблеми, пов'язані з ієрархічною моделлю, виконання простих запитів залишається досить складним процесом. Також, оскільки логіка процедури вибірки даних залежить від фізичної організації цих даних, то ця модель не є повністю незалежною від програми.

Іншими словами, якщо необхідно змінити структуру даних, то потрібно змінити і додаток. За технологією обробки даних бази даних поділяються на централізовані й розподілені. Централізована база даних зберігається у пам'яті однієї обчислювальної системи. Якщо ця обчислювальна система є компонентом мережі ЕОМ (Електро́нна обчи́слювальна маши́на), можливий розподілений доступ до такої бази. Такий спосіб використання баз даних часто застосовують у локальних мережах ПК.

Розподілена база даних складається з декількох, можливо пересічних або навіть дублюючих одна одну частин, які зберігаються в різних ЕОМ

обчислювальної мережі. Робота з такою базою здійснюється за допомогою системи управління розподіленою базою даних (СУРБД) [14].

Концептуальна модель - це модель, представлена безліччю понять і зв'язків між ними, що визначають смислову структуру розглянутої предметної області або її конкретного об'єкта [21]. Концептуальна модель включає високорівневі конструкції даних та може бути не нормалізованою.

Основними одиницями розробленої системи є нотатки та завдання. Саме вони мають зберагтися в базі даних. Користувач буде виконувати операції додавання, редагування та видалення над ними, а система буде аналізувати за допомогою них навантаження людини, тому була виділена сутність «task» та «note».

Оскільки було вирішено, що користувач матиме змогу оцінювати сумарне навантаження саме дня, а не завдання та для подальшої підтримки швидкодії додатку, виділено сутність «analysis», яка буде масив вхідних даних звичок користувача, його режиму роботи, активностей, що буде використовуватись для роботи та навчання систем штучного інтелекту.

Таким чином на етапі концептуального моделювання виділено три сутності «task», «note», «analysis».

Логічні моделі даних подають абстрактну структуру області інформації. Вони часто мають схематичний характер і найтиповіше використовуються у бізнес процесах, які прагнуть захопити речі, що мають важливе для організації значення, та як вони відносяться одна до одної. Одного разу перевірена та схвалена, логічна модель даних може стати основою фізичної моделі даних і сформувати дизайн бази даних.

Логічні моделі даних повинні засновуватися на структурах, визначених у попередній концептуальній моделі даних, оскільки вона описує семантику інформаційного контексту, яку логічна модель повинна також відображати.

Оскільки логічна модель передбачає реалізацію на конкретній обчислювальній системі, вміст логічної моделі даних коригується для досягнення певної ефективності [22]. Також потрібно регулювати кількість

таких даних щоб уникнути приторможування системи при обробці даних оскільки платформа має обмежені ресурси.

Логічна модель даних включає сутності (таблиці), атрибути (колонки / поля) та відношення (ключі) та є нормалізованою.

Таблиця 2.1 – опис атрибутів task

Назва	Опис
_id	Первинний ключ таблиці task, який однозначно ідентифікує запис
name	Назва завдання
duration	Тривалість завдання
priority	Пріорітет завдання

Таблиця 2.2 – опис атрибутів analysis

Назва	Опис
_id	Первинний ключ таблиці analysis, який однозначно ідентифікує запис;
year	Рік
month	Місяць
day	День
total_effectivity	Сумарна ефективність
mark_total_effectivity	Оцінка користувача

На даному етапі було спроектовано базу даних для зберігання інформації користувача.

Для створення, збереження, редагування та видалення інформації в системі Android використовується база даних SQLite, яка є спрощеною версією бази даних SQL. Для роботи з базою пропонується використання фреймворка Room Database який спрощує роботу з вбудованим SQLite API. Він виконує велику частину роботи під час компіляції проекту генеруючи класи для роботи

з базами даних, тому розробнику не потрібно працювати з Cursor або Content Resolver які є складнішими у використанні та забирають багато часу на розробку.

Room має три основних компоненти: Entity, Dao і Database. Які позначаємо в коді за допомогою анотацій.

Entity – анотація яка представляє об'єкт дата класу який буде зберігатись в базі даних. Цей об'єкт після генерації буде представлений у вигляді таблиці, по замовчанню для імені таблиці використовується ім'я класу, але також можна вказати своє ім'я. Анотацією «@PrimaryKey» назначаємо певне поле ключем, кожен клас повинен містити хоча б один ключ, також ключ містить додаткове не обов’язкове поле «autoGenerate», який дозволяє включити режим «autoincrement» в якому база сама буде генерувати значення якщо його не буде вказано. Приклад реалізації анотації зображено на рисунку 2.6.

```
@Entity
public class Employee {

   @PrimaryKey
   public long id;

   public String name;

   public int salary;
}
```

Рисунок 2.6 – Приклад реалізації анотації Entity

Dao – анотація яка представляє об’єкт який буде взаємодіяти з базою даних. В ньому можна прописати запит вручну використавши анотацію «Query», або використати готові запити за допомогою які реалізовані за допомогою анотацій: insert, update, delete. Використання анотацій значно спрощує роботу з базами даних в системі андроїд та зменшує кількість необхідного коду для написання найпростіших операцій. Приклад реалізації анотації зображено на рисунку 2.7.

```
@Dao
public interface EmployeeDao {

    @Query("SELECT * FROM employee")
    List<Employee> getAll();

    @Query("SELECT * FROM employee WHERE id = :id")
    Employee getById(long id);

    @Insert
    void insert(Employee employee);

    @Update
    void update(Employee employee);

    @Delete
    void delete(Employee employee);

}
```

Рисунок 2.7 – Приклад реалізації анотації Dao

Database – анотація яка представляє основний клас по роботі з базою даних, цей клас повинен бути абстрактним та наслідувати RoomDatabase. В параметрах анотації вказуємо які Entity будуть використовуватись, та версію бази даних. Для кожного класа зі списку «entities» буде створена таблиця[27]. Приклад реалізації анотації зображено на рисунку 2.8.

```
@Database(entities = {Employee.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract EmployeeDao employeeDao();
}
```

Рисунок 2.8 – Приклад реалізації анотації Database

Операції по роботі з базою даних синхронні та не повинні виконуватись в UI потоці, при виконанні будь-якої операції в цьому потоці розробник отримає «RuntimeError», що призведе до припинення роботи програми. Для Роботи з такими операціями рекомендується LiveData або RxJava.

RxJava – сховище даних яке працює по принципу патерна observer, туди можна помістити будь який об’єкт в одному місці програми, та підписавшись на нього отримати зміни в об’єкті чи новий об’єкт в іншому. Також має великий набір та зручний функцій для роботи з потоками, та з іншими спостерігачами.

LiveData – працює по принципу observer, натомість має проблеми при роботі з різними масивами даних з яких потрібно зробити одну модель даних[28].

Отже, реляційні таблиці будують за певним критерієм структурування. Необхідність такого структурування зумовлена прагненням систематизувати величезні масиви даних й автоматизувати пошук і селекцію їх компонентів.

У результаті дослідження основних етапів розробки баз даних було розроблено структуру бази даних для функціонування системи планування задач.

2.3 Розробка алгоритмів

Система виділяється з-поміж інших аналогів двома функціями, а саме Scheduling (планування завдань).

Планування завдань є основною функцією розробленої системи. Було вирішено розробити евристичний алгоритм головною ціллю якого є мінімізація різниці між середнім значенням витрат часу та ефективності виконання задач розрахованим за параметрами користувача та фактичними витратами. Таким чином для кожного дня виконується умова представлена нижче.

$$\left|E_{\text{avarage}} - E_{\text{fact}}\right| \rightarrow \min, \tag{2.1}$$

Де $E_{avarage}$ – середнє добовове навантаження,

E_{fact} – фактичне сумарне навантаження дня.

Дана мінімізація може бути досягнена якщо завдання планувати на найменш завантажений день з можливих.

Не менш важливою частиною даного алгоритму є обрання часу початку події. Оскільки всі завдання проведені користувачем зберігаються в базі даних, то можливо знайти найчастіше вживаний час для кожного завдання з певним

ім’ям. Якщо користувач найчастіше ходить на роботу на 9 годину ранку, то система має запропонувати йому саме 9 годину ранку для завдання з назвою «work», що і означає врахування специфіки діяльності людини при плануванні окремих завдань.

Таким чином функція планування завдання працює за наступним алгоритмом.

1. Користувач вводить параметри завдання, а саме назву, тривалість, пріорітет завдання, дати між якими має відбутися завдання.

2. Відбувається пошук днів з заданого проміжку, в яких є вільний проміжок часу, що дорівнює тривалості задачі.

3. З результуючого набору днів обирається найменш навантажений.

4. Початок завдання призначається на час першого вільного проміжку часу, що відповідає тривалість завдання.

5. Проводиться пошук найчастіше використовуваного початку завдання, якщо таке завдання зустрічалося раніше.

6. Якщо часовий період з найбільш використовуваним початком є вільним, то початок даного завдання прирівнюється до найчастіше вживаного, якщо ні – залишається.

Алгоритм планування завдань представлений на рисунку 2.8.

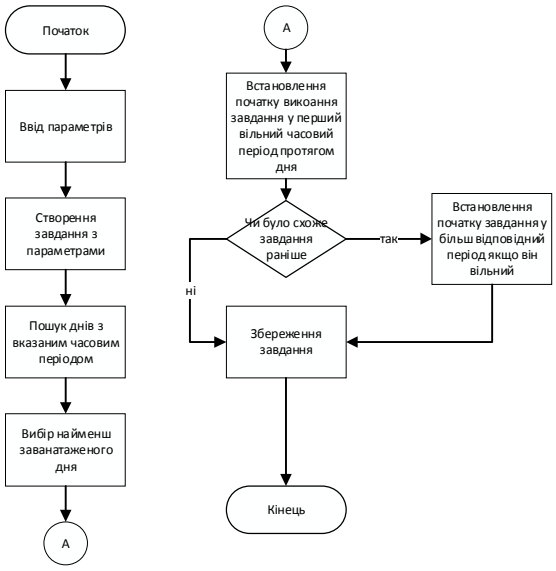


Рисунок 2.8– Алгоритм планування завдань

Другою основною функцією системи є класифікація добового наванатження користувача. Ця фуннкція наочно надає користувачу можливість зрозуміти які дні потрібно навантажити, а які навпаки розвантажити.

Середньо-добове навантаження людини розраховується за формулою Харріса Бенедикта. Це значення, окрім параметрів людин таких як стать, вага, зріст, вік, також залежить від типу активності людини, що відображається у відповідному коефіцієнті. Таким чином «неактивному» типу відповідає коефіцієнт 1,53, «активному» - 1,76, «спортивному» – 2,25[31].

Для збільшення зручності користування додатком було вирішено спростити дану формулу, та використовувати лише дані використання смартфона, аналізуючи які можна визначити основні періоди активності користувача, а також періоди найбільшої продуктивності.

Було вирішено класифікувати навантаження людини за п’ятьма класами: ненавантажений, недовантажений, нормально навантажений, перевантажений, занадто перевантажений.

Для початкової класифікації створюється стартовий набір даних наступним чином (на прикладі активного типу):

1. Після аналізу активності використання смартфона користувачем програми розраховується середньо-добове навантаження людини.

2. Розраховуються значення навантажень для неактивного та спортивного типу.

3. Розраховуються проміжні значення між трьома точками та їх віддзеркалення від крайніх точок, та віддзеркаленням крайніх проміжних точок. Таким чином створюється 5 відрізків, що відповідають класам навантаження, приклад представлений на рисунку 2.9.

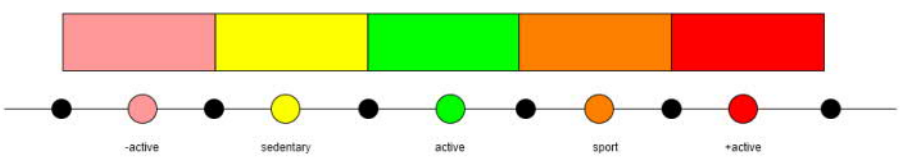


Рисунок 2.9 – Створення меж класів навантаження

Було проаналізовано алгоритм створення стартового датасету за методом Харріса Бенедикта який представлений на рисунку 2.10.

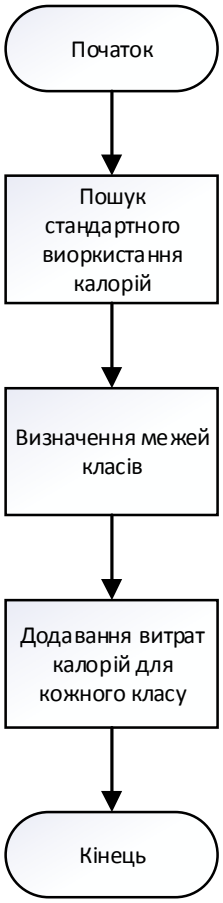


Рисунок 2.10 – Алгоритм створення стартового набору даних

Всього стартовий набір даних складається з 50 значень, що є мінімальним значенням для класифікації. На основі цих даних система класифікує добові навантаження людини у відповідності до п’яти класів.

Методом класифікації було обрано метод k найближчих сусідів. Класифікація кожного дня проходить за наступним алгоритмом:

1. Розраховується сумарне навантаження дня

2. Сортуються всі оцінені дні датасету за евклідовою відстанню між їхніми навантаження та навантаженням дня, що треба класифікувати

3. Розглядаються перші k відсортованих днів та відповідно обирається клас, що найбільше зустрічався

Алгоритм методу k найближчих сусідів представлений на рисунку 2.7.

Рисунок 2.11 – Алгоритм методу k найближчих сусідів

Значення k розраховується як квадратний корінь з кількості даних датасету, як один з поширених підходів його розрахунку [22].

Оскільки середньо-добове навантаження розраховується за формулою та енерговитрат кожного завдання та кількості виконаних та наявних завдань є табличними значеннями та можуть не точно відображати завантаженість користувача, було вирішено поповнювати датасет оцінками користувача, тим самим робити додаток ближчим та пристосованим до конкретного користувача. Таким чином користувач має змогу оцінити кожен попередній день оцінкою, що відповідає класам навантаження, визначених раніше та прямо впливає на роботу додатку, а саме класифікацію. А також користувач має змогу оцінити кожну пропозицію датасету окремо, що зможе підвищити точність рекомендацій.

Користувач може змінювати свої параметри, такі як вік, вага, зріст, тип життєдіяльності тому було вирішено при кожній зміні параметрів перші 50 записів таблиці бази даних «analysis» видаляються та заповнюються відповідно новими розрахованими значеннями.

Також для збільшення точності результатів було прийнято рішення спостерігати за активністю використання смартфона користувачем. Таку можливість в системі Android надає Broadcast Receiver. Broadcast Receiver – це компонент системи, який дозволяє реагувати додатку на повідомлення які розсилаються усім додаткам які підписані на певні події[41]. Широкомовні розсилки працюють наступним таким чином, що в рамках події «публікація - підписка» – подія викликає публікацію і отримання цієї події компонентами які зацікавлені нею.

Система Android виділяє два типи широкомовних розсилок:

− Явна трансляція яка призначена для конкретного додатку. Найбільш розповсюдженим прикладом використання це запуск певної дії в системі. Наприклад коли додатку потрібно набрати номер телефону, він відправить подію яка призначена для телефонного додатку Android, разом з цією дією буде відправлено номер телефона для набору, після цього система перенаправить цю подію в цей додаток.

− Неявна трансляція представляє собою розсилки повідомлень всім додаткам на смартфоні які зацікавлені в цих подіях. Наприклад подія «ACTION_POWER_CONNECTED» публікується кожного разу коли система помічає те, що пристрій заряджається та направляє цю подію у всі додатки які зареєструвались для даної події.

Таким чином відповідно до такої інформації як включення виключення екрану смартфона, відкриття різноманітних додатків системи штучного інтелекту можуть формувати модель поведінки користувача. На основі якої за допомогою метода k найближчих сусідів можна генерувати рекомендації для ефективного проведення дня, тижня, місяця.

Оскільки підписка на різноманітні події системи Android збільшує споживання батареї для оптимізації було розглянуто життєвий цикл додатку зображеного на рисунку 2.12.

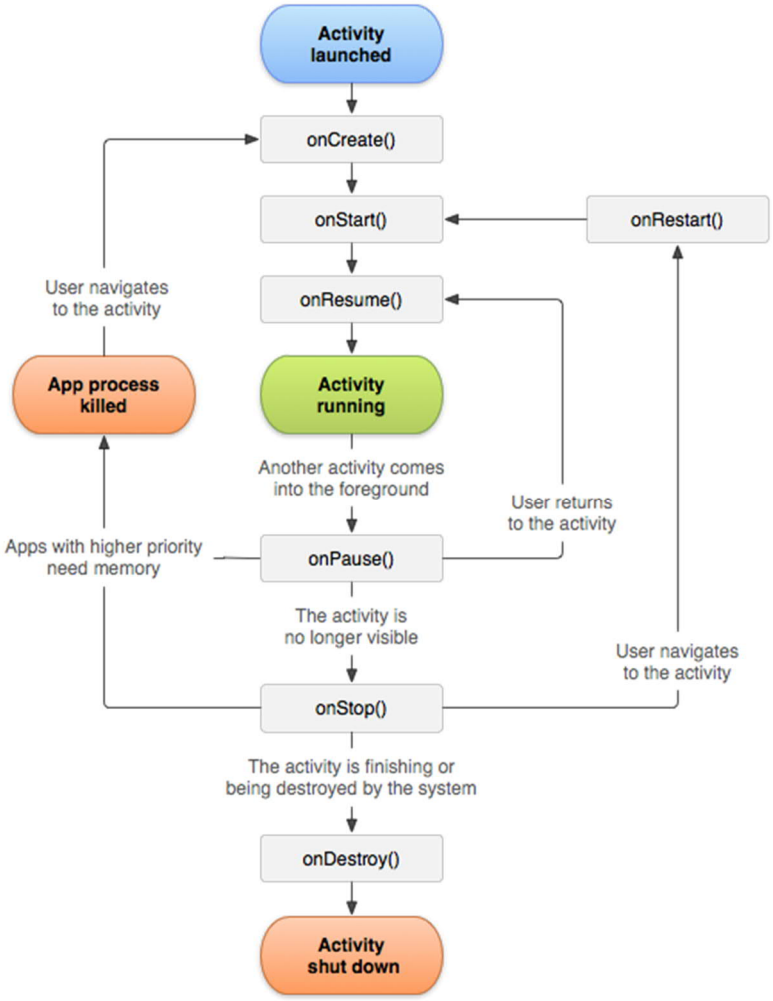


Рисунок 2.12 – життєвий цикл Android додатка

Життєвий цикл додатку в Android жорстко контролюється системою та залежить від потреб користувача та потрібних ресурсів для роботи системи та відкритого аплікейшина. Якщо в конкретний момент часу користувач використовує один додаток то йому надається пріоритет та система буде звільнювати для нього пам’ять шляхом призупинення або викидання з оперативної пам’яті інших додатків з меншим пріоритетом.

Для оптимізації додатку будуть використовуватись методи «onCreate» та «onDestroy».

Метод onCreate викликається при створенні або перезапуску додатку. Система може запускати і призупиняти вікна додатків в залежності від подій які відбуваються. Отож даний клас ідеально підходить для створення сервісу підписки на визначені події на смартфоні користувача. Оскільки методо може

бути перезапущеним потрібно перевіряти об’єкт «Bundle» який приймає даний метод під час перезапуску, якщо він не дорівнює «null» отже цей мето вже викликався та підписка на події не потрібна.

Метод onDestroy викликається по завершенню роботи додатку у випадку якщо система або користувач закривають додаток для вивільнення ресурсів. Саме в цьому методі доцільним буде відписатись від подій на які підписується аплікейшин при початку роботи.

Таким чином маючи доступ до системних подій таких як увімкнення - вимкнення екрану смартфона, вхідні - вихідні дзвінки, відкривання різноманітних додатків та підключення телефону до зарядного пристрою дає можливість сформулювати модель поведінки активності користувача та визначити часовий період в який користувач буде доступний для вирішення задач, а в які періоди він відпочиває, спить, чи заряджає телефон. Чим довше користувач використовує додаток тим точніше будуть рекомендації.

Отже можна сформувати наступний алгоритм роботи кінцевої програми.

1. Реєстрація спостерігача за подіями.

2. Створення нових або перегляд існуючих завдань.

3. При перегляді існуючих завдань їх можна редагувати та міняти дату виконання, а при створенні нових заповнення назви завдання, пріоритету та додавання часу виконання.

4. При додаванні часу система порекомендує оптимальний час для виконання завдання спираючись на даних користувача, завантаженості дня та режиму конкретного користувача.

5. Користувач може прийняти рекомендацію та задіяти запропоновані параметри, а також може ввести свої.

6. Збереження завдання.

7. Коли прийде час виконання завдання система нагадає за допомогою системи сповіщень про початок його виконання.

8. Користувач може помітити завершення завдання або ж перенести його на іншу дату.

9. Завершення роботи програми, відписка спостерігача від системних подій.

Алгоритм роботи системи зображено на рисунку 2.13.

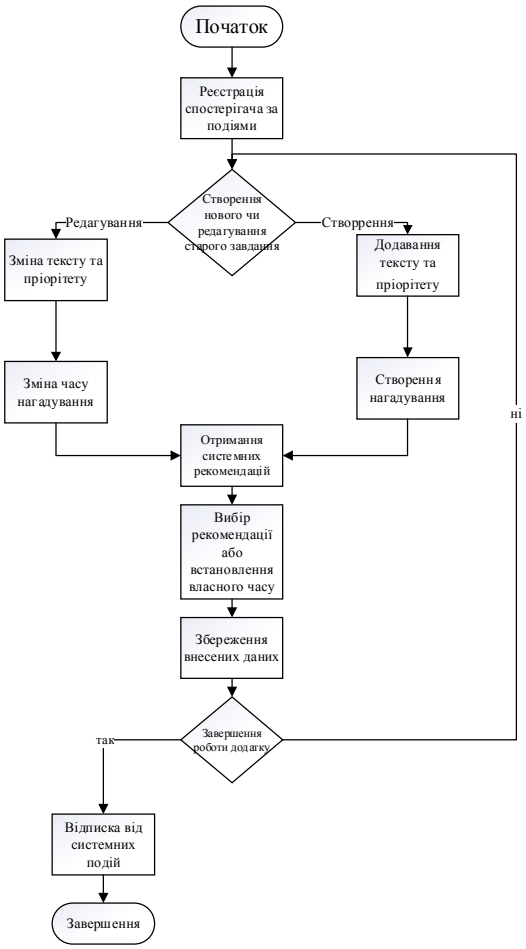


Рисунок 2.13 – алгоритм роботи програми

2.4 Висновки до розділу

Було розроблено максимально простий алгоритм роботи користувача з програмою оскільки ускладнення взаємодії з програмою, збільшення кількості дій для виконання простого завдання та не інтуїтивний інтерфейс програми ведуть до зменшення цільової аудиторії користувачів.

В даному розділі було проаналізовано алгоритми машинного навчання та вибрано найоптимальніший для використання в розроблюваній інформаційній системі планування задач. Основою системи було обрано алгоритм k найближчих сусідів. Було проаналізовано та описано шаблон проектування MVVM який буде використовуватись для розробки продукту. Було спроектовано структуру бази даних. Було розроблено та описано алгоритми роботи програми та її взаємодії з користувачем.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ МОДЕЛІ ПЛАНУВАННЯ ЗАДАЧ НА ОСНОВІ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ

3.1 Реалізація додатку

Для розробки програми було використано архітектуру MVVM та сам код буде розподіллено на незалежні модулі які створюються за допомогою інтерфейсу IDE Android Studio. Вікно вибору необхідного модуля зображено на рисунку 3.1.

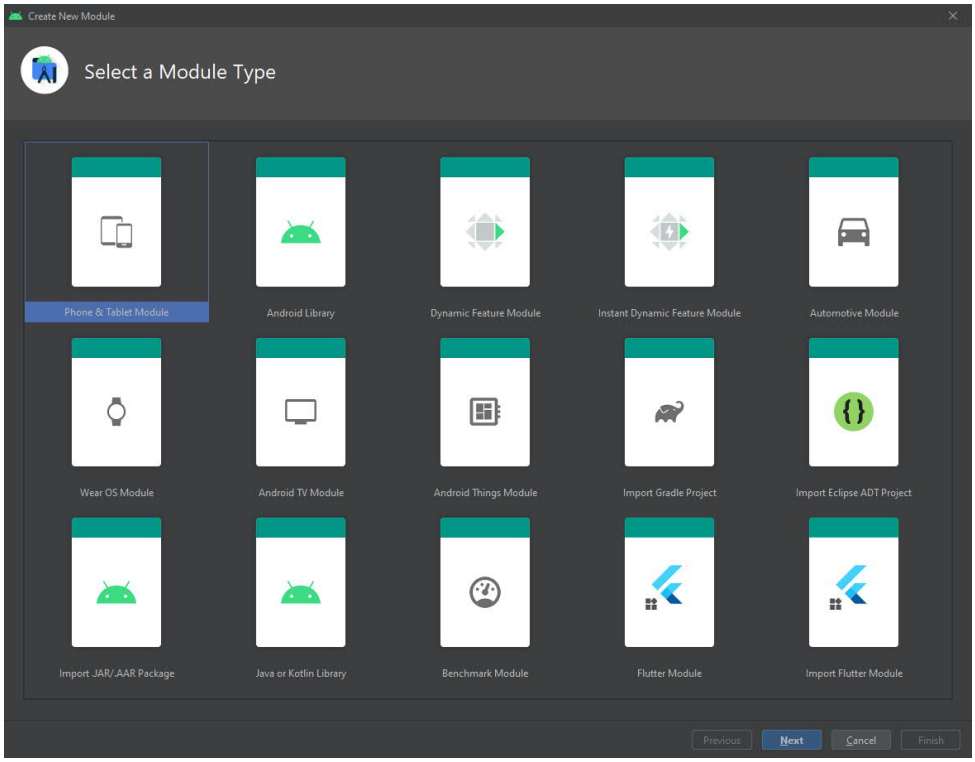


Рисунок 3.1 – вікно вибору модуля в середовищі розробки Android Studio

Було створено два додаткові модулі окрім модулі «app» який створюється автоматично при створенні проекту в Android Studio. Для модуля «domain» який буде використовуватись для зберігання моделей, та передання даних з модуля з даними. Для модуля «data» було використано тип модуля «Android Library» оскільки для створення та роботи з базою даних потрібний контекст

який можна отримати з даного модуля. Список модулів з їх внутрішньо структурою зображено на рисунку 3.2.

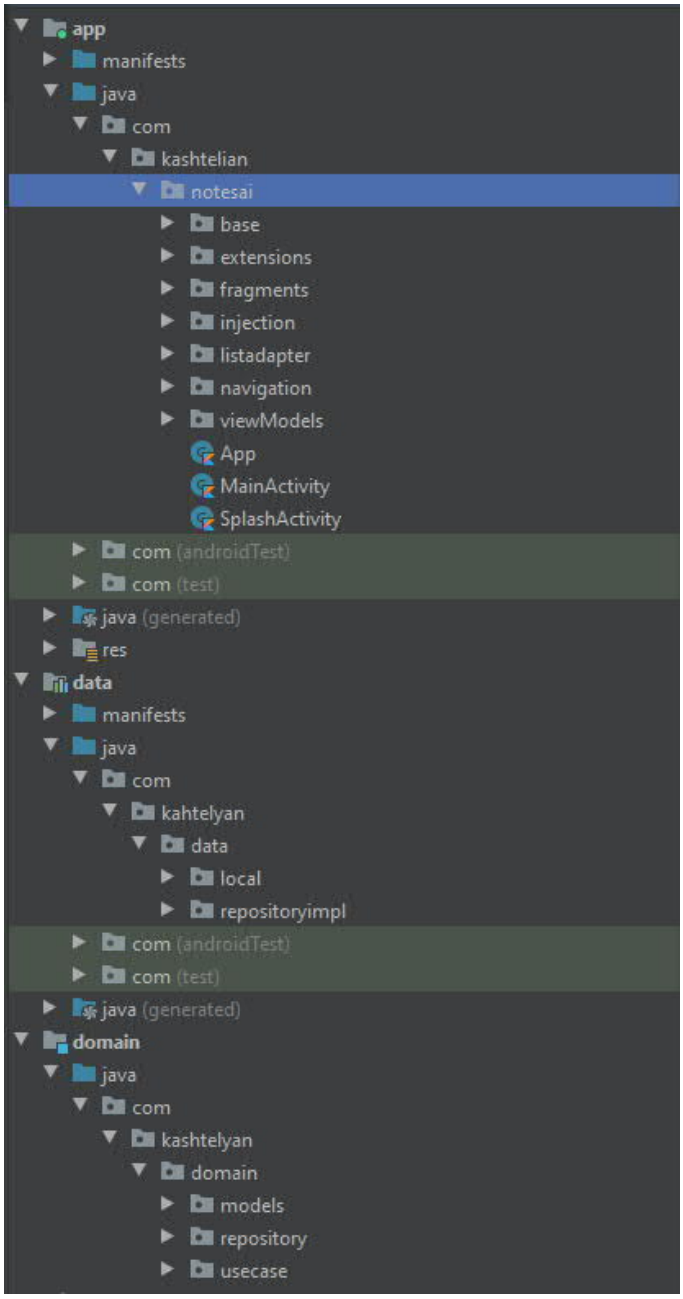


Рисунок 3.2 – список модулів програми разом із їх внутрішньою структурою

Для кожного з модулів автоматично створюється файл збірки Gradle в якому є можливість додавати в проект нові бібліотеки, доступи до інших модулів, та налаштовувати параметри білда андроїд застосунку.

Для підключення бібліотеки Dagger 2 було створено абстракний клас який наслідує клас Application() та інтерфейси: HasActivityInjector,

HasSupportFragmentInjector. Реалізація класу для dependency injection зображено на рисунку 3.3.

Рисунок 3.3. – реалізація класу для інверсії залежностей

База даних створюється програмно при першому запуску додатку на пристрої за допомогою класу NotesDatabase, що наслідує клас для роботи з базами даних SQLite - SQLiteDatabase.

Наявність таблиць, поля та типи полів кожної можливо перевірити шляхом відкриття бази даних в «Database Inspector» який вбудований у Android Studio , приклад представлено на рисунку 3.4.

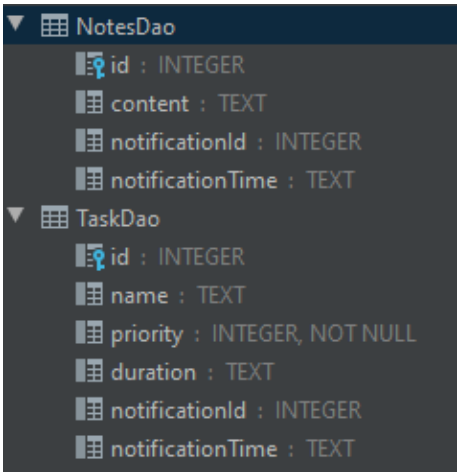


Рисунок 3.4 – Перегляд бази даних за допомогою вбудованого «Database Inspector»

Для роботи додатку з таблицями бази даних були створені відповідні дата класи, та інтерфейси для роботи з базою даних, перелік яких зображений на рисунку3.5.

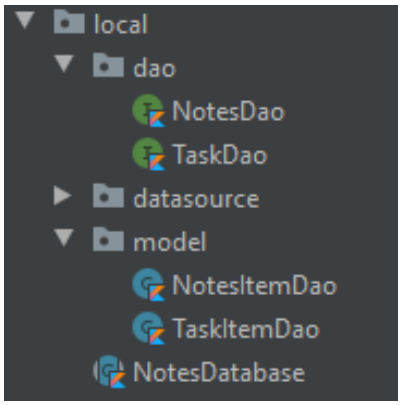


Рисунок 3.5 – Дата класи та інтерфейси для роботи з базою даних

В класі «NotesDatabase» виконується ініціалізація та створення бази даних при запуску програми, а також встановлюється версія бази даних та логіка міграції даних при оновленні версії. Реалізація класу зображена на рисунку 3.6.

```
@Database(entities = [NotesItemDao::class, TaskItemDao::class], version = 1, exportSchema = false)
abstract class NotesDatabase : RoomDatabase() {

    abstract fun notesDao(): NotesDao

    companion object {
        fun newInstance(context: Context): NotesDatabase {
            return Room.databaseBuilder(context, NotesDatabase::class.java,  name: "notes-dao.db")
                .fallbackToDestructiveMigration()
                .build()
        }
    }
}
```

Рисунок 3.6 – реалізація класу NotesDatabase

Також було створено базові класи для viewModel та fragment для спрощення добавляння функціоналу оскільки не потрібно кожного разу писати базові необхідні імплементації та реалізації, достатньо всього лиш наслідувати абстрактний клас. Реалізація класу BaseFragment зображена на рисунку 3.6.

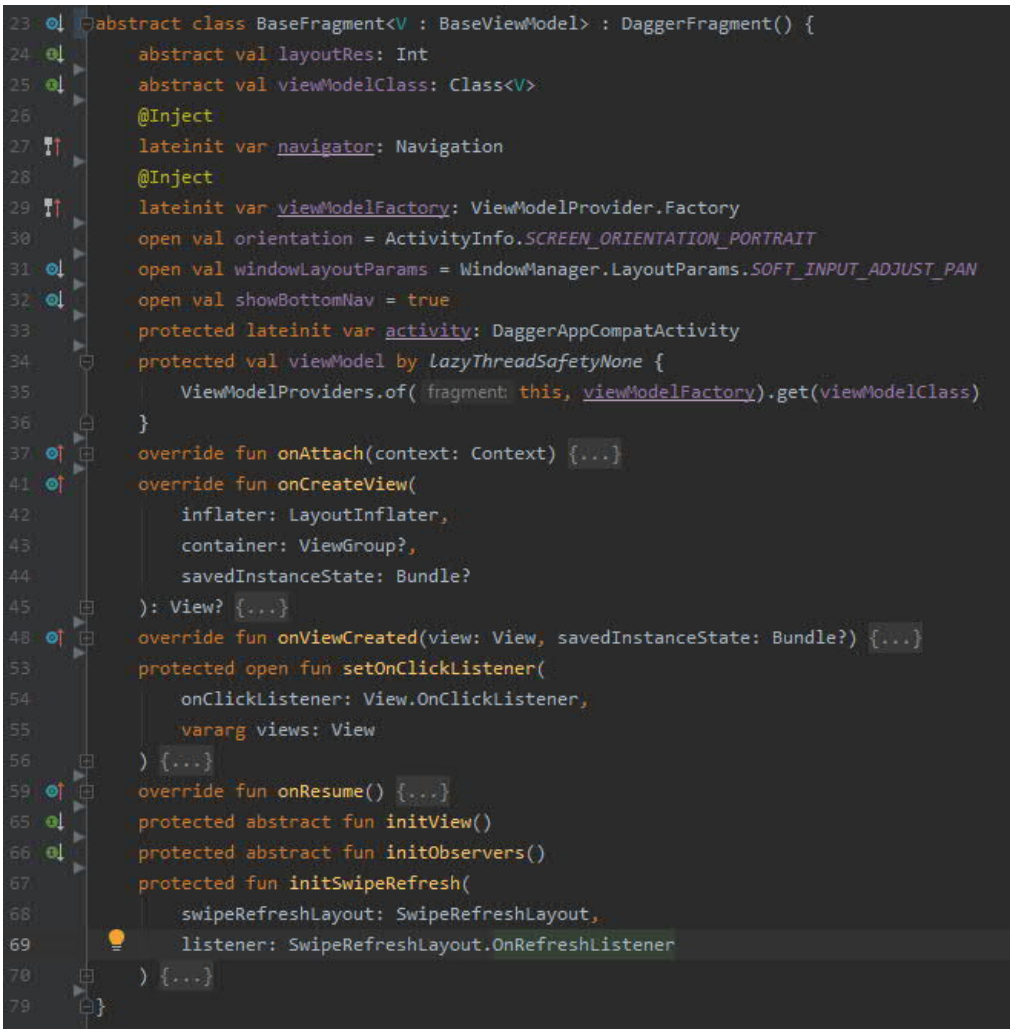
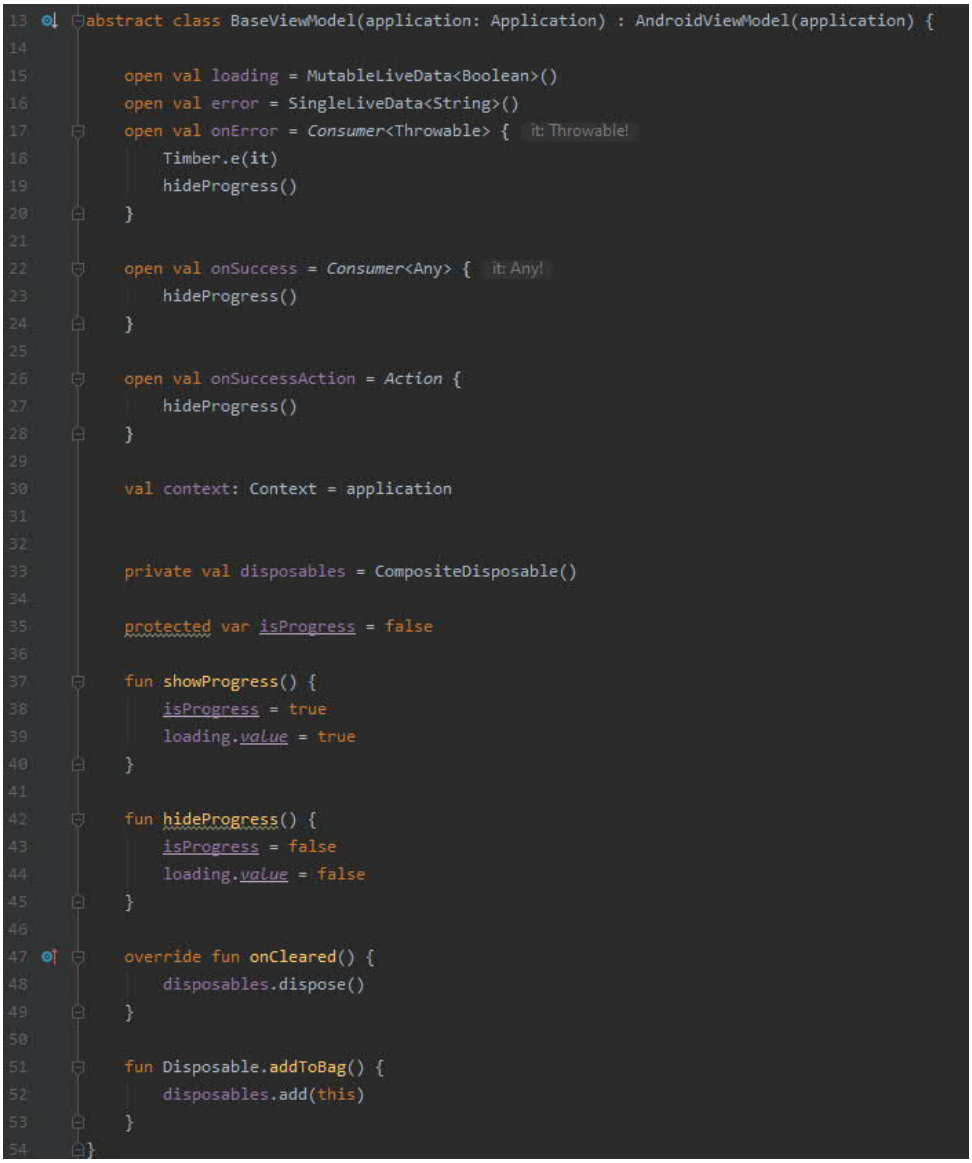


Рисунок 3.6 – реалізація абстрактного класу BaseFragment

Обов’язковим параметром при імплементації базового фрагмента є клас ViewModel який наслідує BaseViewModel який використовується згідно з патерном MVVM тобто надає інформацію та реалізує бізнес логіку. Ініціалізація класу відбувається за допомогою лінивої ініціаліазації яка є доступною з різних потоків. Лінива ініціалізація є особливістю мови програмування Kotlin вона означає, що змінна буде ініцалізована одразу після першого її виклику в коді. Також за допомогою інверсії залежностей будо добавлено інтерфейс Navigation який реалізовує для навігації в додатку. Також було створено обов’язкові для реалізації методи initView та initObservers в яких ініціалізуються ui елементи та спостерігачі за змінами у ViewModel. Реалізація базового класу ViewModel зображена на рисунку 3.7.



```
abstract class BaseViewModel(application: Application) : AndroidViewModel(application) {

    open val loading = MutableLiveData<Boolean>()
    open val error = SingleLiveData<String>()
    open val onError = Consumer<Throwable> {  it: Throwable!
        Timber.e(it)
        hideProgress()
    }

    open val onSuccess = Consumer<Any> {  it: Any!
        hideProgress()
    }

    open val onSuccessAction = Action {
        hideProgress()
    }

    val context: Context = application


    private val disposables = CompositeDisposable()

    protected var isProgress = false

    fun showProgress() {
        isProgress = true
        loading.value = true
    }

    fun hideProgress() {
        isProgress = false
        loading.value = false
    }

    override fun onCleared() {
        disposables.dispose()
    }

    fun Disposable.addToBag() {
        disposables.add(this)
    }
}
```

Рисунок 3.7 – реалізація абстрактного класу BaseViewModel

Обов’язковим параметром є клас Application який використовується для підтримки глобального класу програми та потрібний для імлплементації AndroidViewModel. Також були добавлені базові LiveData такі як завантаження та помилок. Змінна dissposables яка потрібна для очистки пам’яті від Observables які використовуються для передачі даних з інших модулів та є частиною реактивної бібліотеки RxJava 2.

Для контролю версій програми було підключено систему контролю версій Git та створено базові вітки master і develop. Гілка develop використовується для розробки з неї створюються нові вітки та додаються нові фічі, а master буде призначений для контролю релізних версії в плеймаркет.

3.2 Робота інформаційної моделі планування задач

При запуску додатку користувач бачить так званий SplashScreen з іконкою, під час роботи якого вмикається тема додатку, а після він потрапляє на головний екран додатку в якому має можливість створювати об’єкти, які будуть зберігатись та аналізуватись програмою. Головний екран додатку зображено на рисунку 3.8.

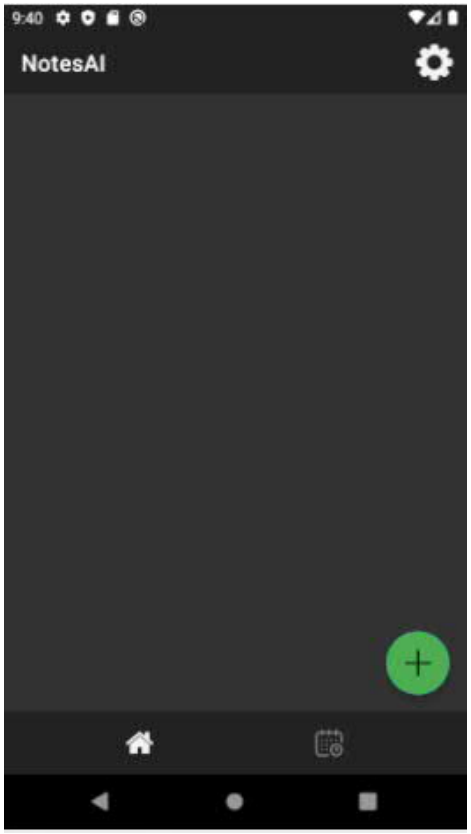
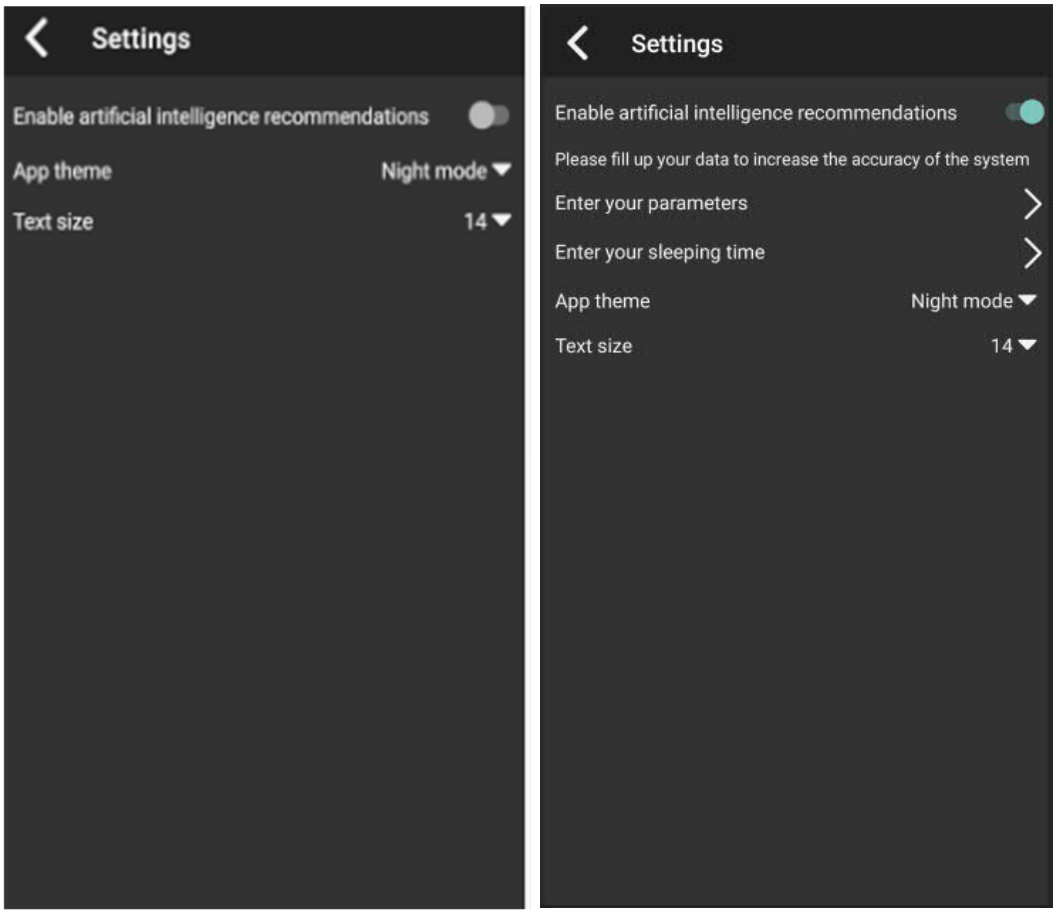


Рисунок 3.8 – Головний екран додатку після першого запуску

Перейшовши на вікно налаштувань користувач має можливість увімкнути рекомендації штучного інтелекту, після його увімкнення з’являться поля в яких користувачу буде запропоновано заповнити дані для покращення точності рекомендацій. На рисунку 3.9(а) та 3.9(б) зображено вікна програми до та після включення рекомендацій.

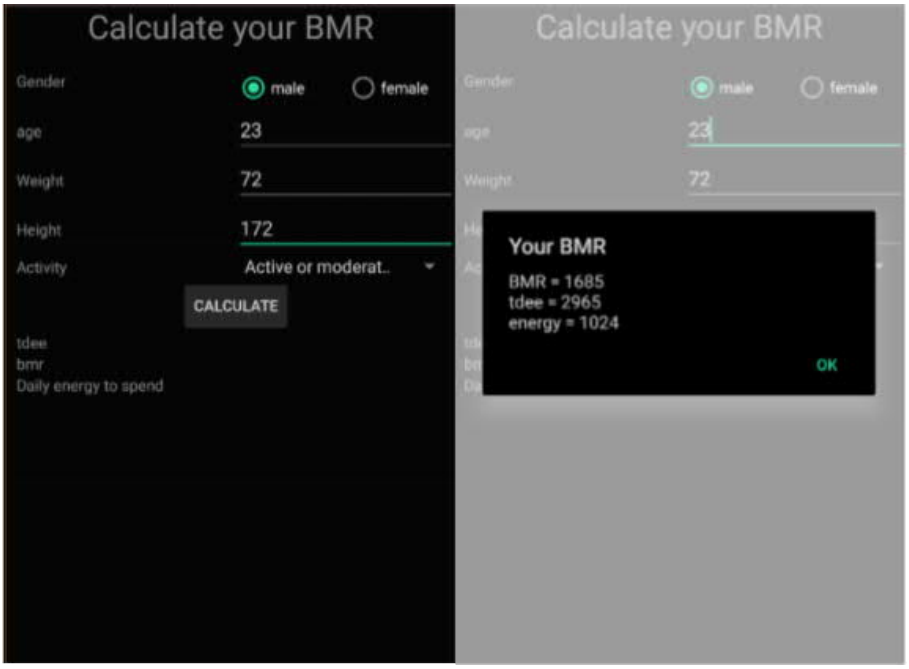


а) б)

Рисунок 3.9 – вікно програми до(а) та після(б) увімкнення рекомендацій штучного інтелекту

Відкривши вікно введення параметрів користувача він має можливісь заповнити: стать, вік, зріст, вагу, тип життєвої активності. Дані параметри будуть використовуватись для розрахунку добового навантаження на користувача та за допомогою рекомендацій допомагатимуть правильно та ефективно спланувати день. Вікно заповнення параметрів зображено на рисунку 3.10(а).

Після заповення усіх параментрів користувачу необхідно натиснути на кнопку «Calculate» та програма розрахує відповідно TDEE, BMR, добову енергію для витрат, про що користувач отримає сповіщення, представлене на рисунку 3.10(б).



а) б)

Рисунок 3.10 – Стартове вікно (а) та сповіщення з розрахованими параметрами TDEE, BMR, середніх добових енерговитрат (б)

Також користувач має можливість заповнити свій стандартний графік сну та програма відповідно не даватиме рекомендацій на цей період. Вікно введення годин сну користувача, зображене на рисунку 3.11.

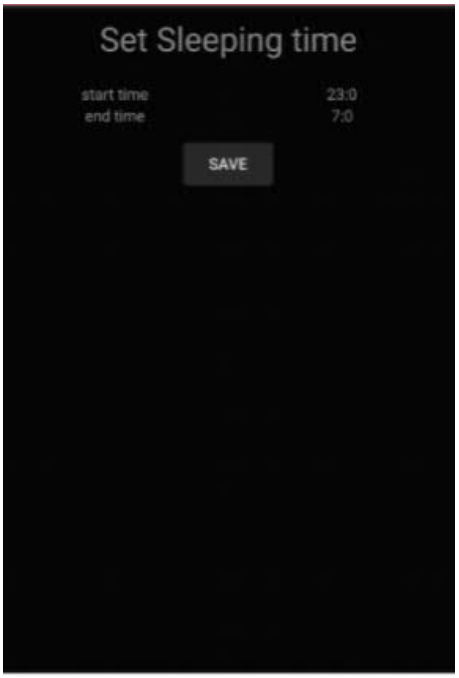


Рисунок 3.11 – вікно заповнення годин сну користувача

Оскільки години сну можуть мінятись, система аналізуватиме час першого та останнього увімкнення екрану смартфона протягом доби, внаслідок чого визначатиме час сну на основі чого визначатиме максимальну завантаженість протягом наступного дня. Та на окремому екрані програми побачити свою продуктивність протягом дня.

Для реалістичного приведення прикладу роботи додатку база даних була заповнена тестовими даними. При вході в додаток головне вікно має вигляд зображений на рисунку 3.12.

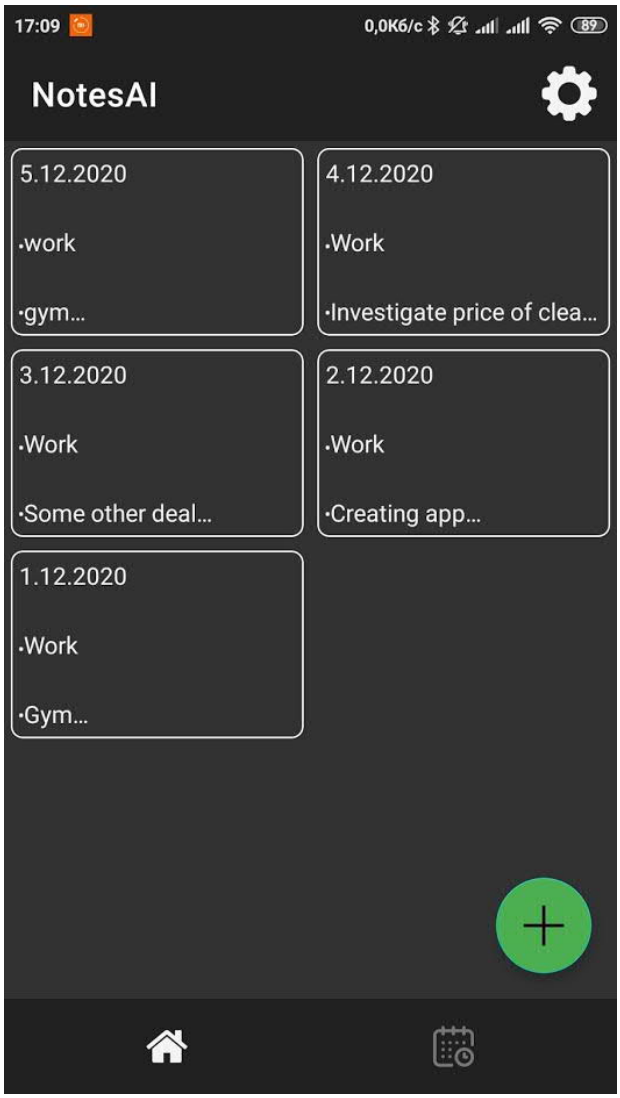


Рисунок 3.12 – головне вікно після заповнення бази даних тестовими даними

Для розмітки вікна з завданнями дня був використаний RecyclerView, що надає можливість при великому наповненні листа активностей прокручувати все наповнення вікна та не втрачати елементів.

З вікна активностей дня можна перейти до вікна активності, зображеного на рисунку 3.13.

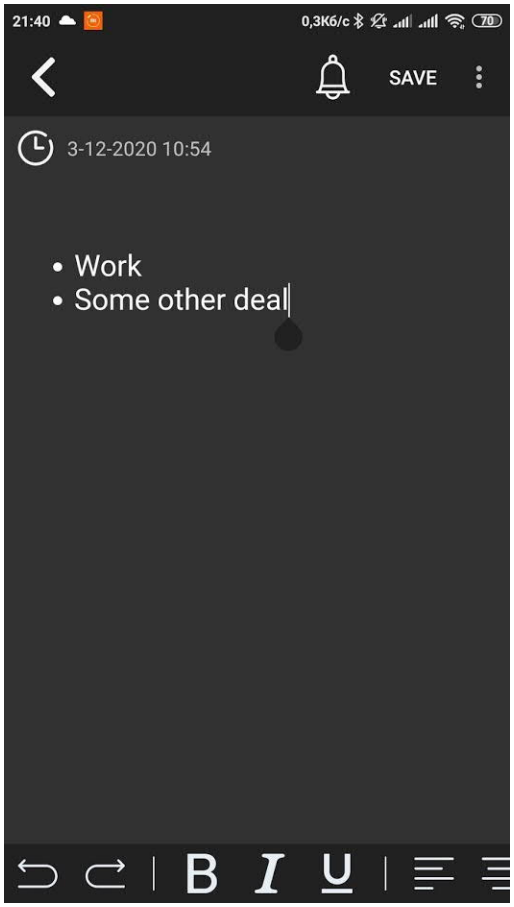


Рисунок 3.13 – Вікно завдань дня з визначеним часом нагадування

Після додавання активності додаток повертається до вікна зі списком завдань. Під час додавання завдання, додаючи час нагадування, система може запропонувати іншу годину, а навіть день виконання як зображено на рисунку 3.14. Також змінюється підказка системи про завантаженість користувача в цей день.

Система починає обраховувати оптимальний час виконання завдання після встановлення дати проведення, аналізуючи даний день та вибираючи оптимальний час. Якщо користувач не прийме рекомендацію та встановить

інший час буде перевірено цей проміжок часу на наявність інших завдань, якщо він буде зайнятим то система повідомить про це та запропонує інший час.

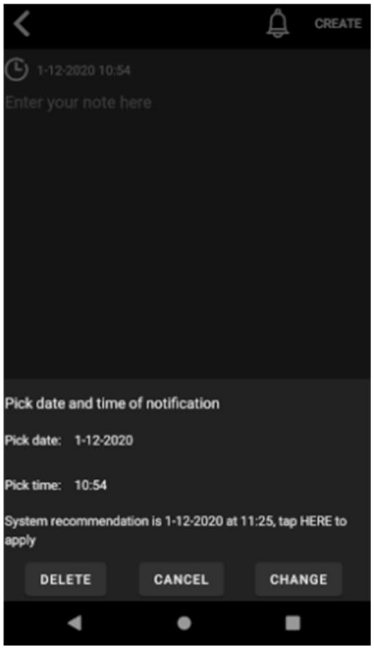


Рисунок 3.14 – рекомендація системи змінити час виконання завдання

Кожне завдання можна редагувати або видаляти у його вікні. Якщо під час редагування буде видалений текст завдання то зберегти буде неможливо та буде показано відповідне системне повідомлення за допомогою toast, як зображено на рисунку 3.15.

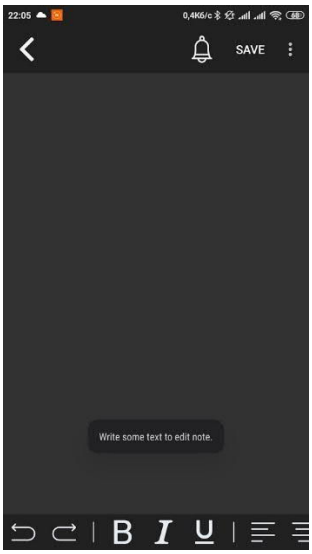


Рисунок 3.15 – повідомлення про неможливість зберегти пусте завдання

Якщо завдання виконувалось раніше, то система запропонує найчастіше використовуваний час початку завдання, якщо він вільний у найменш навантаженому дні.

Якщо користувач певну кількість разів не виконає всі заплановані завдання та додаток проаналізувавши схожі дні за допомогою методу k найближчих сусідів, буде розцінювати дні з подібним навантаженням як перевантажені, а не нормально навантажені, та відповідно рекомендуватиме меншу кількість завдань на такі дні.

Оскільки функціонал додатку буде покращуватись та розширюватись в майбутньому планується повна зміна дизайну, покращення алгоритмів та UX, а також інтеграція з гугл календарем, та збереження даних за допомогою гугл аккаунта в firebase, що буде в нагоді користувачу коли він змінить або загубить смартфон, оскільки всі його дані будуть збережені.

3.3 Публікація додатку

Для того щоб опублікувати додаток в Play Market, потрібний аккаунт розробника. Створити який можна за допомогою звичайного аккаунта Google. Для цього лише необхідно здійснити плату за ліцензію розробника в розмірі 25$, дія такої ліцензії триває двадцять п’ять років[25]. Для реєстрації потрібно вказати ім’я розробника яке буде бачити користувач на сторінці, додаткову електронну адресу на випадок втрати доступу до основної, та номер телефона розробника. Також розробник обов’язково має ознайомитись та прийняти договір розповсюдження додатків через Google Play та умови використання платформи. Вікно реєстрації облікового запису розробника зображено на рисунку 3.16.

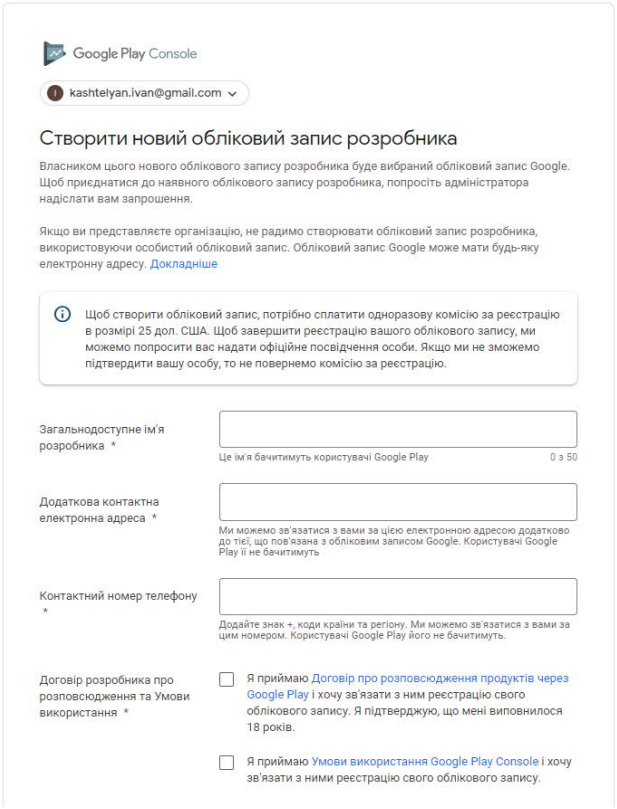


Рисунок 3.16 – Вікно реєстрації розробника

Публікація та інші додаткові настройки додатку здійснюються за допомогою Google play console, яке після реєстрації зустрічає розробника з короткою інструкцією та описом функціоналу зображено на рисунку 3.14

Рисунок 3.14 – Вікно з короткою відео інструкцією та описом функціоналу Google play console

Перед публікацією додатку обов’язково потрібно підтвердити свою особу, що зображено на рисунку 3.15. Для цього щоб підтвердити особу потрібно надати свої персональні дані такі як: ім’я, прізвище, місце проживання, поштовий індекс.

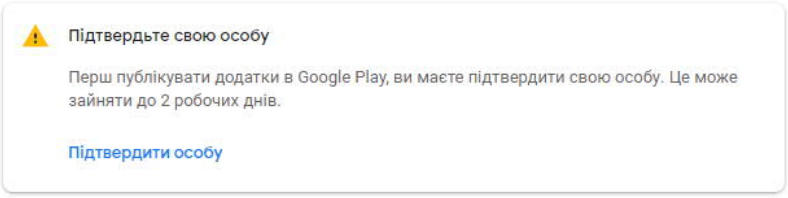


Рисунок 3.15 – Вікно з нагадуванням про обов’язкове підтвердження особи

Також розробнику потрібно вказати різноманітні дані які допоможуть визначити контент додатка такі як: доступ до додатка, реклама, вікові обмеження, цільова аудиторія, новинні додатки, категорія додатка та налаштування сторінки додатка. Початкове налаштування додатку зображено на рисунку 3.16.

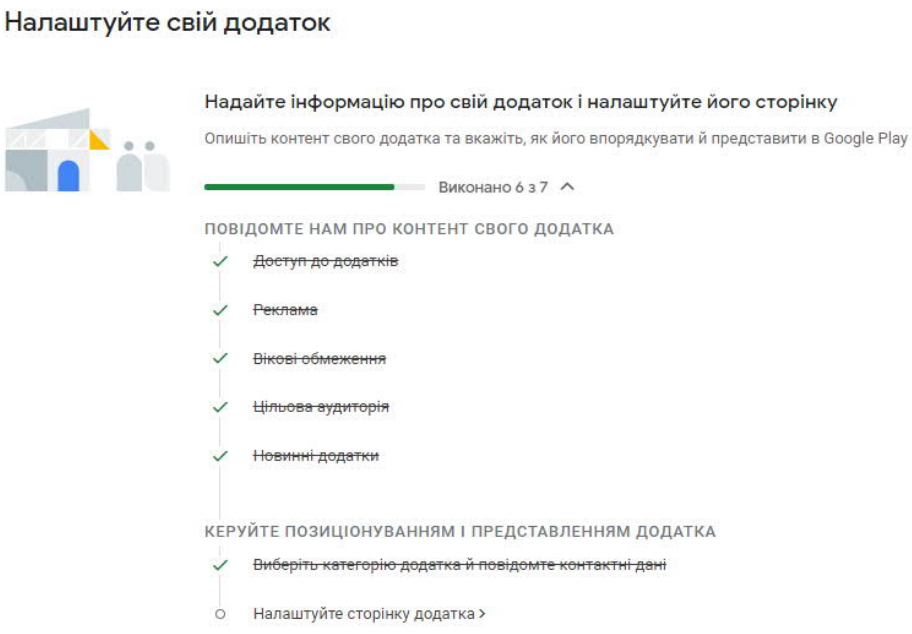


Рисунок 3.16 – початкове налаштування додатку в Play Console

Після завершення початкового налаштування розробник може приступити до публікації тестової або релізної версії додатку на вибір. Є декілька видів тестування додатку:

– з залученням більшості підконтрольних осіб тобто розробник контролює всіх тестувальників за допомогою Google груп або адрес електронної пошти;

– попередня реєстрація допоможе зрозуміти зацікавленість користувачів додатком, а коли додаток буде опубліковано їм буде надіслано сповіщення;

– відкрите тестування у якому може взяти участь будь-який користувач Google Play. Користувачі можуть залишати відгуки які не будуть впливати на загальнодоступну статистику.

Типи публікацій додатку та їх короткий опис зображено но рисунку 3.17.

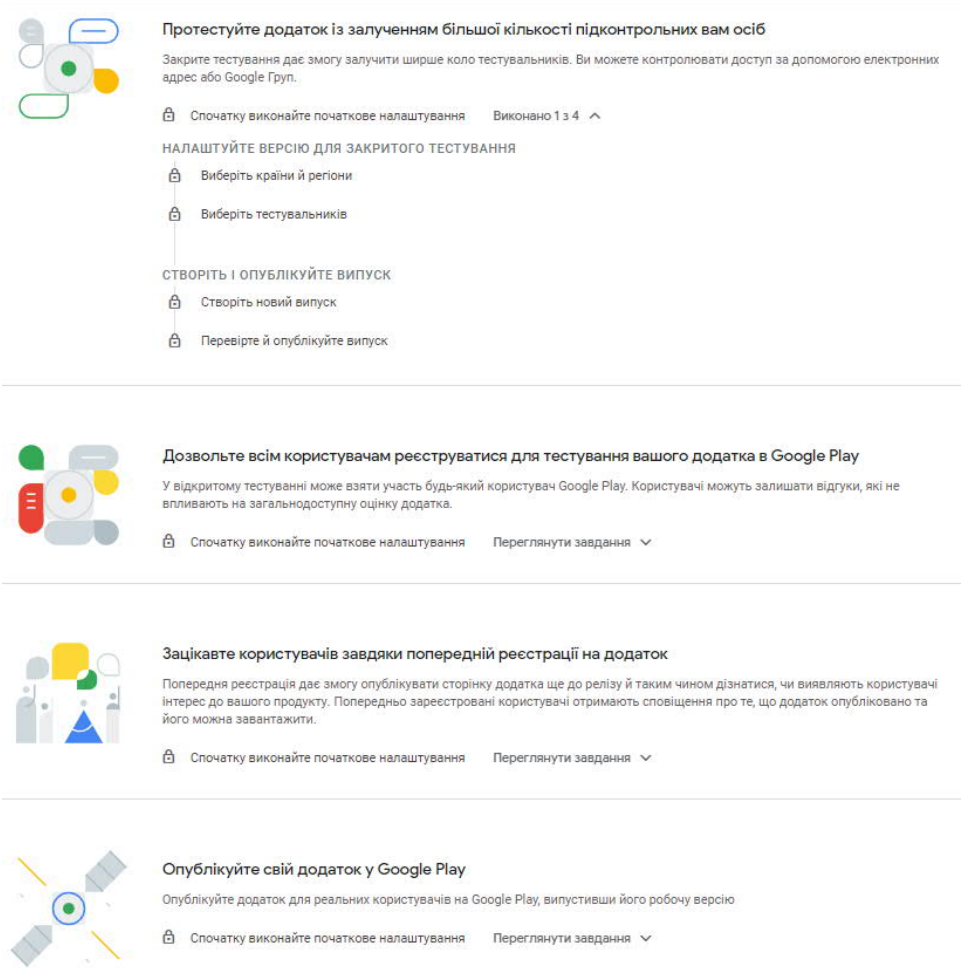


Рисунок 3.17 – типи публікацій додатку та їх короткий опис

Для створення випуску потрібно зробити білд проекту в форматі apk або aab, вказати назву випуску та примітки. Google рекомендує aab формат оскільки він є спеціально оптимізований для Play Market та має менший розмір.

Важливо підписати додаток спеціальним ключем розробника який генерується за допомогою спеціального розділу в IDE Android Studio оскільки Play не пропустить додаток для публікації як зображено на рисунку 3.18.

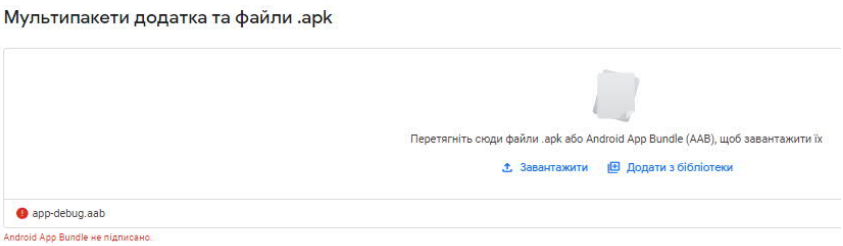


Рисунок 3.18 – повідомлення про те, що додаток не підписано

Для створення нового ключа потрібно вказати особисті дані розробника та країну проживання які він вказував під час реєстрації та зображено на рисунку 3.19.

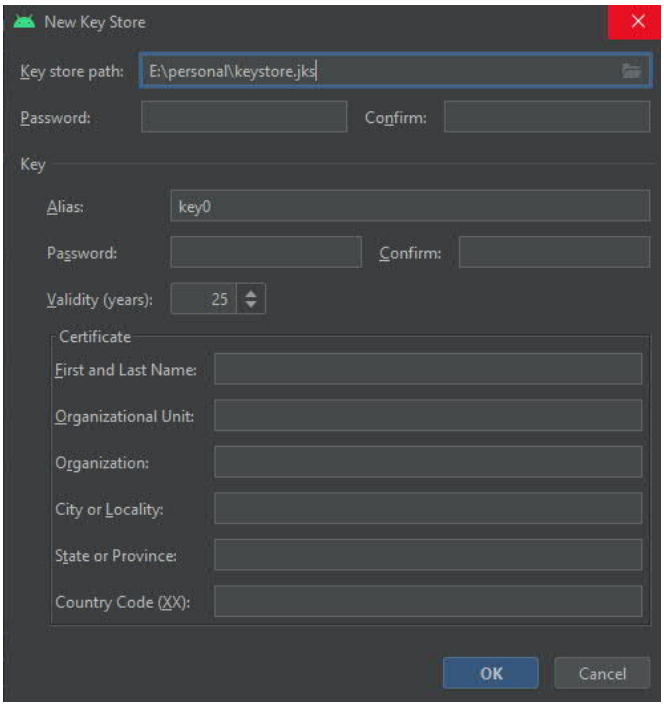


Рисунок 3.19 – дані які необхідно заповнити для створення ключа

Після заповнення даних вказаних на рисунку 3.19 можна створювати підписаний файл застосунку після додавання якого в Play Console можна побачити версію, мінімальний рівень арі та цільову авдиторію девайсів.

Після підтвердження всієї інформації можна почати випуск для обраної групи користувачів якщо це тестова версія, та згодом вона стане доступною для скачування у додатку Play Market на смартфоні.

3.4 Висновки до розділу

У даному розділі описано програмну розробку інформаційної системи планування задач. В ході розробки та проектування користувацького інтерфейсу програми було проаналізовано аналоги з метою знаходження оптимального способу реалізації функціоналу програми. Було реалізовано якомога найпростіший інтерфейс який є інтуїтивним та зрозумілим.

Було реалізовано спроектовану базу даних за допомогою бібліотеки яка спрощує роботу та створення бази даних RoomDatabase. Було описано роботу програмного продукту та спосіб взаємодії з користувачем через зворотній інтерфейс.

Було досліджено та описано реєстрацію та налаштування аккаунта розробника в Google Play після чого є можливість опублікувати додаток для широкого кола користувачів.

ВИСНОВКИ

Результати аналізу предметної області дослідження та існуючих аналогів дозволили обгрунтувати актуальність роботи, а також сформовати мету – розробка інформаційної моделі планування задач на основі алгоритмів машинного навчання.

Були проведені аналіз традиційних та сучасних аналогів інструментів планування часу, огляд існуючих досліджень на тему навантаження людини та аналіз діяльності людини за допомогою смартфона.

Android додаток було обрано як тип продукту системи, інструментом реалізації – Android Studio та мова програмування Kotlin.

Розроблена інтелектуальна система виконує спеціальні функції аналізу навантаження людини та планування завдання в оптимальний з точки зору навантаження час, а також виконує функції todo-list та нотаток.

В процесі розробки було отримано наступні результати:

1. На основі аналітичного підходу проведено порівняльний аналіз аналогів інформаційних систем планування задач на основі алгоритмів машинного навчання.

2. Проаналізовано методи підходів до розробки інтелектуальної системи машинного навчання.

3. Розроблено мобільний аплікейшин з використанням: MVVM, DI, Clean Architecture та мови програмування Kotlin.

4. Розроблено структуру бази даних для зберігання інформації користувача та для аналітичної роботи алгоритмів штучного інтелекту.

5. Представлено структуру додатку NotesAI. Було зареєстровано додаток розробника, що дозволить в подальшому публікувати дану систему в Play Market, для тестування, а після успішного тестування системи реальними користувачами отримувати певний дохід з реклами впровадженої в даний продукт.

Основною перевагою розробленого додатку у даній кваліфікаційній роботі є наявність інтелектуальної системи для визначення навантаження людини та рекомендацій обрати оптимальний час для виконання завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Каштелян І.В., Кулиняк Р.В. Використання систем штучного інтелекту для покращення засобів планування задач. III Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі». 26 листопада 2020, Тернопіль, Україна. Тернопіль: ЗУНУ, 2020. с.13

2. Кулиняк Р.В., Каштелян І.В. Алгоритм визнаачення «токсичності» повідомлення з використанням засобів NLP. III Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі». 26 листопада 2020, Тернопіль, Україна. Тернопіль: ЗУНУ, 2020. с.23

3. Керування часом. Веб-сайт. URL: https://uk.wikipedia.org/wiki/Керування_часом

4. Десять причин планировать свой день. Веб-сайт. URL: http://www.vitamarg.com/article/causes/7541-10-prichin-planirovat-svoj-den

5. Планування часу керівників і спеціалістів підприємств. Веб-сайт. URL: http://osvita.ua/vnz/reports/econom_pidpr/22220/

6. This New App Wants to Optimize Your Schedule by Running It Through an Algorithm. Веб-сайт. URL: https://www.entrepreneur.com/article/236094/

7. Meet Timeful, the intelligent 'Time Assistant'. Веб-сайт. URL: https://bgr.com/2014/07/31/timeful-time-assistant-app-for-iphone//

8. Refanidis, I., & Alexiadis, A. (2011). Deployment and evaluation of selfplanner, an automated individual task management system. Computational intelligence, 27(1), 41-59

9. Time's AI-powered time tracking app will help you stop procrastinating. Веб-сайт. URL: https://techcrunch.com/2017/01/09/times-ai-powered-time-tracking-appwill-help-you-stop-procrastinating/

10. 6 Artificial Intelligence Apps That Will Help You Reach Your Goals. Веб-сайт. URL: https://www.makeuseof.com/tag/artificial-intelligence-apps-reach-goals

11. Introducing Smart Schedule, a more intelligent way to plan your day. Веб-сайт. URL: https://doist.com/blog/todoist-smart-schedule/

12. Bayesian networks Веб-сайт. URL: https://habr.com/post/276355/

13. Soft Actor Critic—Deep Reinforcement Learning with Real-World Robots Веб-сайт. URL: https://bair.berkeley.edu/blog/2018/12/14/sac/

14. Cluster analysis Веб-сайт. URL: http://statlab.kubsu.ru/sites/project_bank/claster

15. Gradient descent Веб-сайт. URL: https://towardsdatascience.com/gradient-descent-simplyexplained-1d2baa65c757

16. H. Zhang (2004). The optimality of Naive Bayes. Proc. FLAIRS.

17. Дерево_ухвалення_рішень. Веб-сайт. URL: https://uk.wikipedia.org/wiki Дерево_ухвалення_рішень

18. Brett Lantz. Machine Learning with R. Pack Publishing. BirmonghamMumbai, 2013.

19. А.А. Бородинова , В.В. Мясников «Сравнение алгоритмов классификации в задаче распознавания объектов на радарных изображениях базы MSTAR», 2017

20. iOS vs Android Development: Which One is Best for Your App? Веб-сайт. URL: https://rubygarage.org/blog/ios-vs-android-development

21. Архітектура інформаційної системи. Веб-сайт. URL: https://elearning.sumdu.edu.ua/free_content/lectured:de1c9452f2a161439391120eef364dd 8ce4d8e5e/20151208095132/170352/index.html

22. ISO/IEC 2382:2015, Information technology — Vocabulary — Part 1

23. Концептуальная модель. Веб-сайт. URL: https://ru.wikipedia.org/wiki/Концептуальная_модель

24. Логічна модель даних. Веб-сайт. URL: https://uk.wikipedia.org/wiki/Логічна_модель_даних#Концептуальна,_логічна_та_фіз ична_модель_даних

25. Decorators. Веб-сайт. URL: https://github.com/prolificinteractive/materialcalendarview/wiki/Decorators

26. Material Calendar View. Веб-сайт. URL: https://github.com/prolificinteractive/material-calendarview

27. Инструкция по публикации Android-приложения в Google Play. Веб-сайт. URL: https://habr.com/ru/company/livetyping/blog/326874/

28. WPF Apps with the Model-View-ViewModel Design Pattern. Веб-сайт. URL: http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

29. Room. Основы. Веб-сайт. URL: https://startandroid.ru/ru/courses/architecture-components/27-course/architecture-components/529-urok-5-room-osnovy.html

30. LiveData. Веб-сайт. URL: https://startandroid.ru/ru/courses/architecture-components/27-course/architecture-components/525-urok-2-livedata.html

31. Dagger 2. Лечим зависимости по методике Google. Веб-сайт. URL: https://www.dataart.com.ua/news/dagger-2-lechim-zavisimosti-po-metodike-google/

32. Принцип інверсії залежностей. Веб-сайт. URL: https://uk.wikipedia.org/wiki/Принцип_інверсії_залежностей

33. Формула Харріса-Бенедикта. Веб-сайт. URL: https://www.calc.ru/Formula-Kharrisabenedikta.html

34. Т. Нуркевич, Б. Кристенсен. Реактивное программирование с использованием RxJava. ДМК Пресс, 2017. – С. 6-74.

35. Рейтинг языков программирования 2018: Go и TypeScript вошли в высшую лигу, Kotlin стоит воспринимать серьезно. Веб-сайт. URL: https://dou.ua/lenta/articles/language-rating-jan-2018

36. Why Kotlin language use is skyrocketing. Веб-сайт. URL: https://appdevelopermagazine.com/why-kotlin-language-use-isskyrocketing

37. Нейронный процессор. Веб-сайт. URL: https://ru.wikipedia.org/wiki/Нейронный_процессор

38. Не просто бот, который играет в Dota 2. Как OpenAI создает будующоё. Веб-сайт. URL: https://www.championat.com/cybersport/article-3738019-kak-rabotaet-bot-openai-five.html

39. S. Hijazi. Convolutional Neural Networks for Image Recognition. Веб-сайт. URL: https://ip.cadence.com/uploads/901/cnn_wp-pdf

40. Kiwiel, Krzysztof C. (2001). Convergence and efficiency of subgradient methods for quasiconvex minimization. Mathematical Programming (Series A) 90 (1) (Berlin, Heidelberg: Springer). с. 1–25.

41. Clean Architecture. Веб-сайт. URL: https://habr.com/ru/company/mobileup/blog/335382/

42. Газизова Э.Р. Аутентификация. Теория и практика обеспечения безопасного доступа к информационным ресурсам. Учебное пособие для вузов / Э.Р. Газизова, Л.Т. Веденьев, А. Афанасьев / За ред. Э.Р. Газизова. – СПб.:БХВ-Петербург, 2009 – 559 с.

43. Широковещательные приемники в Android. Веб-сайт. URL: https://docs.microsoft.com/ru-ru/xamarin/android/app-fundamentals/broadcast-receivers

44. Жизненный цикл приложения на Android. Веб-сайт. URL: http://developer.alexanderklimov.ru/android/theory/lifecycle.php

45. Логічна модель даних. Веб-сайт. URL: https://uk.wikipedia.org/wiki/Логічна_модель_даних#Концептуальна,_логічна_та_фізична_модель_даних

46. Advantages of using Android official tools. Веб-сайт. URL: http://androiddeveloper.galileo.edu/2017/09/25/advantages-using-android-official-tools/

47. SQLite. Веб-сайт. URL: https://ru.wikipedia.org/wiki/SQLite

48. Какую технику хранения данных Android использовать? Веб-сайт. URL: http://qaru.site/questions/180398/which-android-data-storage-technique-to-use

49. Березький О.М., Дубчак Л.О., Мельник Г.М. Методичні рекомендації до виконання кваліфікаційної роботи з освітнього ступеня “Магістр”. Спеціальність: 123 - Комп’ютерна інженерія. Магістерська програма -

Комп’ютерна інженерія"/ Під ред. О.М. Березького. Тернопіль:ЗУНУ,2020.32 с.

50. Гураль І.В., Дубчак Л.О. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп’ютерна інженерія»/Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 33 с.