Хмельницький національний університет
Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра телекомунікацій. медійних та інтелектуальних технологій

ДИПЛОМНА РОБОТА МАГІСТРА

Розробка мобільного додатку «Smart Clock»
Назва теми

Галузь знань 11 – Математика та статистика

Спеціальність 113 – Прикладна математика

Шифр ДРІІМ.2019/120.01.33.00

Виконав: студент 2 курсу, група ІІМм-19-1 Чеснюк М.В.
Підпис Ініціали, прізвище

Керівник Медзатий Д.М.
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри ТМІТ Підченко С.К.
Підпис, дата Ініціали, прізвище
10 12 2020 р.

Хмельницький, 2020

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем

Кафедра телекомунікацій, медійних та інтелектуальних технологій

Освітній рівень магістр

Галузь знань 11 Математика та статистика

Спеціальність 113 Прикладна математика

Освітня програма освітньо-професійна програма підготовки магістра

ЗАТВЕРДЖУЮ

Зав. кафедри д.т.н. доцент Підченко

Сергій Костянтинович

" 01 " 09 2020 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Чеснюка Максима Валерійовича

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Розробка мобільного додатку "Smart Clock"

Керівник проекту (роботи) к.т.н. доц. Медзатий Дмитро Миколайович

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.09.2020 р. № 118

2. Строк подання студентом проекту (роботи) на кафедру 01.12.2020 р.

3. Вихідні дані до проекту (роботи) Проведення порівняльного аналізу підходів до вирішення та реалізації додатків та алгоритмів визначення пробудження людини. Створити власний алгоритм визначення пробудження людини. Розробити додаток на основі якого і буде працювати алгоритм.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Аналіз моделей, методів та алгоритмів визначення пробудження людини. Дослідження різних методів та варіантів реалізації додатку. Розробка власного алгоритму визначення пробудження людини. Реалізація мобільного додатку.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Аналіз результатів порівняння різних алгоритмів. Модель роботи алгоритму. Реалізація алгоритму визначення пробудження людини. Реалізація мобільного додатку.

6. Консультанти розділів дипломного проекту (роботи)

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

7. Дата видачі завдання «_1_»_____вересня____2020р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1. Вибір напряму дослідження та узгодження тематики ДРМ з керівником	1.09.2020	
2. Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	3.09.2020	
3. Робота над розділом 1 – аналіз відомих моделей. методів за темою; постановка задачі	7.09.2020	
4. Робота над розділом 2 – розробка моделей і методів для вирішення поставленої задачі	14.09.2020	
5. Робота над тезами на наукову конференцію	21.09.2020	
6. Робота над розділом 3 – розробка алгоритмів та технологій, їх аналіз	5.10.2020	
7. Робота над розділом 3 – розробка мобільного додатку.	28.10.2020	
Узгодження отриманих; оформлення пояснювальної записки згідно вимог	8.11.2020	
Попередній захист ДРМ	10.11.2020	
Захист ДРМ на засіданні ЕК	5.12.2020	

Студент _____ Чеснюк М.В.
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Медзатий Д.М.
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема дипломної роботи: Розробка мобільного додатка «Smart Clock».
Автор роботи: Чеснюк Максим Валерійович
Керівник роботи: Медзатий Дмитро Миколайович
Загальний обсяг роботи: 120 сторінок, 56 рисунків, 3 додатки, 23 посилань.
АЛГОРИТМ ВИЗНАЧЕННЯ,РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ, РОЗРОБКА АЛГОРИТМУ, ОПТИМІЗАЦІЯ ДОДАТКУ

Метою роботи є розробка мобільного додатку для платформи Android, що базується на алгоритмі визначення пробудження людини.

Дана дипломна робота присвячена розробці програмного продукту для визначення у людини проблем зі сном.

Було розглянуто існуючі методи розробки алгоритмів, та створено унікальний алгоритм, що допоможе людям виявити проблеми зі сном та вчасно звернутись до лікаря.

Розроблено мобільний додаток «Smart Clock». Додаток-будильник, який аналізує протягом ночі, скільки разів людина прокидалась. Це відбувається завдяки унікальному алгоритму визначення пробудження людини. Це відбувається завдяки сенсорам вбудованим у будь який нині смартфон.

ANNOTATION

Thesis topic: Development of a mobile application "Smart Clock".
Author of the work: Chesniuk Maxym
Mentor: Dmitry Medzatiy
Total volume of work: 120 pages, 56 figures, 3 appendices, 23 references
DEFINITION ALGORITHM, MOBILE APPLICATION DEVELOPMENT, ALGORITHM DEVELOPMENT, APPLICATION OPTIMIZATION

The aim of the work is to develop a mobile application for the Android platform, based on the algorithm for determining human awakening.

This thesis is devoted to the development of a software product for determining a person's sleep problems.

Existing methods of developing algorithms were considered, and a unique algorithm was created to help people identify sleep problems and see a doctor in time.

The mobile application "Smart Clock" has been developed. An alarm clock application that analyzes how many times a person wakes up during the night. This is due to a unique algorithm for determining human awakening. This is due to the sensors built into any current smartphone.

2.12.2020
Дата/Date

Підпис студента/Signature

ЗМІСТ

С.

Вступ

У сучасному світі дуже мала кількість людей може повністю обходитись без смартфону. У смартфонах зараз зберігається величезна частина даних про людину. Усі користуються інтернетом, месенджерами такими як: Telegram, Viber, WhatsApp, грають в ігри, розважаються у соцмережах, роблять презентації та друкують різні документи. Смартфон повільно заміняє великі комп'ютери та ноутбуки. В середньому в день людина проводить 3-4 години в день користуючись смартфоном. Він завжди з нами, куди б ми не йшли. Щоб з ним не сталось нічого, люди докупляють аксесуари такі як: універсальні мобільні батареї для підзарядки смартфону у будь яку мить, чохли та бампери для захисту від падіння та пошкоджень, захисне скло та захисна плівка, щоб не було подряпин на екрані. Презентації нових моделей телефонів чекають з нетерпінням десятки мільйонів людей. Тільки на офіційному каналі Apple, що на відео платформі YouTube, презентацію нових моделей Apple iPhone 12 дивились 55 мільйонів людей. Так як презентація проходила і на офіційному сайті apple.com та в неофіційних джерелах, цю цифру сміливо можна множити на 2.

Метою дослідження є пошук методу реалізації алгоритму виявлення пробудження людини та розробка мобільного додатку-будильника "Smart Clock" на базі якого і буде працювати алгоритм.

Мобільний додаток - це прикладне програмне забезпечення, призначене для роботи на телефонах, планшетних комп'ютерах та інших мобільних пристроях». Додаток має сенс, якщо метою є взаємодіяти з користувачами в інтерактивному режимі або надати програму, яка вимагає роботи більше схожої на комп'ютерну програму, ніж веб-сайт. Додатки доступні на платформах розповсюдження в конкретних магазинах програм. Є як безкоштовні, так і платні додатки. Є декілька додатків, які спочатку доступні безкоштовно, але пізніше для отримання преміальних переваг потрібна мінімальна плата. "Потужне програмне забезпечення Apple iPhone, революційний користувальницький інтерфейс та потужна платформа для розробки спричинили майже нічний вибух програм. Найбільш широко

використовуваними смартфонами для мобільних додатків є Apple iPhone та телефони на ОС Android. Для платних додатків, як правило, відсотків 20-30%, надходить до постачальника дистрибуції, тобто так звані магазини додатків, а решта надходить до розробника програми. Згідно з мобільною статистикою, кількість програм, встановлених середнім одним користувачем телефону, дорівнює 26ти програм. Тож ця кількість чітко показує, що програми є доцільним, завдяки якому споживачі хочуть споживати вміст на мобільних телефонах.

Спочатку мобільні програми пропонувались для інформаційних цілей та для підвищення продуктивності, включаючи електронну пошту, календар, контакти, калькулятор та інформацію про погоду. Завдяки швидкому збільшенню технології та перспектив користувачів, розробники впроваджують їх в інші категорії, такі як мобільні ігри, GPS, банкінг, покупка квитків, соціальні медіа, відео-чати, автоматизація підприємств, послуги на основі місцезнаходження, фітнес-програми та нещодавно мобільні медичні програми.

Додаток може витягувати вміст та інформацію з Інтернету подібно до веб-сайту, але він також може завантажувати вміст, щоб він міг бути використаний пізніше за відсутності підключення до Інтернету, що є великою перевагою. Отже, програми, які не потребують з’єднання з Інтернетом, можна використовувати “де завгодно і коли завгодно”, тобто додаток можна використовувати в автономному режимі. Кілька недоліків популярності мобільних додатків постійно зростали, оскільки їх використання стає все більш поширеним серед користувачів мобільних телефонів. Це наочно видно з цифр, наведених, тобто загальний прогнозований обсяг завантажень мобільних додатків у 2019 році становить 2 трильйони, а загальний прогнозований дохід від мобільних додатків у 2019 році - 250 мільярдів доларів. із числа зростаючих з кожним роком розробників, з кожним роком зростає кількість додатків, доходи від додатків, кількість додатків, які з’являються на різних платформах, і найпопулярніші програми на різних платформах. Не так багато веб-сайтів / статей показують кількість видалених додатків, кількість користувачів, які видаляють програми, кількість хороших програм та поганих програм, дії, які спонукають користувачів видаляти програми, елементи, які

спричиняють неприємний досвід користування програмою. Існує обмежена кількість літератури про те, як з хороших додатків можна зробити чудові програми, і наскільки погані програми можна змінити, щоб стати хорошими та чудовими.

Для розробників мобільних застосунків на смартфони вкрай важливо знати характеристики споживачів. Процес розпізнавання цих характеристик зазвичай називають профілем користувача, який забезпечує основу знань для прийняття бізнес-рішень, забезпечує інтелектуальні послуги та забезпечує унікальну конкурентоспроможність. Як основний компонент профілю користувача, час сну може відображати спосіб життя, стан здоров'я та заняття людей. У цій роботі представлений гнучкий алгоритм під назвою ПЧС (Прогнозування часу сну), який призначений для прогнозування часу пробудження та сну за допомогою аналізу стану екрану смартфона. ПЧС спочатку збирає дані журналу стану екрану смартфона користувача та проводить попередню обробку з низкою допоміжних профілів користувача. Тоді, він виявляє та реєструє час пробудження та сну користувачів за один день шляхом пошуку та поєднання основних періодів гасіння екрану за останні 24 години. Нарешті, ПЧС передбачає майбутній час сну, узгоджуючи поточну послідовність стану екрана з усіма історичними записами. Застосовуючи ПЧС, більшість програм, що базуються на нічних та ранкових сценаріях, можуть надавати більш уважні послуги, а не дотримуватися фіксованого часу виконання, наприклад будильника. Експерименти з практичних додатків доводять, що ПЧС може ефективно прогнозувати час пробудження та час сну без застосування складних алгоритмів машинного навчання або завантаження даних на сервер. а не слідувати фіксованому часу виконання, як будильник. Експерименти з практичних застосувань доводять, що ПЧС може ефективно прогнозувати час пробудження та час сну без застосування складних алгоритмів машинного навчання або завантаження даних на сервер. а не слідувати фіксованому часу виконання, як будильник. Експерименти з практичних додатків доводять, що ПЧС може ефективно прогнозувати час пробудження та час сну без застосування складних алгоритмів машинного навчання або завантаження даних на сервер.

Задачами дослідження моєї дипломної магістерської роботи є:

- Виконати аналіз особливостей програмування для мобільних додатків;
- Виконати аналіз схожих рішень від інших науковців;
- Розробити модель поведінки алгоритму;
- Розробити алгоритм визначення пробудження людини;
- Розробити мобільний застосунок та інтеграція алгоритму в нього;

Об’єктом дослідження є актуальна на даний момент діяльність, в якій до уваги буде прийматись не лише основна функція мобільного застосунку, а й достовірна інформація чи будь-які дані, що можуть представляти інтерес для користувача.

Оскільки ця галузь передбачає роботу з застосунками, які працюють на пристроях Android (смартфони, планшети, вбудовані системи, тощо), для реалізації цього завдання було обрано мову програмування Kotlin, тому що вона є однією з офіційних мов програмування для ОС Android та мову Java.

1 Аналіз предметної області і теоретичні основи

1.1 Операційні системи та дизайн

Говорячи про «розробку додатків», ми зазвичай говоримо про розробку для мобільних пристроїв - включаючи смартфони та планшети.

Більш конкретно, ми зупинимось на мобільних додатках для двох найбільших та найважливіших мобільних операційних систем: iOS та Android.

У всьому світі мобільні пристрої більшості людей працюють на iOS та Android.

Станом на 2019 рік Android контролює близько 88% ринку мобільних пристроїв у всьому світі, а більшість решти належить Apple з їхніми пристроями iPhone та iPad. Кількість користувачів Android зросла з 1,8 мільярда пристроїв у вересні 2015 року до понад 2,7 мільярда сьогодні. Оскільки все більше і більше світового населення виходить в Інтернет протягом наступного десятиліття, Android буде продовжувати зростати.

iOS розробляється та підтримується Apple і використовується лише на власних пристроях, iPhone та iPad.

Іншими словами, у всесвіті Apple вони контролюють як апаратне, так і програмне забезпечення. Завдяки цьому вони можуть більш ретельно контролювати роботу своїх пристроїв (і мобільних додатків на своїх пристроях, завантажених із iOS App Store), що дозволяє їм підтримувати лояльну базу користувачів і солідну частку ринку.

Android розробляється і підтримується Google, часто вважається більш відкритою платформою в порівнянні з Apple.

Насправді Android - це операційна система з відкритим кодом, що означає, що ряд виробників пристроїв можуть використовувати Android на своїх пристроях.

Google продає кілька власних пристроїв, але багато користувачів Android використовують пристрої, створені іншими компаніями, такими як Samsung, Xiaomi, Huawei, LG, VIVO тощо.

На жаль програми, створені для iOS та Android, не взаємозамінні. Це означає, що власні програми iOS не будуть працювати на телефонах Android, і навпаки.

Незважаючи на те, що ви бачите, наприклад, Telegram або Instagram, які працюють на обох телефонах і виглядають дуже схожими, вони насправді були побудовані повністю окремо.

Велика кількість найпопулярніших програм, таких як Viber, YouTube, Twitter та багато інших, є повністю рідними.

Однією з найбільших проблем для мобільного дизайну є забезпечення очікуваних, прийнятних презентацій для широких цілей пристроїв. Вони можуть мати різне співвідношення сторін, роздільну здатність дисплея, пропускну здатність та різницю потужності. Такі постачальники, як Facebook, опублікували Facebook Lite на Android, який, наприклад, не включає функції відео. У багатьох випадках постачальник має версії програм для планшетів і телефонів. У наведеному нижче списку перші три елементи стосуються описів пристрою на рівні фізичної презентації. Сюди входять розмір, операційна система та орієнтація. На основі них відображається інший вигляд інтерфейсу. Останні два пункти стосуються міркувань щодо дизайну, які диференціюють випадки використання на користь мобільних пристроїв та офлайн.

Ці умови стосуються мобільного веб-дизайну та дизайну мобільних додатків:

Адаптовий дизайн : Адаптивний дизайн був представлений серйозно з появою мобільних пристроїв, використовуючи переваги бізнес-логіки, розміру та орієнтації CSS-медіа-запитів. Його намір полягає у реагуванні на різні статичні та мінливі особливості презентації програми.

Адаптивний дизайн : Різниця між адаптивним та адаптовим дизайном, як правило, відноситься до фіксованих, дискретних вимірювань першого порівняно із типовим розміром рідини та розміщенням другого. В адаптивному дизайні ви бачите більш точні розміри для кількох роздільних здатностей дисплея.

Прогресивний дизайн : Прогресивний дизайн, такий як у прогресивних веб-програмах, відноситься до програм або сайтів, які можуть швидко завантажуватися і мають рівень автентифікації, доступний для таких завдань, як заповнення полів форми або налаштування відображення карти.

Перш за все для мобільних пристроїв : По- перше для мобільних пристроїв, керівництво розробників розробляє традиційний веб-вміст таким чином, щоб менші екрани телефонів та інших пристроїв мали перше врахування та найвищі якості перегляду, коли це можливо.

По-перше в автономному режимі. Спочатку в автономному режимі мається на увазі розробка програми, яка здатна функціонувати, коли немає з’єднання з мережею. Це може включати такі функції, як завантаження початкового вмісту в програму, використання локальної пам’яті для доступу та збереження даних та уникнення перебоїв у роботі користувача.

Незважаючи на те, що ці терміни можуть здаватися домашніми, якщо говорити про веб-сайти та сторінки, застосування тих самих стильових методів до мобільних пристроїв, як правило, вимагає дещо більш точного налаштування, ніж їх традиційні аналоги. Як Android, так і iOS забезпечують ряд різних роздільних здатностей та розмірів піктограм, екранів та шрифтів. Навіть при використанні масштабованого вмісту, будь то SVG чи динамічно розрахований розмір і розміщення, точність, необхідна для кожного з конкретних пристроїв для кожної платформи, може бути страшною. На щастя, більшість інструментів сьогодні виконують за вас велику частину роботи. Це гарна ідея знати конкретні пристрої та платформи, які ви маєте намір підтримувати. Зокрема, ви повинні знати мінімальні вимоги та точки обмеження розміру, які вимагають різних презентацій програми.

Тому мій вибір при розробці програми впав на операційну систему Android. Так як це найросповсюдженіша в світі операційна система. Також Android має переваги у вигляді підтримки найпопулярнішої об’єктно-орієнтованої мови програмування Java та власної мови програмування Kotlin. Та про сон також потрібно не забувати, бо саме з ним зв’язаний додаток.

1.2 Розвиток мобільної операційної системи

Історія Android починається в жовтні 2003 року - задовго до того, як термін смартфон широко використовувався, і за кілька років до того, як Apple оголосила про свої перші iPhone та iOS. Компанія Android Inc була заснована в Пало-Альто, штат Каліфорнія. Його чотирма засновниками були Річ Майнер, Нік Сірс, Кріс Уайт та Енді Рубін. Тоді Рубін цитував слова, що Android Inc збирається розробляти "розумніші мобільні пристрої, які більше знають про місцезнаходження та уподобання власника".

У виступі в Токіо 2013 року Рубін показав, що ОС Android спочатку мала на меті вдосконалити операційні системи цифрових камер. Але навіть тоді ринок автономних цифрових камер занепадав. Всього через кілька місяців Android Inc вирішила змінити швидкість на використання ОС всередині мобільних телефонів.

У 2007 році Apple випустила перший iPhone і відкрила нову еру в мобільних обчисленнях. У той час Google все ще працював над Android в секреті, але в листопаді того ж року компанія повільно почала розкривати свої плани щодо конкуренції з Apple та іншими мобільними платформами. У значній мірі розвиток Google очолив формування так званого Альянсу відкритих слухавок. До нього увійшли виробники телефонів, такі як HTC та Motorola, виробники чіпів, такі як Qualcomm і Texas Instruments, та оператори, включаючи T-Mobile.

Тоді голова та генеральний директор Google Ерік Шмідт сказав: "Сьогоднішнє оголошення є більш амбітним, ніж будь-який окремий" Google Phone", про який спекулювала преса протягом останніх кількох тижнів. Ми бачимо, що потужна платформа, яку ми представляємо, буде працювати на тисячі різних моделей телефонів ".

Публічна бета-версія Android версії 1.0 стартувала для розробників 5 листопада 2007 року.

У вересні 2008 року було оголошено про перший смартфон на базі Android - T-Mobile G1, який в інших частинах світу також називають HTC Dream. Він надійшов у продаж у США в жовтні того ж року. Телефон із його спливаючим 3,2-дюймовим сенсорним екраном у поєднанні з фізичною клавіатурою QWERTY не був зовсім чудом дизайну. Дійсно, T-Mobile G1 отримав досить погані відгуки від технічних ЗМІ. У пристрої навіть не було стандартного 3,5-міліметрового роз'єму для навушників, що, на відміну від сьогоднішнього дня, фактично було де-факто функцією телефону серед конкурентів Android.

Однак ОС Android 1.0 всередині вже мала торговельні марки плану Google для ОС. Він інтегрував низку інших продуктів та послуг компанії, включаючи Карти Google, YouTube та браузер HTML (до Chrome), який, звичайно, використовував пошукові служби Google. У ньому також була перша версія Android Market, магазин програм, про який Google з гордістю заявив, що має «десятки унікальних, перших у своєму роді додатків для Android». Зараз усі ці функції звучать досить примітивно, але це був лише початок зростання Android на ринку мобільних пристроїв.

Перше офіційне загальнодоступне кодове ім’я для Android з’явилося лише до випуску версії 1.5 Cupcake у квітні 2009 року (Рисунок 1.1). За присвоєння назв версій Android іменем солодких цукерок та десертів традиційно відповідав менеджер проектів у Google Райан Гібсон. Однак його конкретні причини використання такої конвенції про найменування залишаються невідомими.

Рисунок 1.1 – Скульптура присвячена Android Cupcake в офісі Google.

Cupcake додав чимало нових функцій та вдосконалень порівняно з першими двома загальнодоступними версіями. Сюди входять речі, які ми зараз сприймаємо як належне, такі як можливість завантажувати відео на YouTube, спосіб автоматичного обертання дисплеїв телефонів та підтримка сторонніх клавіатур.

У вересні 2009 року Google швидко запустив Android 1.6 Donut (Рисунок 1.2). Нова ОС тепер пропонувала підтримку операторів, що використовують мережі на базі CDMA. Це дозволило продавати телефони Android всім перевізникам по всьому світу.

Рисунок 1.2 – Скульптура присвячена Android Donut в офісі Google.

Інші функції включали введення вікна швидкого пошуку та швидке перемикання між камерою, відеокамерою та галереєю для спрощення захоплення медіа. Donut також представив віджет Power Control для управління Wi-Fi, Bluetooth, GPS тощо.

Одним із телефонів, що продавались із встановленою «Пончиком», був Dell Streak. Він мав величезний (на той час) 5-дюймовий екран і на нашому власному сайті був описаний як "смартфон / планшет". У наші дні 5-дюймовий дисплей вважається порівняно невеликим для смартфона.

У жовтні 2009 року - приблизно через рік після запуску Android 1.0 - Google випустив версію 2.0 ОС з офіційною кодовою назвою Éclair (Рисунок 1.3). Ця версія першою додала підтримку перетворення тексту в мову, а також представила живі шпалери, підтримку кількох облікових записів та навігацію Google Maps, серед багатьох інших нових функцій та вдосконалень.

Рисунок 1.3 – Скульптура присвячена Android Éclair в офісі Google.

Motorola Droid був першим телефоном, який не підтримував Android 2.0. Droid також був першим телефоном на базі Android, який був проданий Verizon Wireless. У кумедних дрібницях, хоча Google безпечно використовував Android як назву для своєї ОС, термін «Droid» був зареєстрований торговою маркою Lucasfilm, посилаючись на роботів франшизи «Зоряні війни». Motorola повинна була отримати дозвіл і заплатити гроші Lucasfilm, щоб використати ім'я для свого телефону. Motorola продовжувала використовувати бренд Droid для багатьох своїх телефонів ще в 2016 році.

Android 2.2 Froyo (скорочення від «Frozen yogurt») був офіційно запущений у травні 2010 року (Рисунок 1.4). Смартфони зі спортивним Froyo могли скористатися перевагами кількох нових функцій, включаючи функції мобільної точки доступу Wi-Fi, надсилання сповіщень через послугу Android Cloud to Device Messaging (C2DM). , підтримка спалаху тощо.

Рисунок 1.4 – Скульптура присвячена Android Froyo в офісі Google.

Перший смартфон, який використовував фірмову марку Google Nexus - Nexus One - був випущений з Android 2.1 готовим раніше в 2010 році, але пізніше того ж року він швидко отримав ефірне оновлення для Froyo. Це ознаменувало новий підхід для Google, оскільки компанія співпрацювала, як ніколи раніше, з виробником обладнання HTC, щоб продемонструвати чистий Android.

Android 2.3 Gingerbread (Рисунок 1.5) був випущений у вересні 2010 р. ОС отримала оновлення користувальницького інтерфейсу під програмою Gingerbread. Це додало підтримку використання функцій ближнього зв'язку (NFC) для смартфонів з необхідним обладнанням. Першим телефоном, який використовував як пряники, так і обладнання NFC, був Nexus S, який був спільно розроблений Google і Samsung. Пряники також заклали основу для селфі, додавши підтримку декількох камер та підтримку відеочату в Google Talk.

Рисунок 1.5 – Скульптура присвячена Android Gingerbread в офісі Google.

Android 3.0 Honeycomb вийшов дивним (Рисунок 1.6). Стільник створений для планшетів та інших мобільних пристроїв із великими дисплеями. Вперше він був представлений у лютому 2011 року разом із планшетом Motorola Xoom. Він включав такі функції, як перероблений інтерфейс для великих екранів, а також панель сповіщень, розміщену внизу дисплея планшета.

Ідея полягала в тому, що Honeycomb буде пропонувати функції, з якими не змогли б впоратися менші дисплеї, знайдені на той час на смартфонах. Це також була реакція Google та її сторонніх партнерів на випуск iPad від Apple у 2010 році. Незважаючи на те, що Honeycomb був доступний, деякі планшети все-таки випускались із версіями Android 2.x на базі смартфонів. Зрештою, Honeycomb виявився версією Android, яка не побачила широкого поширення. Google вирішив інтегрувати більшість своїх функцій у свою наступну велику версію 4.0 - Ice Cream Sandwich. Це трохи відхилено в історії Android.

Рисунок 1.6 – Скульптура присвячена Android Honeycomb в офісі Google.

Випущена в жовтні 2011 року версія Ice Cream Sandwich для Android принесла ряд нових функцій (Рисунок 1.7). У ньому поєднано багато варіантів стільникової версії для планшетів та пряників, орієнтованих на смартфон. На головному екрані він також містив "лоток для улюблених", а також першу підтримку для розблокування телефону за допомогою камери для фотографування обличчя власника. З того часу така біометрична підтримка для входу значно еволюціонувала та вдосконалювалась.

Інші помітні зміни в ICS включали підтримку всіх екранних кнопок, жестикулювання пальцем, щоб відхилити сповіщення та вкладки браузера, а також можливість контролювати використання даних через мобільний телефон та Wi-Fi.

Рисунок 1.7 – Скульптура присвячена Android Ice Cream Sandwich в офісі Google.

Ера Jelly Bean для Android розпочалася в червні 2012 року випуском Android 4.1 (Рисунок 1.8). Google швидко випустив версії 4.2 та 4.3 - обидві під маркою Jelly Bean - відповідно у жовтні 2012 року та липні 2013 року.

Деякі нові доповнення до цих оновлень програмного забезпечення включали нові функції сповіщень, які відображали більше вмісту або кнопок дій, а також повну підтримку версії Android веб-браузера Chrome від Google, яка була включена в Android 4.2. Google Now також з’явився як частина пошуку, тоді як “Project Butter” був представлений для прискорення анімації та покращення чутливості до дотику Android. Зовнішні дисплеї та Miracast також отримали підтримку, як і HDR-фотографія.

Рисунок 1.8 – Скульптура присвячена Android Jelly Bean в офісі Google.

Android 4.4 - це перша версія ОС, яка фактично використовувала торговельну марку цукерки. Перед офіційним запуском у вересні 2013 року компанія випустила підказки на своїй конференції Google I/O того року, що кодова назва для Android 4.4 насправді буде «Key Lime Pie». Дійсно, більшість команд Google з Android вважали, що це теж має бути.

Як виявилось, директор Google з питань глобальних партнерських відносин Android Джон Лагерлінг вважав, що "Key Lime Pie" не буде досить відомим ім'ям для використання у всьому світі. Натомість він вирішив зробити щось інше. Він зв’язався з Nestle, творцями батончика KitKat, і запитав їх, чи можна використовувати ім’я для Android 4.4. Nestle погодилася, і KitKat став назвою наступної версії Android (Рисунок 1.9). Це був експеримент у маркетингу, який Google не відновив до запуску Oreo (ми до цього дійдемо).

KitKat не мав величезної кількості нових функцій, але мав одну річ, яка справді допомогла розширити загальний ринок Android. Він був оптимізований для роботи на смартфонах, які мали лише 512 МБ оперативної пам'яті. Це дозволило виробникам телефонів використовувати останню версію Android на значно дешевших телефонах. Смартфон Google Nexus 5 був першим із попередньо встановленою Android 4.4.

Рисунок 1.9 – Скульптура присвячена Android KitKat в офісі Google.

Вперше запущений восени 2014 року, Android 5.0 Lollipop став серйозним зрушенням в загальному вигляді операційної системи (Рисунок 1.10). Це була перша версія ОС, яка використовувала нову мову Google Design Material. У ньому було використано ліхтарі та тіньові ефекти, серед іншого, для імітації паперового вигляду інтерфейсу користувача Android. Інтерфейс користувача також отримав

деякі інші оновлення, включаючи оновлену панель навігації, розширені сповіщення про екран блокування та багато іншого.

Подальше оновлення Android 5.1 внесло ще кілька змін. Сюди входила офіційна підтримка дзвінків із двома SIM-картами, голосових дзвінків HD та захисту пристрою, щоб злодії не мали доступу до телефону навіть після скидання заводських налаштувань.

Смартфон Google Nexus 6 разом із планшетом Nexus 9 стали першими пристроями, на яких попередньо встановлено Lollipop.

Рисунок 1.10 – Скульптура присвячена Android Lollipop в офісі Google.

Випущений восени 2015 року, Android 6.0 Marshmallow використовував солодкі ласощі, яке подобається кемперам, як головний символ. Внутрішньо Google використовував “Горіхове печиво Macadamia” для Android 6.0 до офіційного оголошення Marshmallow (Рисунок 1.11). Він включав такі функції, як нова вертикально прокручувана скринька додатків, поряд із Google Now on Tap, вбудована підтримка біометричного розблокування відбитків пальців, підтримка USB Type-C, введення Android Pay (тепер Google Pay) та багато іншого.

Першими пристроями, які постачались із попередньо встановленою програмою Marshmallow, були смартфони Google Nexus 6P та Nexus 5X разом із планшетом Pixel C.

Рисунок 1.11 – Скульптура присвячена Android Marshmallow в офісі Google.

Версія 7.0 мобільної операційної системи Google, запущена восени 2016 року. До того, як Нуга була розкрита, Google Android внутрішньо називав "Нью-

Йоркський чізкейк". Багато нових функцій Нуги включали кращі багатозадачні функції для зростаючої кількості смартфонів із більшими дисплеями, такими як режим розділеного екрану, а також швидке перемикання між програмами.

Google також зробив ряд великих змін за лаштунками. Він перейшов на новий компілятор JIT, щоб пришвидшити роботу програм, підтримав API Vulkan для швидшого 3D-рендерінгу та дозволив OEM-виробникам підтримувати свою неіснуючу платформу Daydream VR.

Google також використав цей реліз, щоб сміливо проштовхнутися на ринок преміум-смартфонів. Власні компанії Pixel та Pixel XL, а також LG V20 були першими, хто вийшов з попередньо встановленою Nougat (Рисунок 1.12).

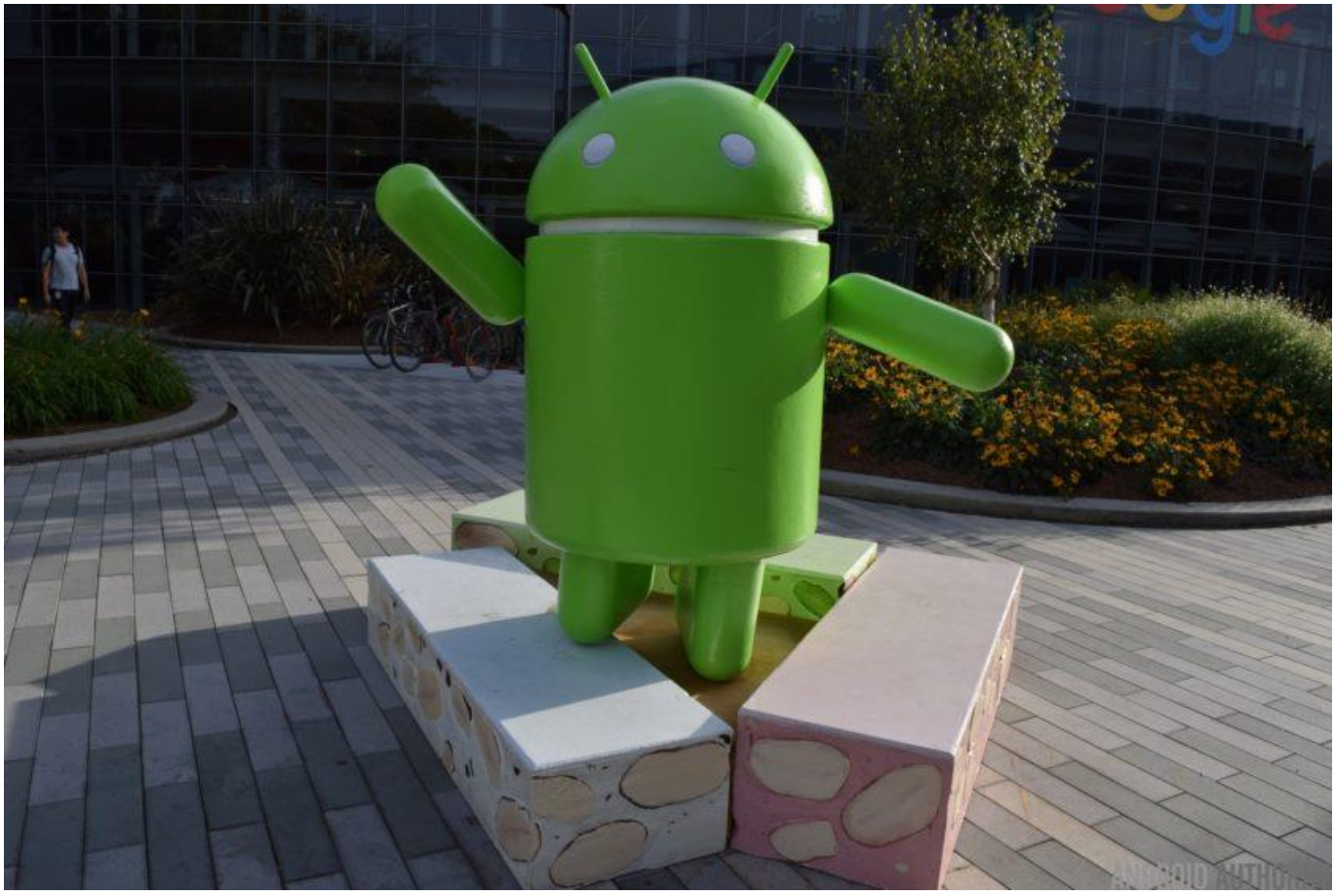


Рисунок 1.12 – Скульптура присвячена Android Nougat в офісі Google.

У березні 2017 року Google офіційно оголосив і випустив перший попередню бету для розробників для Android O, також відомий як Android 8.0. До цього випуску Хіросі Локхаймер, старший віце-президент Android у Google, опублікував у Twitter GIF торта Oreo - перший надійний натяк на те, що популярне печиво Oreo справді буде офіційною кодовою назвою для Android 8.0.

У серпні Google підтвердив загальнодоступну назву для Android 8.0, яка надихнута файлами cookie. Це був другий раз, коли компанія обрала торгову марку для Android (Oreo належить Nabisco). Відірвавшись від традицій, Google вперше показав статую талісмана Android Oreo на прес-заході в Нью-Йорку, а не в штаб-квартирі Googleplex (Рисунок 1.13). Статуя зображує талісман Android як літаючого супергероя в комплекті з мисом. Пізніше того ж дня в головній штаб-квартирі Google була встановлена друга статуя.

Що стосується функцій, Android Oreo забезпечив безліч візуальних змін у меню Налаштування, а також вбудовану підтримку режиму "картинка в картинці", канали сповіщень, нові API автозаповнення для кращого управління паролями та даними заповнення та багато іншого. Android Oreo вперше був встановлений на власних телефонах Google Pixel 2.

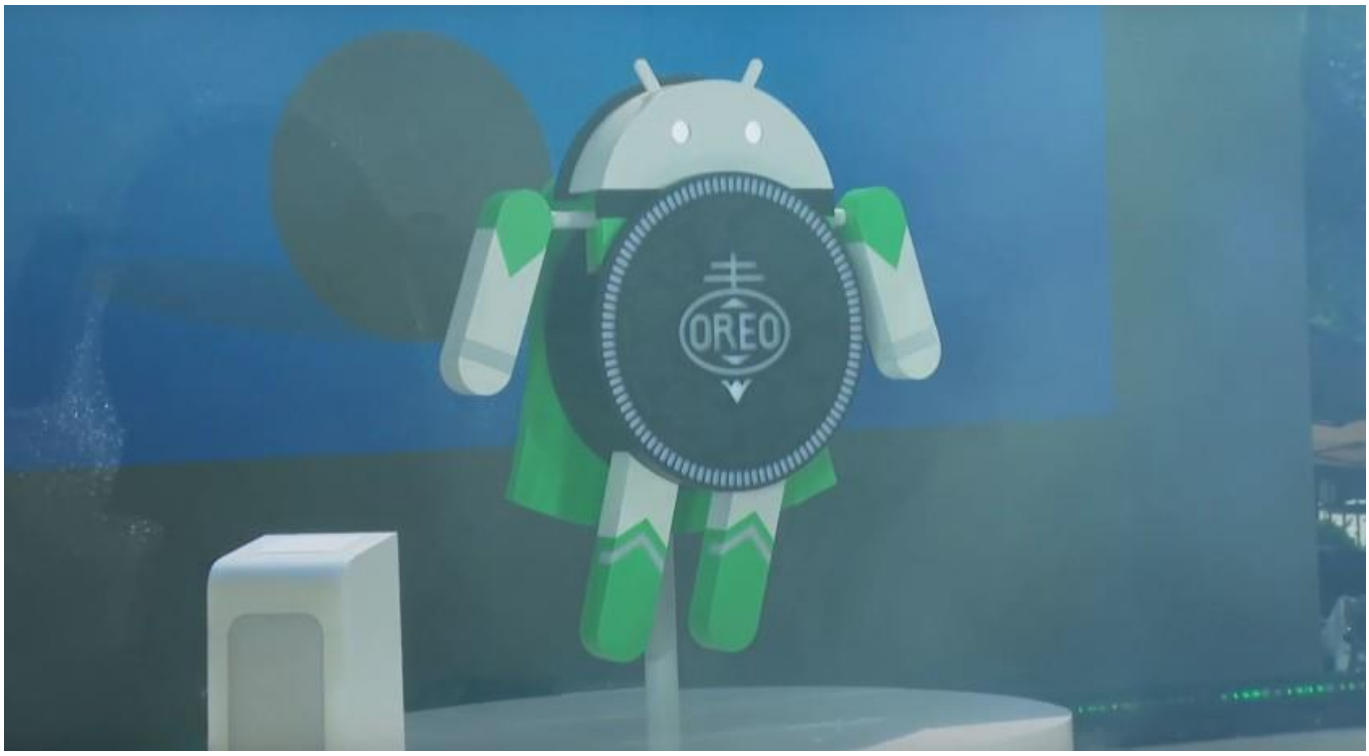


Рисунок 1.13 – Скульптура присвячена Android Oreo в офісі Google.

Google запустив перший попередню бету для розробників наступного великого оновлення Android, Android 9.0 P, 7 березня 2018 року. 6 серпня 2018 року компанія офіційно запустила остаточну версію Android 9.0, надавши їй офіційну кодову назву «Pie» (Рисунок 1.14).

Android 9.0 Pie включав ряд основних нових функцій та змін. Одна з них відмовилася від традиційних навігаційних кнопок на користь однієї подовженої кнопки в центрі, яка стала новою кнопкою додому. Проводячи пальцем угору, відкриється Огляд із найновішими програмами, рядком пошуку та п’ятьма пропозиціями додатків внизу. Ви можете провести пальцем ліворуч, щоб переглянути всі нещодавно відкриті програми, або перетягнути кнопку головного вправо, щоб швидко прокрутити програми.

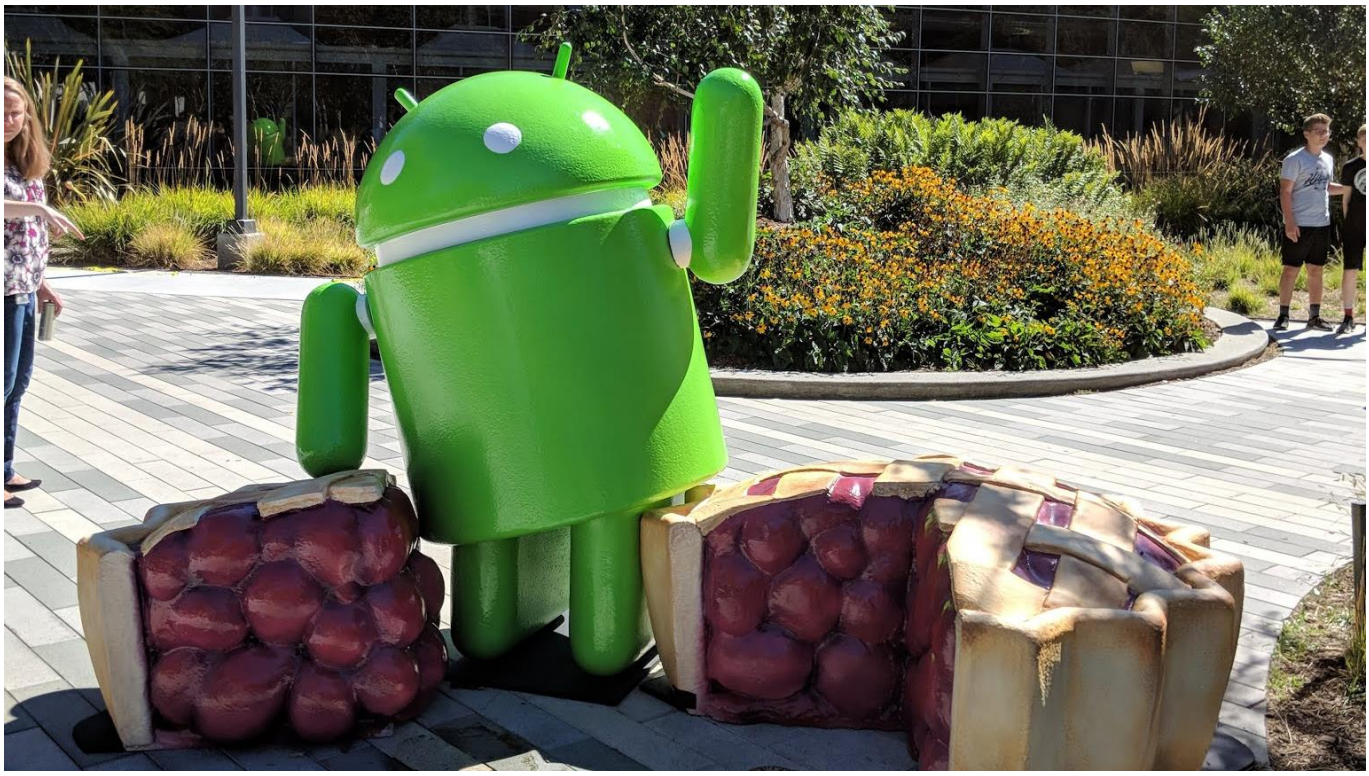


Рисунок 1.14 – Скульптура присвячена Android Pie в офісі Google.

Android 9.0 Pie також включав деякі нові функції, покликані допомогти продовжити час автономної роботи вашого смартфона. Цього вдалося досягти завдяки використанню машинного навчання на пристрої, яке передбачає, якими програмами ви будете користуватися зараз, а якими додатками не користуватиметесь пізніше. Pie також має функцію Shush, яка автоматично переводить ваш телефон у режим "Не турбувати", коли ви перекидаєте екран телефону на рівну поверхню. Також є Slices, який надає меншу версію встановленого додатка в Пошуку Google, пропонуючи певні функції додатка без відкриття повної програми.

Як завжди, Android 9.0 Pie був вперше офіційно доступний для телефонів Google Pixel, але він також вийшов на Essential Phone одночасно.

Через 10 років після запуску ОС ми отримали ще одну важливу віху історії Android. 13 березня 2019 року Google запустив перший офіційний попередній перегляд Android Q для розробників. 22 серпня 2019 року Google оголосив про

основне оновлення бренду Android (Рисунок 1.15). Це включало новий логотип і, що більш важливо, рішення відмовитись від традиційної назви десерту для наступної версії. Як результат, Android Q офіційно відомий просто як Android 10. Він був офіційно запущений 3 вересня 2019 року для пристроїв Pixel від Google.

Рисунок 1.15 – Логотип Android Q 10.0.

Як завжди з будь-яким новим випуском Android, Android 10 мав ряд нових функцій та вдосконалень, а також ряд нових API. Сюди входила підтримка прискорення складних телефонів. Android 10 також представив загальносистемний темний режим, а також нові елементи керування жестами, ефективніше меню спільного доступу, функції розумної відповіді для всіх програм обміну повідомленнями та більше контролю над дозволами на основі додатків.

18 лютого Google запустив перший попередній перегляд для розробників Android 11. Після випуску ще декількох публічних бета-версій остаточна версія Android 11 була запущена 8 вересня 2020 року (Рисунок 1.16).

Android 11 надійшов з великою кількістю нових функцій. Це включає нову категорію сповіщень про розмови, де всі ваші чати з різних програм збираються в одному місці. Ви також можете зберегти кожне сповіщення, яке з'явилося на вашому телефоні за останні 24 години. Зовсім нова функція дозволяє записувати екран телефону разом із звуком, не потребуючи стороннього додатка. Також є новий розділ Android 11, присвячений керуванню розумними домашніми пристроями.

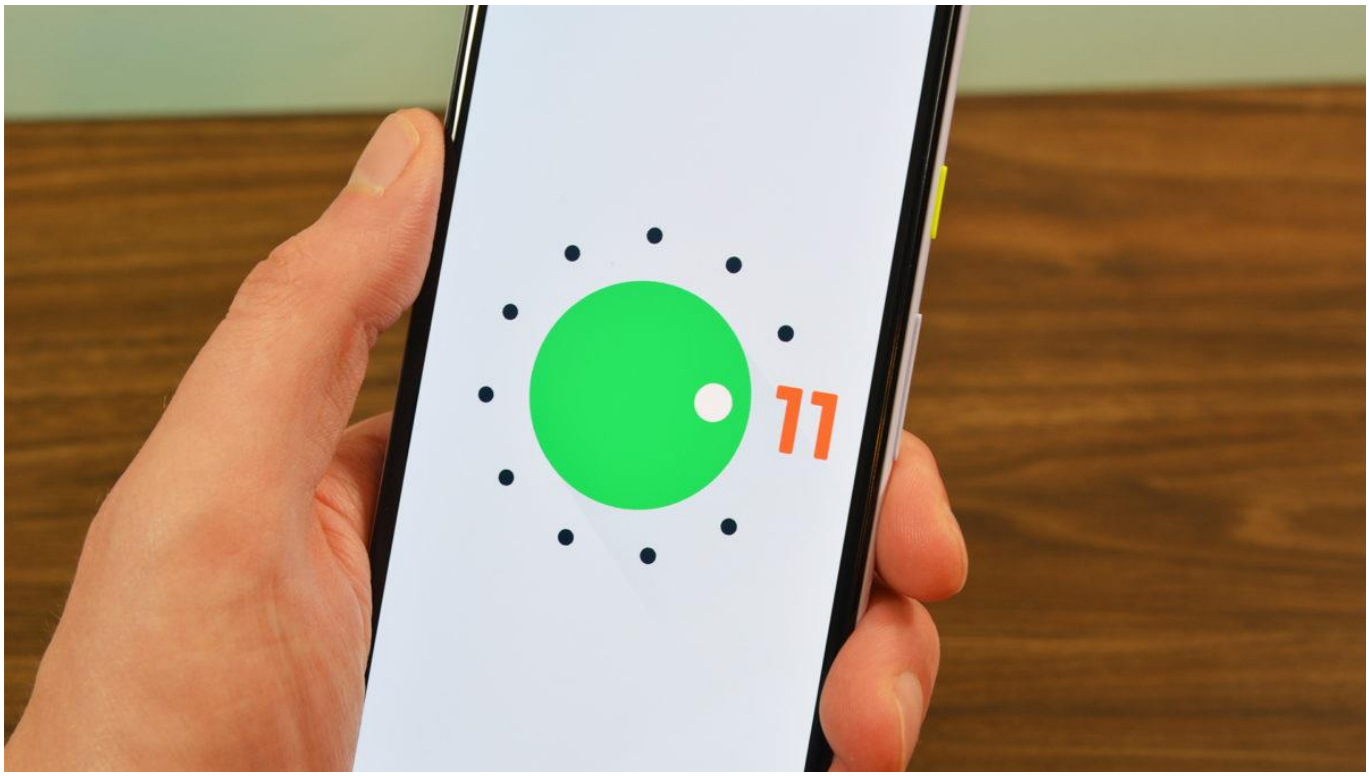


Рисунок 1.16 - Логотип Android 11.

Однак телефони Pixel отримують ексклюзивну функцію Android 11. Він використовує ШІ та машинне навчання, щоб контролювати, які програми з'являються на док-станції вашого телефону.

Google встановив свою традиційну статую на святкування запуску Android 11, але також випустив AR-версію статуї для всіх телефонів Android ARCore. У ньому є навіть пара писанок, включаючи рецепт приготування червоного оксамитового торта. Це також є внутрішньою кодовою назвою ОС в Google.

Android пройшов довгий шлях з часів свого скромного початку як продукт невеликого запуску. На сьогоднішній день це провідна мобільна операційна система у світі, де частка ринку становить близько 75%.

Компанія Mountain View все ще надзвичайно віддана подальшому розвитку Android, хоча є ознаки, що її довгострокові плани можуть поширитися далі.

Картата історія Android із випуском оновлень покращилася завдяки таким ініціативам, як Project Treble та Project Mainline, але фрагментація все ще викликає занепокоєння. Подібним чином, хоча такі компанії, як Samsung та OnePlus, зобов'язуються пропонувати три роки оновлення ОС та оновлення безпеки для багатьох своїх телефонів, є багато виробників обладнання, які все ще припиняють підтримку через два роки або навіть лише 12 місяців.

Смартвони Google на Android - серія Google Pixel - продовжує розділяти критиків та споживачів, але справжнє занепокоєння викликає дедалі більше експериментальних форм-факторів від інших виробників телефонів - форм-факторів, які розширюють межі поточних можливостей Android. Складні телефони та телефони з двома екранами можуть бути новою категорією з розкішними цінниками та привабливістю, але вони вже виявили слабкі сторони Android як ОС для більших розмірів екрану.

Незважаючи на те, що незабаром, можливо, доведеться ще раз пристосуватися до цієї нової хвилі апаратного дизайну, здається розумним передбачити, що Android продовжуватиме домінувати на ринку мобільних ОС. ОС встановлюється на телефонах, які продаються за ціною менше 100 доларів, аж до дорогих флагманських пристроїв, вартість яких перевищує 1000 доларів. Ця

гнучкість у поєднанні з щорічними оновленнями повинна забезпечити, щоб Android залишався лідером у цій галузі на довгі роки.

1.3 Розвиток мов програмування програмування

Так як з операцыйною системою ми визначились і нею стала зручна опенсоурсна операційна система андроїд, перейдемо до мов програмування.

Java - це об’єктно-орієнтована мова програмування, розроблена Джеймсом Гослінгом та колегами з Sun Microsystems на початку 1990-х. На відміну від звичайних мов, які, як правило, призначені для компіляції до власного (машинного) коду, або для інтерпретації з вихідного коду під час виконання, Java призначена для компіляції в байт-код, який потім запускається (як правило, за допомогою компіляції JIT) Віртуальна машина Java.

Сама мова запозичує багато синтаксису з C та C++, але має простішу об'єктну модель і менше засобів низького рівня. Java лише віддалено пов'язана з JavaScript, хоча вони мають схожі назви та мають спільний C-подібний синтаксис.

Java був започаткований Джеймсом Гослінгом під назвою "Дуб" у червні 1991 р. Цілями Гослінга було впровадження віртуальної машини та мови, яка мала звичне позначення на C, але з більшою одноманітністю та простотою, ніж C / C ++. Першою публічною реалізацією стала Java 1.0 в 1995 році. Вона дала обіцянку "Писати один раз, працювати куди завгодно", з безкоштовним виконанням на популярних платформах. Він був досить безпечним, а його безпеку можна було налаштувати, що дозволило обмежити доступ до мережі та файлів. Основні веб-браузери незабаром включили його у свої стандартні конфігурації в безпечній конфігурації "аплет". популярний швидко. Незабаром нові версії для великих і малих платформ (J2EE та J2ME) були розроблені з появою "Java 2". Sun не повідомляв про будь-які плани щодо "Java 3".

У 1997 році Sun звернувся до органу зі стандартів ISO / IEC JTC1, а згодом і до Ecma International, щоб формалізувати Java, але незабаром він вийшов із цього процесу. Java залишається власним фактичним стандартом, який контролюється процесом спільноти Java. Sun робить більшість своїх реалізацій Java доступними безкоштовно, при цьому дохід приносить спеціалізовані продукти, такі як Java Enterprise System. Sun розрізняє свій набір для розробки програмного забезпечення (SDK) та середовище виконання (JRE), яке є підмножиною SDK, основним розрізненням є те, що в JRE компілятор відсутній.

У створенні мови Java було п’ять основних цілей:

1. Вона повинна використовувати методологію об'єктно-орієнтованого програмування.
2. Вона повинна дозволяти виконувати одну і ту ж програму в декількох операційних системах.
3. Вона повинна містити вбудовану підтримку використання комп’ютерних мереж.
4. Вона повинна бути розроблена для безпечного виконання коду з віддалених джерел.
5. Вона повинна бути простим у використанні, вибираючи те, що вважалося хорошими частинами інших об’єктно-орієнтованих мов.

Для досягнення цілей мережевої підтримки та віддаленого виконання коду програмісти Java іноді вважають за потрібне використовувати такі розширення, як CORBA, Internet Communications Engine або OSGi.

Перша характеристика, об'єктна орієнтація ("ОО"), відноситься до методу програмування та мовного проектування. Хоча існує багато інтерпретацій ОО, одна основна відмінна ідея полягає в розробці програмного забезпечення таким чином, що різні типи даних, якими він маніпулює, поєднуються разом із відповідними операціями. Таким чином, дані та код об'єднуються в сутності, які називаються об'єктами. Об’єкт можна сприймати як самодостатній комплект поведінки (код) та

стану (дані). Принцип полягає у відокремленні речей, що змінюються, від речей, які залишаються такими ж; часто зміна певної структури даних вимагає відповідної зміни коду, який працює на цих даних, або навпаки. Цей поділ на цілісні об'єкти забезпечує більш стабільну основу для проектування програмної системи. Намір полягає в тому, щоб спростити управління великими програмними проектами, таким чином покращуючи якість та зменшуючи кількість невдалих проектів.

Ще однією основною метою програмування ОО є розробка більш загальних об'єктів, щоб програмне забезпечення стало більш багаторазовим між проектами. Наприклад, загальний "клієнтський" об'єкт повинен мати приблизно однаковий базовий набір поведінки між різними програмними проектами, особливо коли ці проекти перекриваються на якомусь фундаментальному рівні, як це часто роблять у великих організаціях. У цьому сенсі об'єкти програмного забезпечення, можна сподіватися, можна розглядати більше як підключаються компоненти, що допомагає індустрії програмного забезпечення будувати проекти в основному з існуючих і добре перевірених деталей, що призводить до значного скорочення часу розробки. Повторне використання програмного забезпечення зумовило неоднозначні практичні результати з двома основними труднощами: проектування справді загальних об’єктів недостатньо зрозуміле, а методологія широкого зв’язку з можливостями повторного використання відсутня. Деякі спільноти з відкритим кодом хочуть полегшити проблему повторного використання, надаючи авторам способи розповсюдження інформації про загально використовувані об'єкти та бібліотеки об'єктів.

Друга характеристика, незалежність від платформи, означає, що програми, написані мовою Java, повинні працювати аналогічно на різноманітному обладнанні. Треба мати можливість написати програму один раз і запустити її де завгодно.

Це досягається більшістю компіляторів Java шляхом компіляції коду мови Java "наполовину" до байт-коду (зокрема байт-коду Java) - спрощених машинних інструкцій, специфічних для платформи Java. Потім код запускається на

віртуальній машині (VM), програмі, написаній у власному коді на апаратному забезпеченні, яка інтерпретує та виконує загальний байт-код Java. Крім того, надаються стандартизовані бібліотеки, що дозволяють уніфіковано отримувати доступ до функцій хост-машин (таких як графіка, створення потоків та створення мереж). Зауважте, що, хоча існує явна стадія компіляції, в якийсь момент байт-код Java інтерпретується або перетворюється на рідні машинні інструкції компілятором JIT.

Існують також реалізації компіляторів Java, які компілюються до власного об’єктного коду, наприклад, GCJ, видаляючи проміжний етап байт-коду, але вихід цих компіляторів може виконуватися лише на одній архітектурі.

Ліцензія Sun на Java наполягає на тому, щоб усі реалізації були "сумісними". Це призвело до юридичного суперечки з Microsoft після того, як Sun заявив, що реалізація Microsoft не підтримує інтерфейси RMI та JNI і додала власні функції, що стосуються певної платформи. У відповідь на це Microsoft більше не постачає Java з Windows, а в останніх версіях Windows Internet Explorer не може підтримувати аплети Java без стороннього плагіна. Однак Sun та інші безкоштовно надали доступні системи виконання Java для тих та інших версій Windows.

Перші реалізації мови використовували інтерпретовану віртуальну машину для досягнення мобільності. Ці реалізації створили програми, які працювали повільніше, ніж програми, скомпільовані на власні виконувані файли, наприклад, написані на C або C ++, тому мова страждала від репутації низької продуктивності. Останні реалізації JVM створюють програми, які працюють значно швидше, ніж раніше, використовуючи безліч методів.

Перший прийом полягає у простому компілюванні безпосередньо у власний код, як у більш традиційного компілятора, повністю пропускаючи байт-коди. Це дозволяє досягти хорошої продуктивності, але за рахунок переносимості. Інша техніка, відома як компіляція "вчасно" (JIT), перетворює байт-коди Java у власний код під час запуску програми, що призводить до того, що програма виконується

швидше, ніж інтерпретований код, але також спричиняє накладні витрати на компіляцію під час виконання. Більш досконалі віртуальні машини використовують динамічну перекомпіляцію, в якій віртуальна машина може аналізувати поведінку запущеної програми та вибірково перекомпілювати та оптимізувати критичні частини програми. Динамічна рекомпіляція може досягти оптимізацій, що перевершують статичну компіляцію, оскільки динамічний компілятор може базувати оптимізацію на знаннях про середовище виконання та набір завантажених класів. Компіляція JIT та динамічна рекомпіляція дозволяють програмам Java використовувати переваги швидкості власного коду, не втрачаючи при цьому мобільності.

Портативність - це технічно важка мета, яку можна досягти, і успіх Java у цій меті неоднозначний. Незважаючи на те, що дійсно можна писати програми для платформи Java, які поводяться послідовно на багатьох хост-платформах, велика кількість доступних платформ з невеликими помилками або невідповідностями призвела до того, що деякі пародіювали слоган Sun "Пиши один раз, біжи будь-де" як "Пиши один раз, налагоджуй скрізь ".

Однак незалежна від платформи Java дуже успішна з додатками на стороні сервера, такими як веб-сервіси, сервлети та Enterprise JavaBeans, а також із вбудованими системами на базі OSGi, що використовують вбудовані середовища Java.

Одна ідея автоматичної моделі управління пам'яттю Java полягає в тому, що програмісти повинні бути позбавлені тягаря необхідності виконувати ручне управління пам’яттю. У деяких мовах програміст виділяє пам'ять для створення будь-якого об'єкта, що зберігається в купі, і відповідає за подальше ручне вивільнення цієї пам'яті для видалення таких об'єктів. Якщо програміст забуде вивільнити пам'ять або напише код, який не вдається це зробити вчасно, може статися витік пам'яті: програма буде споживати потенційно довільно великий обсяг пам'яті. Крім того, якщо область пам'яті вивільнюється двічі, програма може стати нестабільною і може вийти з ладу. Нарешті, в середовищах, де не збирається сміття,

існує певний рівень накладних витрат та складність коду користувача для відстеження та завершення розподілу.

У Java ця потенційна проблема зникнула завдяки автоматичному збору сміття. Програміст визначає, коли створюються об'єкти, а середовище виконання Java відповідає за управління життєвим циклом об'єкта. Програма або інші об'єкти можуть посилатися на об'єкт, утримуючи посилання на нього (що, з точки зору низького рівня, є його адресою в купі). Коли посилань на об’єкт не залишається, збирач сміття Java автоматично видаляє недосяжний об’єкт, звільняючи пам’ять і запобігаючи витоку пам’яті. Витік пам’яті все ще може відбуватися, якщо код програміста містить посилання на об’єкт, який більше не потрібен - іншими словами, вони все ще можуть відбуватися, але на більш високих концептуальних рівнях.

Використання збору сміття в мові також може вплинути на парадигми програмування. Наприклад, якщо розробник вважає, що вартість виділення / згадування пам'яті низька, вони можуть вибрати вільніше будувати об'єкти, а не попередньо їх ініціалізувати, утримувати та використовувати повторно. З невеликою вартістю можливих покарань за продуктивність (побудова внутрішнього циклу великих / складних об’єктів), це полегшує ізоляцію потоків (не потрібно синхронізувати, оскільки різні потоки працюють на різних екземплярах об’єктів) та приховування даних. Використання перехідних незмінних об'єктів значення мінімізує програмування побічних ефектів.

Порівнюючи Java та C++, у C++ можна реалізувати подібні функціональні можливості (наприклад, модель управління пам'яттю для певних класів може бути розроблена в C++, щоб покращити швидкість та значно знизити фрагментацію пам'яті), з можливими витратами на додатковий час розробки та деякі складність застосування. У Java збір сміття вбудований і практично непомітний для розробника. Тобто розробники можуть не уявляти, коли відбуватиметься збирання сміття, оскільки це може не обов'язково співвідноситися з якимись діями, явно виконуваними кодом, який вони пишуть. Залежно від передбачуваного

застосування, це може бути вигідним або невигідним: програміст звільняється від виконання низькорівневих завдань, але в той же час втрачає можливість написання коду нижчого рівня.

Синтаксис Java значною мірою походить від C++. Однак, на відміну від C++, який поєднує синтаксис для структурованого, загального та об'єктно-орієнтованого програмування, Java була побудована з нуля і була практично повністю об'єктно-орієнтована: все в Java є об'єктом за винятком атомних типів даних (порядковий та реальні числа, булеві значення та символи) і все в Java написано всередині класу.

Зовнішній вигляд і відчуття програм графічного інтерфейсу, написаних на Java з використанням набору інструментів Swing, сильно відрізняється від власних програм. Можна встановити інший вигляд та відчуття за допомогою підключається системи відчуття та відчуття. Клони Windows, GTK та Motif постачаються Sun. Apple також пропонує зовнішній вигляд Aqua для Mac OS X. Хоча попередні реалізації цього вигляду та відчуттів вважалися відсутніми, Swing в Java SE 6 вирішує цю проблему, використовуючи більше власних процедур малювання віджетів базових платформ. В якості альтернативи, сторонні набори інструментів, такі як wx4j або SWT, можуть бути використані для більшої інтеграції з власною системою вікон.

Первісні типи Java не є об'єктами. Первісні типи містять свої значення в стеку, а не посилання на значення. Це було свідоме рішення дизайнерів Java з міркувань продуктивності. Через це Java не вважається чистою об’єктно-орієнтованою мовою програмування. Однак, починаючи з Java 5.0, автобокс дозволяє програмістам писати так, ніби примітивні типи є їхніми класами-обгортками, і вільно обмінюватися між ними для поліпшення гнучкості. Дизайнери Java вирішили не застосовувати певні функції, присутні в інших мовах ОО, зокрема:

- множинне успадкування
- перевантаження оператора

- властивості класу
- кортежі

Середовище виконання Java або JRE - це програмне забезпечення, необхідне для запуску будь-якої програми, розгорнутої на платформі Java. Кінцеві користувачі зазвичай використовують JRE у програмних пакетах та плагінах веб-браузера. Sun також розповсюджує набір JRE, який називається Java 2 SDK (більш відомий як JDK), який включає такі засоби розробки, як компілятор Java, Javadoc та налагоджувач.

У 2009 році Oracle придбала Sun Microsystems за 7,4 мільярда. Під час придбання було порушено багато питань, які навіть спонукали деяких членів громади оголосити це смертю Java.

Однак Java не вмерла, як мова програмування. В даний час це найпоширеніша мова програмування номер 1 згідно з Індексом TIOBE.

Нещодавно прийнятий цикл випуску обіцяє нову версію Java кожні півроку. Багато дискусій про стан JDK продовжує наповнювати ефір. Ми озираємося на заяву, зроблену в 2006 році, і переформулюємо її для поточного віку: Де буде Java через 15 років?

Очікуючи Java 12, ми повинні озирнутися на історію і побачити, як далеко ми просунулися. Хоча ніхто не може передбачити, що чекає майбутнє, одне впевнене: Java змінила Інтернет і, в свою чергу, світ.

Kotlin - це загальна, безкоштовна, з відкритим кодом, статично набрана «прагматична» мова програмування, спочатку розроблена для JVM (Java Virtual Machine) та Android, що поєднує об’єктно-орієнтовані та функціональні функції програмування. Він орієнтований на взаємодію, безпеку, чіткість та підтримку інструментів. Також виготовляються версії Kotlin, орієнтовані на JavaScript ES5.1 та власний код (з використанням LLVM) для ряду процесорів.

Kotlin виникла в JetBrains, компанії, що стоїть за IntelliJ IDEA, в 2010 році і була відкритою з 2012 року. Команда Kotlin наразі налічує понад 90 штатних членів з JetBrains, а проект Kotlin на GitHub має більше 300 учасників. JetBrains використовує Kotlin у багатьох своїх продуктах, включаючи флагманський IntelliJ IDEA.

Незважаючи на те, що JetBrains офіційно випустив Kotlin у 2016 році, мова стикається з дійсно довгим процесом розробки. Вперше Котлін з'явився в 2011 році. Коли IDE Emperor вперше оголосив про проект Котлін усьому світу. Хоча Котлін сьогодні широко використовується, JetBrains не продавав Котлін. Натомість вони змусили вирішити власні проблеми розвитку.

Більше 70 відсотків наших продуктів були побудовані на Java, говорить Хаді Харірі, віце-президент з питань адвокації розробників у JetBrains, а більшість решти написано в C # від Microsoft. Хоча Java була дуже потужною, але використання Java означає написання великої кількості коду. Що з часом стає важко писати, читати та підтримувати. Навіть для найпростішого коду потрібно оголосити повні класи та об'єкти і все таке, що стає великим головним болем для розробників JetBrains. Вони просто хочуть позбутися зайвої повторюваної роботи.

Команди JetBrains дійсно хотіли використовувати більш сучасну мову, але вони все ще мали багато програм, написаних на Java, які потрібно було б підтримувати. Просто немає сенсу переписувати всі продукти Java на C # або будь-якою іншою мовою. Отже, їм потрібна була мова, сумісна з Java, щоб вони могли зберегти весь старий код таким, який він є, і почати створювати нові функції новою мовою.

Вони також спробували деякі існуючі варіанти, такі як Scala, Groovy та Clojure. Scala була найкращою з усіх, але головним недоліком було те, що вона не була швидкою і настільки простою, на яку розробники JetBrains оглядаються.

Тож команда вирішила створити власну мову з усіма функціями, які вони хотіли. Створюючи команду Kotlin JetBrains, зосередьтеся на трьох основних речах:

1. Швидко (принаймні так швидко, як Java).
2. 100% сумісність з Java.
3. Стислий та виразний код.

Замість того, щоб підтримувати проект внутрішнім, JetBrains відкрив проект у 2012 році за ліцензією Apache з відкритим кодом. JetBrains не отримує прибутку безпосередньо від використання Kotlin серед розробників, але компанія сподівається заробляти гроші на своїх майбутніх основних продуктах, що підтримують Kotlin.

У 2017 році Google IO Google оголосив Kotlin як офіційно підтримувану мову для Android. З тих пір величезна популярність Котліна зростає з кожним днем. Kotlin також використовується в браузерах, і через Kotlin Native в майбутньому ми можемо створювати крос-платформні власні програми.

2 Аналітичне дослідження

2.1 Дослідження проблеми

Якість сну можна оцінювати з об'єктивної та суб'єктивної точок зору. Об’єктивна якість сну стосується того, як важко людині заснути і залишатися у сплячому стані, і скільки разів вона прокидається протягом однієї ночі. Погана якість сну порушує цикл переходу між різними стадіями сну. Суб’єктивна якість сну, у свою чергу, означає відчуття відпочинку та відновлення після пробудження зі сну. Дослідження А. Харві виявив, що безсоння були більш вимогливими до оцінки якості сну, ніж особи, які не мали проблем зі сном.

Неякісний сон пов’язаний із такими захворюваннями, як серцево-судинні захворювання, ожиріння та психічні захворювання. Хоча поганий сон є поширеним явищем серед хворих на серцево-судинні захворювання, деякі дослідження показують, що поганий сон може бути причиною цього. Короткий час сну менше семи годин корелює з ішемічною хворобою серця та підвищеним ризиком смерті від ішемічної хвороби серця. Тривалість сну більше дев'яти годин також корелює з ішемічною хворобою серця, а також інсультом та серцево-судинними подіями.

Як у дітей, так і у дорослих коротка тривалість сну пов’язана з підвищеним ризиком ожиріння, а різні дослідження повідомляють про підвищений ризик у 45–55%. Інші аспекти здоров’я сну пов’язані з ожирінням, включаючи денну дрімоту, час сну, мінливість часу сну та низьку ефективність сну. Однак тривалість сну є найбільш вивченою через вплив на ожиріння.

Проблеми зі сном часто розглядаються як симптом психічних захворювань, а не як причинний фактор. Однак все більша кількість доказів свідчить про те, що вони є і причиною, і симптомом психічних захворювань. Безсоння є важливим провісником великого депресивного розладу ; мета-аналіз 170000 людей показав, що безсоння на початку періоду дослідження показало більш ніж дворазове збільшення ризику для великого депресивного розладу. Деякі дослідження також

вказують на кореляцію між безсонням та тривогою, посттравматичним стресовим розладом та суїцидом. Порушення сну може збільшити ризик розвитку психозу та погіршити тяжкість психотичних епізодів.

Гомеостатична схильність до сну (потреба у сні як функція часу, що минув з останнього адекватного епізоду сну) повинна бути збалансована з циркадним елементом для задовільного сну. Разом із відповідними повідомленнями з циркадного годинника це повідомляє тілу, що йому потрібно спати. Час правильний, коли наступні два циркадні маркери виникають після середини епізоду сну та перед пробудженням: максимальна концентрація гормону мелатоніну та мінімальна температура тіла.

Взагалі кажучи, чим довше організм не спить, тим більше він відчуває потребу спати ("борг під час сну"). Цей драйвер сну згадується як процес S. Баланс між сном і неспанням регулюється процесом, який називається гомеостазом. Індукований або сприйманий недосип називається депривацією сну .

Процес S обумовлений виснаженням глікогену та накопиченням аденозину в передньому мозку, що дезінгібує вентролатеральне преоптичне ядро, дозволяючи гальмувати висхідну сітчасту активуючу систему .

Депривація сну, як правило, спричиняє уповільнення мозкових хвиль у лобовій корі, скорочення тривалості уваги, підвищену тривожність, порушення пам’яті та неприємний настрій. І навпаки, добре відпочений організм, як правило, покращує пам’ять і настрій. Нейрофізіологічні та функціональні візуалізаційні дослідження продемонстрували, що лобові ділянки мозку особливо чутливі до гомеостатичного тиску уві сні.

Існують розбіжності щодо того, скільки боргу під час сну можна накопичити, і чи накопичується борг за час сну щодо середнього сну людини чи якогось іншого еталону. Також незрозуміло, чи поширеність боргу за сном серед дорослих помітно змінилася в індустріальному світі за останні десятиліття. Сонний борг справді свідчить про те, що він є кумулятивним. Однак суб’єктивно люди, здається,

досягають максимальної сонливості після 30 годин пробудження. Ймовірно, що в західних суспільствах діти сплять менше, ніж раніше.

Одним з нейрохімічних показників боргу у сні є аденозин, нейромедіатор, який пригнічує багато тілесні процеси, пов’язані з неспанням. Рівні аденозину збільшуються в корі та базальному передньому мозку під час тривалого неспання та знижуються протягом періоду відновлення сну, потенційно виступаючи в ролі гомеостатичного регулятора сну. Кава та кофеїн тимчасово блокують ефект аденозину, подовжують час затримки сну та зменшують загальний час та якість сну.

Від епох ведення записів, пов’язуючи вузли, до великих даних, техніка аналізу даних розвивалася протягом історії людської цивілізації. У міру припливу мобільного Інтернету, підприємства та уряди навіть окремі люди почали використовувати свій бізнес на портативних пристроях, особливо на смартфонах. В даний час мобільний Інтернет вже перевищив кількість персональних комп’ютерів. Для кращого обслуговування клієнтів дуже важливо визначити їх уподобання та характеристики. Через різне життя, Інтернет-бізнес повинен піклуватися про персоналізовані потреби. Цей спосіб ідентифікації характеристик користувача зазвичай називають профілем користувача.

Точне передбачення поведінки користувачів може допомогти постачальникам послуг мобільного Інтернету оптимізувати свій бізнес за великими сценаріями. Наприклад, система може попередньо завантажити програму, яка має найбільшу ймовірність використання наступного моменту для скорочення часу очікування користувачів. Вставати і лягати спати - два критичні випадки, які зазвичай вказують на початок і кінець людського розкладу, а також розумне обслуговування. Крім того, режим сну може відображати стан здоров'я людини. Можна об'єднати режим сну з іншими характеристиками, такими як вік, стать та робочий час, щоб зробити висновок про недоздоров'я. Крім того, програми можуть давати індивідуальні пропозиції на основі цих висновків. “Pzizz” допомагає людям заснути та “SleepCycle” пропонує послугу пробудження. Ці програми

«колискова» та будильник суворо залежать від активації вручну та детальних налаштувань. Це має сенс звільнити користувачів від багаторазових механічних операцій та зробити послуги більш природними та гнучкими.

Смартфон, інтегрований з декількома датчиками та мікросхемами, стає основною платформою збору даних користувачів завдяки тісному зв'язку з людьми. Багато дослідників проводили дослідження та додатки на основі записів про використання смартфонів для моделювання профілю користувача. Застосовували контрольовані методи навчання, щоб зробити висновки про профілі користувачів, такі як релігія, статус стосунків, розмовні мови, країни, що цікавлять, і чи є користувач батьком маленьких дітей, спостерігаючи лише один знімок встановлених програм. Експерименти на 200 смартфонах показують, що для більшості рис модель може досягти понад 90% точності. Зосереджено на аналізі профілів користувачів шляхом видобутку інформації про використання з декількох програм на базі NFC, як на смартфоні, так і на сервері. Спершу спробував зрозуміти користувачів за допомогою інформації ідентифікатора набору послуг (SSID). Він також запропонував структуру очищення даних та збагачення інформації, щоб забезпечити розуміння переваг користувачів на основі зібраних журналів Wi-Fi. Розробили метод прогнозування наступної програми, який покращує досвід використання додатків на головному екрані.

Ця робота спрямована на вирішення двох завдань. Перший - виявити випадки вставання та відходу до ліжка. Другий - передбачити наступний час пробудження та час сну.

Спочатку питання вивчення сну висувають інститути медичної допомоги. У лікарні застосовують полісомнографію (PSG) для діагностики розладів сну, реєструючи хвилі мозку, рівень кисню в крові, частоту серцевих скорочень та дихання, а також рухи очей та ніг учасників. Однак для цього методу потрібні професійні пристрої та спеціалізоване середовище сну, що неможливо для щоденного використання.

Носимі пристрої - другий вибір. Пристрої носять на зап’ясті, використовуються для реєстрації рухів та оцінки параметрів сну, які називаються зап’ястним актиграфом (АСТ). Незважаючи на те, що актиграф викликає сумніви в правильності ідентифікації неспання чи сну з медичної точки зору, носяться пристрої з точним алгоритмом ідентифікації все ще придатні для щоденного використання через свою низьку вартість та портативність. Пропонували використовувати зап’ястний актиграф для оцінки часу сну на ранній стадії. Пізніше розробили систему моніторингу на її основі та замінивши ручне обчислення алгоритмами підрахунку. І ще більше вдосконалили його, змусивши алгоритми відрізнити сон від неспання приблизно за 88% набраних хвилин. В останні роки розробка інтегральної схеми (IC) та Інтернету речей (IoT) дозволяє мініатюризувати датчики. Трекери активності, такі як браслет Fitbit та розумні годинники, такі як Apple Watch, дозволяють збирати та аналізувати дані про рух зап’ястя користувачів. Використовували штучну нейронну мережу для класифікації сну та неспання на основі даних актиграфа нічного зап’ястя. Інтегровані некеровані алгоритми машинного навчання та евристика знань домену для виявлення стану сну чи сну.

Без додаткового пристрою існують також методи, які використовують смартфон як монітор і завантажують дані для обчислення тривалості сну. Представлена модель під назвою BES, яка визначає тривалість сну шляхом аналізу моделей використання смартфонів. Функції включають зовнішню яскравість, використання телефону, стан руху та фоновий шум. Для збору цих атрибутів потрібна програма постійного моніторингу на смартфонах. Розроблена програма під назвою Toss “N” Turn, яка може визначати стан сну та якості сну за захопленими датчиками даних, включаючи амплітуду звуку, прискорення, інтенсивність світла та близькість екрану, запущені програми, заряд батареї та стан екрану. Цей документ перевіряє свою дійсність, повідомляючи користувачам про те, щоб вони щоранку вводили істину.

Однак існують три обмеження цих методів, перелічених вище:

1) Незважаючи на те, що носні пристрої швидко розвивалися в останні роки, рівень проникнення на ринок все ще обмежений. У всьому світі поставки пристроїв, що носяться, складають 25,1 млн., Тоді як є 344,4 млн. Смартфонів (джерело: International Data Corporation, IDC). Крім того, різні виробники застосовують різні типи датчиків, і не існує єдиного API для розробки програмного забезпечення для всіх них. Це рішення, очевидно, обмежує рівень охоплення інтелектуальних послуг.

2) Більшість сучасних методів вимагають передачі даних на веб-сервер та виконання алгоритмів на кластері серверів. Це буде пов'язано з проблемою захисту конфіденційності. Завантаження даних про конфіденційність без явного дозволу користувачів у багатьох країнах може спричинити суворе покарання. Загальний регламент про захист даних (GDPR) регламентував, що передача даних вимагає згоди користувача в Європейському Союзі. Окрім того, заявлення про збір різних великих обсягів даних викликало б огиду користувачів.

3) Обчислювальні ресурси смартфона суворо обмежені. Великі та важкі обчислення можуть спричинити швидку втрату енергії, високу завантаженість пам'яті та ядра. Кожен із цих факторів обмежить час роботи пристрою та призведе до зменшення користувацького досвіду. Крім того, обчислювальне середовище на смартфоні не дуже добре підтримує машинне навчання, особливо алгоритми глибокого навчання.

Для другого завдання, наскільки нам відомо, попередні дослідження рідко можна зустріти. Літератури, отримані за такими ключовими словами, як «передбачення часу сну», завжди зосереджувались на тому, як виявити стан пробудження / сну користувача або обчислити тривалість сну. У цій роботі ми класифікуємо їх як рішення першого завдання - виявлення. Оскільки інтелектуальні служби на основі профілю користувача та сценарію використання нещодавно стали популярними, відсутність досліджень щодо прогнозування часу пробудження та сну є обґрунтованою, наприклад, Google випустив Awareness API на конференції розробників вводу-виводу, 2016. API поінформованості

фокусується на забезпеченні контекстних служб шляхом видобутку звичок користувачів. Будильник не порушить вашого сну, якщо він дізнається, коли ви засинаєте вчора ввечері та наступної зустрічі. ПЧС робить крок вперед, намагаючись передбачити подальші дії користувача, що є цінною та новою роботою.

Вклади в нашу роботу такі: (1)Легкий алгоритм прогнозування: ПЧС просто використовує дані, зібрані зі смартфона, без додаткових носимих пристроїв. Дані журналу стану екрану можна отримати з операційної системи смартфона, оскільки загоряння екрана та згасання події є стандартизованими системними трансляціями. Крім того, завдяки всюдисущості смартфона, ПЧС може охоплювати найширшого потенційного користувача.(2)Уникайте дилеми щодо збору даних, не порушуючи правил захисту конфіденційності. ПЧС також не пов'язаний з веб-сервісом або хмарними обчисленнями. Іншими словами, ПЧС виконує роль процесора даних, а не контролера. Користувачі не будуть турбуватися про ризик розголошення особистої інформації.(3)ПЧС споживають обмежений ресурс на смартфоні. Для цього не потрібні спеціальні рамки або додаткова бібліотека. Розрахунок не спричинив би помітних затримок. За допомогою наших експериментів ПЧС виконує прийнятну складність.

2.2 Проблематика

Процес виявлення та прогнозування АТС може бути придушений багатьма інцидентами. Недостатньо використовувати лише дані журналу стану екрана. У реальних проектах та дослідженнях ми зіткнулися з такими труднощами.

Перший - це ненормальна поведінка. Хтось живе регулярно, але повертається додому дуже пізно або час від часу залишається на вулиці. Цей випадок слід визначити як відхилення в теорії аналізу даних. ПЧС містить моделі, засновані на знанні шаблонів, тому ці викиди слід виявити та відмовитись. Рисунок 2.1 демонструє типовий відхилення.

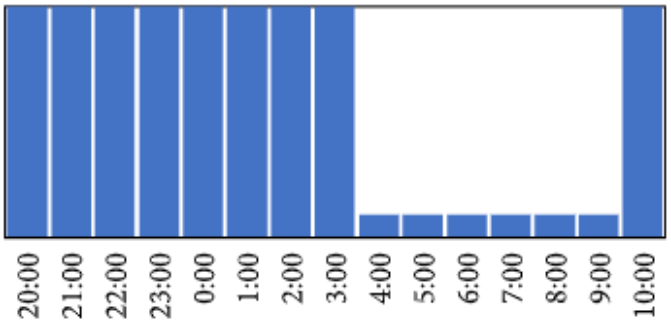


Рисунок 2.1 - Пізній випадок повернення додому (користувач завжди лягає спати близько 22:00).

Друге - несподіване переривання. Коли люди сплять, вхідні дзвінки, отримання повідомлень та сповіщення від програм можуть засвічуватися на екрані, дзвонити або вібрувати, а потім спричиняти пробудження. Також можливо, що люди засинають глибоким сном і ігнорують ці події. Але для даних журналу стану екрану тривалий період гасіння екрану переривається дивною подією. Стани екрану у випадку несподіваного переривання показані на рисунку 2.2 .

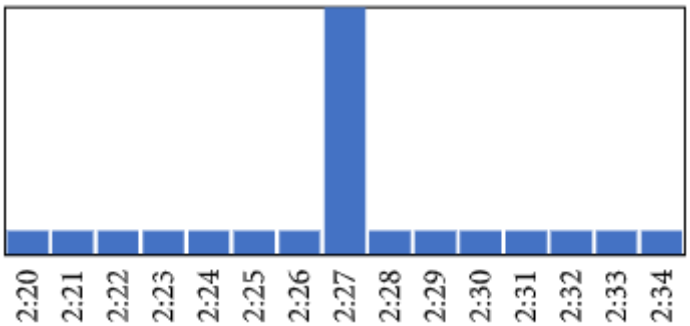


Рисунок 2.2 - Випадок несподіваного переривання (вхідний дзвінок о 2:27).

Третій - вимкнення смартфона. Якщо смартфон вимкнено, стан екрана та допоміжні події відсутні. Деякі люди не люблять режим польоту, вони вимикають смартфон перед сном і запускають його, коли встають. Але гірше, користувач може забути зарядити акумулятор, а потім автоматично вимкнути живлення

смартфона. Втрата даних журналу є найбільшою загрозою та ризиком ПЧС. На рисунку 2.3 показано стан екрану випадку вимкнення.

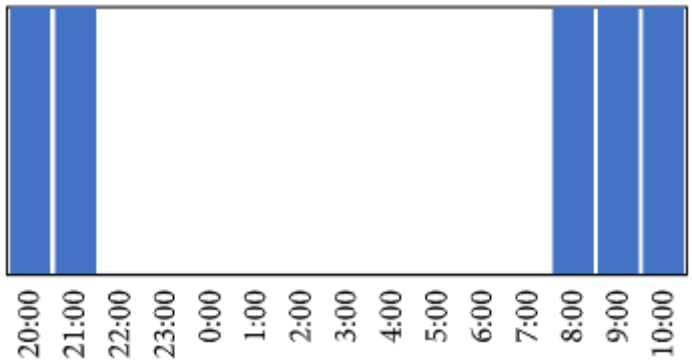


Рисунок 2.3 - Вимкнення телефону (користувач вимикає смартфон до 22:00, а запускає після 8:00).

Три вищезазначені труднощі згадуються при виявленні, а четверта стосується передбачення. Зважаючи на різний життєвий досвід, норми часу неспання та сну між людьми пристосовані.

Більшість людей працює в офісі або на заводі, тобто в стабільному цільовому місці. Вони показують ідентифікований час пробудження та розподіл сну протягом робочих днів. Але багато з них - вихідні у вільний стиль. Інші не працюють з понеділка по п’ятницю. Через свої професійні особливості вони демонструють дивні розподіли, такі як ходити на роботу кожні два дні чи три дні роботи та один день відпочинку. Це призводить до великої помилки при прогнозуванні за допомогою простих моделей часових рядів або спробі описати ці закономірності за тижнями та вихідними. На рисунку 2.4 показані схеми будніх днів та вихідних днів.

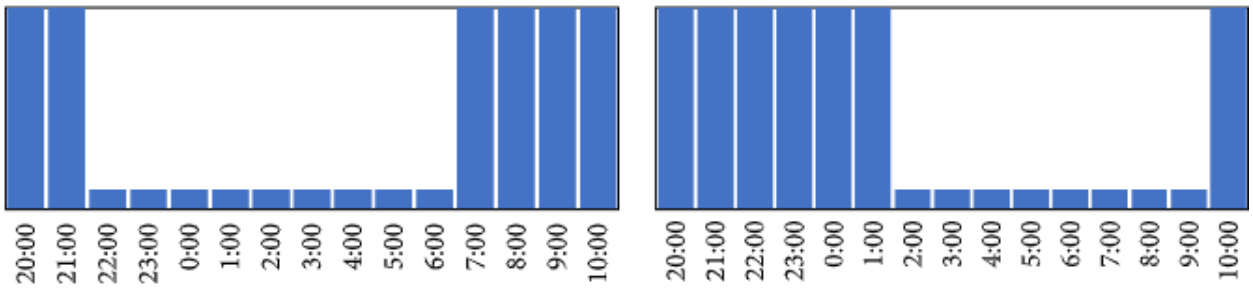


Рисунок 2.4 - Схеми буднів та вихідних (у вихідні лягайте спати і вставайте пізніше робочих днів).

Окрім труднощів, ми можемо використати дві переваги для виправлення та вдосконалення нашої моделі. Одне - подія будильника. Для більшості людей вони ходять на роботу в будні. Момент дзвінка будильника - це також час пробудження. Це правило має дуже високу впевненість, і ми можемо перевірити подію, що активується прискорювачем, щоб підтвердити це. На рисунку 2.5 показано стан екрана, коли відбувається подія будильника.

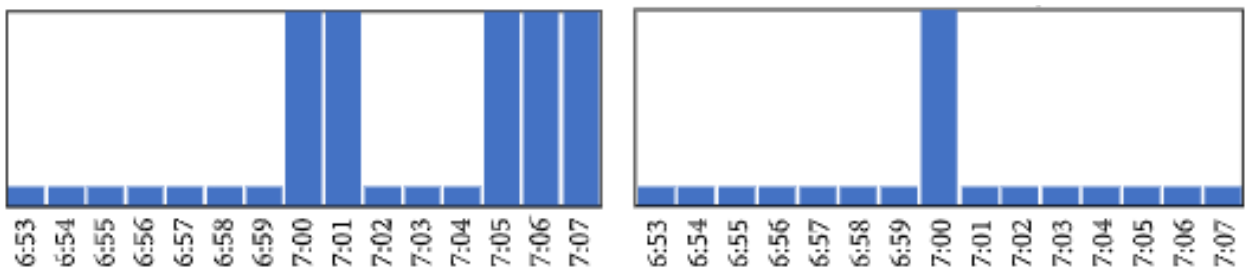


Рисунок 2.5 - Подія будильника (будильник дзвонить о 7:00 і будить користувача).

Інша подія - активована акселерометром. Коли активовано акселератор, ми знаємо, що смартфон переміщується і користувач будить. На рисунку 2.6 показано стан екрана при активації акселерометром.

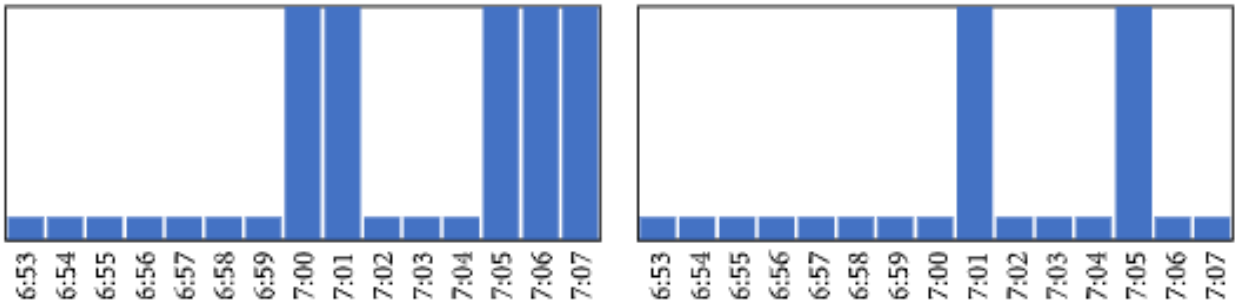


Рисунок 2.6 - Активована подія прискорювача (користувач торкається екрана о 7:01, щоб перестати дзвонити, і встає о 7:05).

Ми долаємо та зменшуємо ефект чотирьох зазначених вище труднощів за допомогою наступних рішень.

(1) Аномальна поведінка. БТП припускає, що люди сплять вдома. Він відмовився від послідовностей не вдома, щоб уникнути впливу ненормальної поведінки. На рисунку 2.7 показано стан екрану після відмови від ненормальної поведінки.

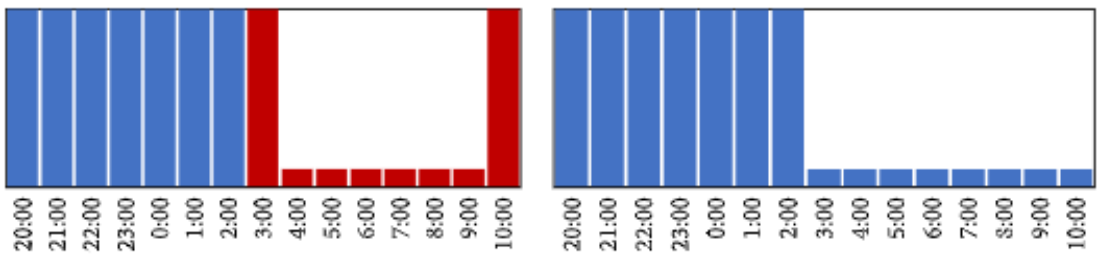


Рисунок 2.7 - Відмовитися від ненормальної поведінки (послідовність після 3:00 відмовляється від виходу додому).

(2) Несподіване переривання. Порушення сну вирішуються шляхом злиття суміжних періодів стану загасання екрану. Тут ми визначаємо центральну точку сну, приблизно 3 години, найсонніший час доби. Коли людей прокидає подія переривання біля центральної точки сну, вони, швидше за все, залишаться спати

через кілька хвилин. Але коли вони менш сонні, наприклад о 5:00, люди можуть вставати прямо. На рисунку 2.8 показаний випадок про зв'язки двох суміжних періодів.

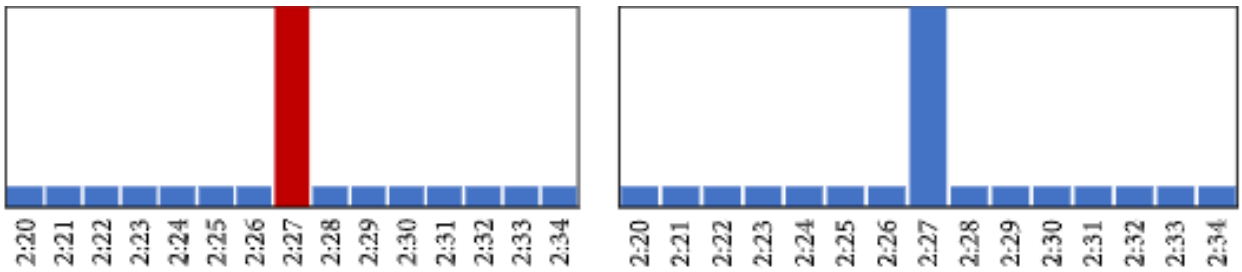


Рисунок 2.8 - Підключення сусіднього періоду (ПЧС ігнорує світло в 2:27).

(3) Вимкнення смартфона. Для користувачів, які регулярно та розумно вмикають і вимикають, ПЧС розглядає події ввімкнення та вимкнення як час пробудження та час сну. Але ми трактуємо нерегулярне вимкнення живлення як відхилення і відмовляємось від нього. На рисунку 2.9 показано стан екрана, коли сталася подія вимкнення живлення.

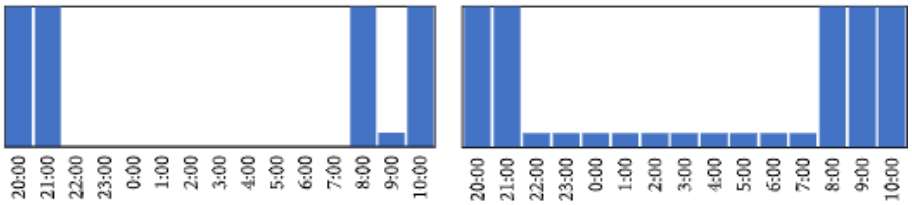


Рисунок 2.9 - Розумна подія вимкнення живлення (під час вимкнення стан екрану має значення "нуль", але не "0").

(4) Індивідуальні правила між людьми. Раніше ми вважали, що прогнозування поведінки користувачів має базуватися на робочих днях і вихідних днях, але це заплутано в налаштованих правилах робочого дня, що призводить до

більшої помилки. Потім ми змінили нашу стратегію та порівняли поточну послідовність стану екрана з іншими послідовностями кожного минулого дня. Найбільш схожі послідовності представляють однакову модель поведінки. Оскільки прокидання та лягання спати - це початок і кінець цілоденних занять, вони природно відображаються у цій подібній системі моделей. На рисунку 2.10 показано дві подібні моделі поведінки.

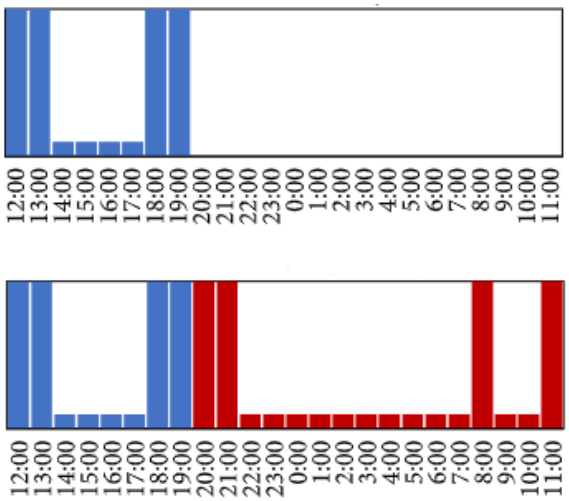


Рисунок 2.10 - Подібна модель поведінки (наступний час сну та пробудження передбачається як 22:00 та 8:00).

Підводячи підсумок, ми ефективно покращуємо точність та доступність послідовності стану екрану при виявленні та прогнозуванні часу пробудження та сну, представляючи низку допоміжних подій. Основний процес ПЧС показаний на рисунку 2.11. ПЧС спочатку збирає дані журналу стану екрану користувача та інші допоміжні події, а потім здійснює пошук та підключення періодів гасіння головного екрану, щоб отримати щоденний результат виявлення. Тоді прогнозування здійснюється шляхом узгодження поточної послідовності стану екрана з історичними послідовностями, такими як K-Найближчий сусід.

Рисунок 2.11 - Потік обробки ПЧС.

2.3 Структура даних алгоритму

По-перше, ми визначаємо структуру даних. Дані журналу містять інформацію як про стан екрану, так і про допоміжний фактор. Подія складається з дати, моменту та події. (Рисунок 2.12).


```
{
"Date": "2018-09-01",
"Moment": "22:39:02",
"Event": "SON"
}.
```


Рисунок 2.12 – Дані журналу.

Різні події відрізняються кодами подій. Тут код SON являє собою загоряння екрану. Для першого завдання, визначення часу пробудження та сну, ми вводимо набір даних журналу з усіма типами подій зі смартфона протягом багатьох днів. Ось частина цього. (Рисунок 2.13)


```
{ …
{"Date": "2018-09-01", "Moment": "22:39:02", "Event": "SON"}, //Screen light on;
{"Date": "2018-09-01", "Moment": "22:45:02","Event": "SOFF"}, //Scree
extinguish;
{"Date": "2018-09-01", "Moment": "22:49:50", "Event": "AA"}, //Pick out phone;
{"Date": "2018-09-01", "Moment": "22:49:55", "Event": "SON"}, //Unlock phone;
{"Date": "2018-09-01", "Moment": "22:50:01", "Event": "SOFF"}, //Screen
extinguish;
{"Date": "2018-09-01", "Moment":"22:50:02", "Event": "POFF"} //Phone
shutdown.
… }.
```

Рисунок 2.13 – Усі типи подій.

Результат виявлення на рисунку 2.14.

{ …
{"Date": "2018-09-01", "Bedtime": "22:39:02", "Wake time": "6:29:59"},
{"Date": "2018-09-02", "Bedtime": "23:12:45", "Wake time": "6:40:04"},
{"Date": "2018-09-03", "Bedtime": "22:59:12", "Wake time": "6:30:09"},
… }.

Рисунок 2.14 – Результат виявлення.

Час людського сну, як правило, відповідає циклам днів. Обговорюючи один день, ми повинні визначити його початок. Ми встановлюємо граничний пункт між двома днями о 12:00 замість 0:00. Тоді ми уникаємо переривання безперервного сну, що також є продовженням періоду згасання екрану в нашій моделі.

Для другого завдання, передбачення часу пробудження та сну, ми також вводимо набір даних журналу, як зазначено вище. Результат прогнозування виглядає наступним чином, якщо поточний час "2018-09-10 18:00:00",

{“Date”:”2018-09-10”,”Bedtime”:”22:40:15”,”Wake time”:”60:30:00”}.

Тут час пробудження стосується фактично "2018-09-11 6:30:30".

2.4 Збір та попередня обробка даних

Перш ніж запускати процес виявлення та прогнозування, ми повинні перетворити дані журналу на кілька двійкових послідовностей. Враховуючи, що в день відбувається 1440 хвилин, ми поділяємо різні події на шість послідовностей довжиною 1440. Для стану екрану 0 означає погашення екрану, а 1 - загоряння екрану. Послідовності наведені в таблиці 2 . Для стану екрану та стану живлення ми відібрали першу секунду стану хвилини. Однак для іншої послідовності, якщо якась подія відбулася за хвилину, статус буде 1. Рисунок 2.15 демонструє графік роботи волонтера.

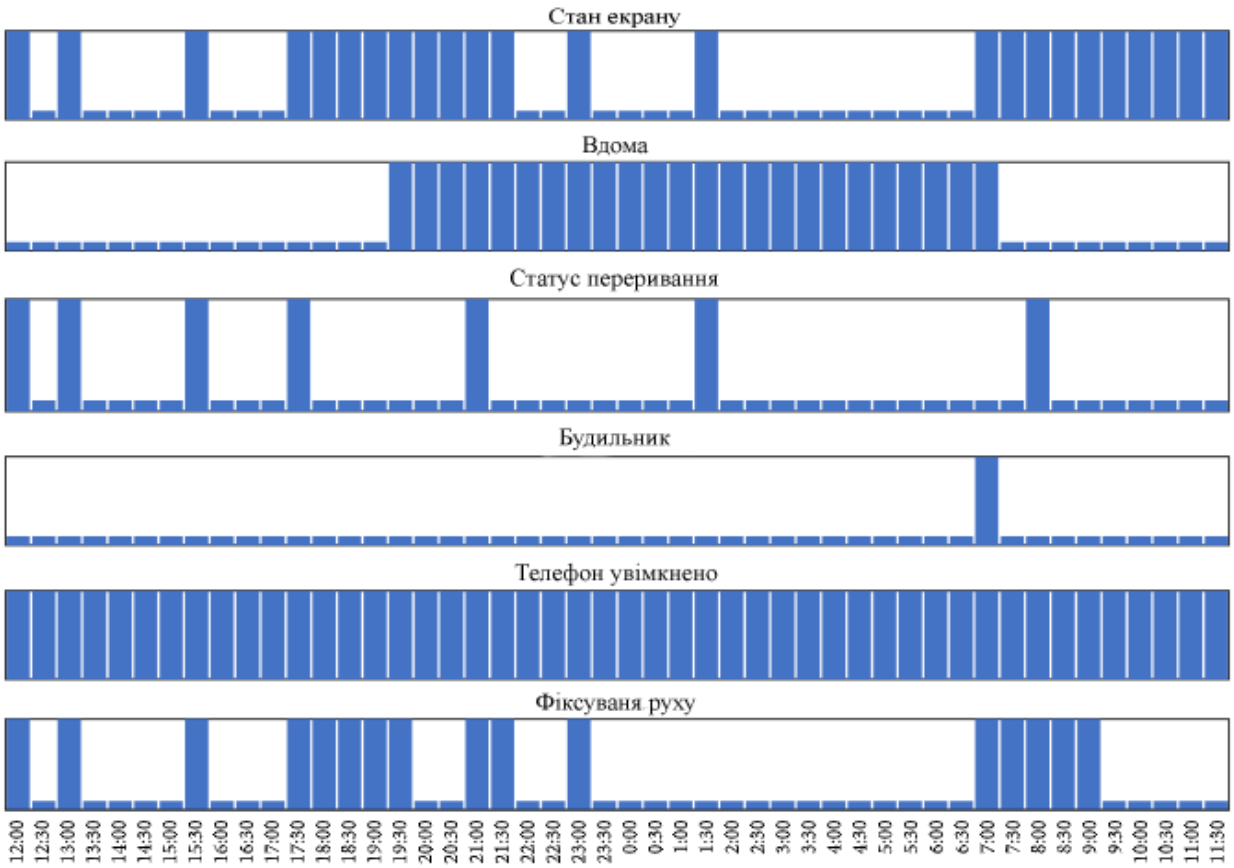


Рисунок 2.15 – Цілодобові послідовності подій.

2.5 Виконання методу.

ПЧС зазвичай має дві основні фази - виявлення та прогнозування.

У частині виявлення ПЧС виконують чотири кроки для кожного дня, який потрібно виявити:

1) Відфільтруйте період загасання екрану, коли користувач відсутній вдома, шукаючи послідовність стану будинку.

2) Шукайте послідовність статусів екрану та з’ясовуйте, чи триває період гасіння екрану, який триває понад 30 хвилин, потім сортуйте їх за шкалою часу.

3) Для кожного періоду, вибраного на кроці 2, ПЧС використовує (2.1), щоб визначити, чи можна цей період об'єднати з сусідніми періодами; тобто, якщо тимчасовий інтервал між ними коротше ніж *T*, ми з'єднаємося дві суміжних послідовностей , як один. У формулі 1, *T* є поріг, який визначає, слід поєднати дві послідовності, *l* використовуються для настройки абсолютного значення, *t* знаходиться в декількох хвилинах від 12:00 до середини періоду, *c* знаходиться в декількох хвилинах від 12:00 до часу , що люди мають глибокий сон, і *off* це значення зміщення.

$$T = 1 \cdot \frac{1}{1+e^{|t-c|/60-off}} \tag{2.1}$$

де *T* – поріг;

l - використовуються для настройки абсолютного значення;

t - знаходиться в декількох хвилинах від 12:00 до середини періоду;

c - знаходиться в декількох хвилинах від 12:00 до часу, що люди мають глибокий сон

off - значення зміщення.

4) Налаштуйте результат за допомогою послідовностей стану переривання, стану будильника, стану живлення та стану руху.

Після отримання часу пробудження та сну, частина передбачення має наступні кроки:

1) Зберіть послідовність стану екрана сьогодні (з 0:00 до поточного часу).

2) Обчисліть подібність між сьогоднішньою послідовністю екранів та кожною історичною послідовністю, використовуючи (2.2). Тут *s* є послідовність статусу екрана з 0:00 до поточного часу, *t* це послідовність, яку потрібно порівняти, і *m* становить хвилини з 0:00 до поточного часу.

$$dist(s,t) = \sum_{i=1}^{'''} |s_i - t_i|$$

(2.2)

3) Виберіть результат виявлення відповідно до k найбільш схожих послідовностей та використовуйте (2.3) для отримання результату. Отримане значення є результатом передбачення, s_i є i_{th} послідовністю k найбільш подібних послідовностей і D_i є попереднім i_{th} результатом виявлення.

$$\sum_{i=1}^{k} \left[\frac{\sum_{j=1}^{k} (dist(s, s_j) - dist(s, s_i))}{(k-1) * \sum_{j=1}^{k} dist(s, s_j)} * D_i \right]$$

(2.3)

3 Реалізація алгоритму та розробка додатку

3.1 Результат експерименту

Критеріями вимірювання для виявлення є середня абсолютна похибка (MAE), показана в (3.1). Тут – n це кількість людей, які беруть участь у цьому експерименті, y_i є справжнім значенням та $f(x_i)$ виявленим значенням методів, що підлягають тестуванню.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \int(x_i)|}{n} \tag{3.1}$$

Критеріями вимірювання для прогнозування є RMSE, які показані в (3.2). Ось N кількість зразків. x_i представляє справжнє значення і x_i' є передбачуваним значенням.

$$RMSE = \sqrt{\frac{\sum_{1}^{N} (x_i - x_i')^2}{N}} \tag{3.2}$$

Наш набір даних містить 30 даних про добровольців. Вони встановили ту саму програму збору даних, яку ми розповсюджували на своїх смартфонах. Додаток має чотири місії: збір даних журналу, виявлення часу пробудження та сну, завантаження оригінальних даних та результатів на сервер та отримання зворотного зв'язку. Тут ми завантажуємо дані для налагодження та підписуємо угоду з волонтерами. Клієнтська програма постійно проживає в Android і збирає дані журналу системи вдень і вночі. Наприкінці дня, як правило, близько 0:00, починається наш метод виявлення, щоб виявити, коли користувач прокидається і

лягає спати. Він також регулярно відображає анкети та повертає відгук на сервер. Ці опитувальники мають лише просте запитання: коли ви заснули вчора ввечері, а сьогодні прокинулись? Щоб волонтери уважно відповідали на запитання, деякі відгуки, які завжди плутають із значенням передбачення, фільтруються після перевірки їх вихідних даних журналу. Після завершення 13-денного безперервного тестування 39 добровольців провели експерименти. Але дев'ять із них були відфільтровані за необережні відповіді. Під час експерименту ми провели першу фазу ПЧС для виявлення часу неспання та сну та провели BES, запропоновану Chen et.al. [17], для виявлення тривалості сну. Потім ми провели другу фазу ПЧС, ковзну середню (MA) [20] та експоненціальне згладжування (ES) [20], щоб передбачити майбутній час пробудження та час сну на лівих 30 зразках із записами на 13 днів, а потім розрахувати RMSE.

Всі експерименти проводились на смартфонах Huawei P10 добровольців із EMUI 8.0 (Android 8.0), Kirin 960 (4 × 2.4GHz + 4 × 1.8GHz), 4GB RAM і 64GB ROM. Ми використовували Android Studio 2.3.3 з Android SDK 7.0 та Java 1.8 для розробки програми збору даних, яка регулярно збирає та завантажує дані на сервер. Сервер для прийому даних запускає середовище CentOS 6.5 у поєднанні з процесором Intel (R) Xeon (R) 4 × 2,5 ГГц, 64 ГБ оперативної пам'яті, 1200 ГБ HD при 15000 об / хв. Для порівняння ПЧС та інших алгоритмів, обидва вони були виконані на робочій станції Think Station P300 з 64-бітною системою Windows10, Intel i7@4×3,6 ГГц, 16 ГБ оперативної пам'яті @ 1600 МГц, жорстким диском SATA3 2 ТБ при 7200 об / хв.

3.2 Результат виявлення

Алгоритм BES, який ми застосували, має MAE 48,19, а не 42,5 у роботі Чи на через різні набори даних. Результат порівняння наведено на рисунку 3.1. ПЧС має набагато нижчий рівень MAE, ніж BES, на тому ж практичному наборі даних.

Завдання	Визначення тривалості сну	
Алгоритм	BTP	BES
MAE	25,84	48.19

Рисунок 3.1 - Виявлення тривалості сну.

У рівнянні (3.1) є три параметри, *c*, *off*, і *l*. Ми використовуємо метод змінних керування для здійснення налаштування параметрів, використовуючи MAE як метрику еволюції. Рисунок 3.2 показує, що найкраще - близько 900. Це означає, що 3:00 - це справді центральна точка сну людей.

c	Wake Time	Bedtime	Sleep Duration
840/2:00	12.99	21.86	29.57
900/3:00	10.56	19.09	25.84
960/4:00	14.61	19.32	27.13

Рисунок 3.2 - MAE різні *l* (*off* = 3, *c* = 900).

Ми доводимо, що ПЧС та BES дотримуються лінійної складності. Вони швидкі, тому ми повторили їх, щоб записати їх трудомісткість. ПЧС не потребує регресії як BES, але повинен переносити важче включення-виключення.

3.3 Результат передбачення

Ми порівнюємо ПЧС із ковзним середнім (MA) та експоненціальним згладжуванням (ES). MA та ES - це два класичні алгоритми прогнозування часових рядів. ПЧС перевершує показники MA та ES за всіма показниками, оскільки час пробудження та час сну мають періодичність, і цикли різняться у кожного.

Ми застосували метод контрольних змінних для виявлення впливів різних параметрів *m* у (3.2) та *k* (3.3). Це доводить, що навчання збірці використовує надійність. Але ми маємо записи лише за 13 днів; *k* обмежена.

Точність зростає із довшою порівняною послідовністю. Ось, *m*=1; 75% тривалості дня забезпечує баланс між точністю та зручністю використання.

D представляє кількість історичних днів, які ми використовуємо для прогнозування. Якщо у нас буде більше історичних зразків, модель буде більш надійною. Ось *D*=6 може відповідати умові переобладнання.

Часові складності ПЧС, MA та ES такі . На практиці MA та ES виконувались швидше, оскільки розмір даних набагато менший. ПЧС використовує історичну послідовність стану екрана для відповідності подібного шаблону. Кожне передбачення вимагає обходу, і представляє кількість днів, що порівнюються. Однак MA та ES використовують лише історичний час неспання та сну. Ця різниця незначна при роботі в смартфоні замість централізованого обчислення на сервері.

3.4 Розробка додатку

Як завжди, першим кроком при створенні додатку для Android є налаштування середовища розробки. На щастя, це легко зробити за допомогою Android SDK. Нам просто потрібно було завантажити Android Studio і встановити її у своїй системі. Після завершення встановлення та ініціалізації у нас з’явились усі необхідні інструменти для запуску нашого проекту Android (Рисунок 3.4).

Наступним кроком є створення нового проекту Android за допомогою Android Studio. У діалоговому меню, що з'явилося, активували підтримку Java, оскільки саме цією мовою ми будемо користуватися. Ми використовували Kotlin та Java для нашого додатка, оскільки Kotlin вимагає набагато менше коду порівняно з Java, що призводить до швидшого написання коду, але не все можна реалізувати на ньому.

Що стосується рівня API, вам слід вибрати рівень API 26, оскільки наш додаток не вимагає розширених функцій, які існують в останньому SDK.

Остання частина процесу створення проекту є основною діяльністю програми. Отож, маючи це на увазі, ми назвали це HomeActivity та використаємо Empty Activity як шаблон для нашої початкової діяльності (Рисунок 3.3).

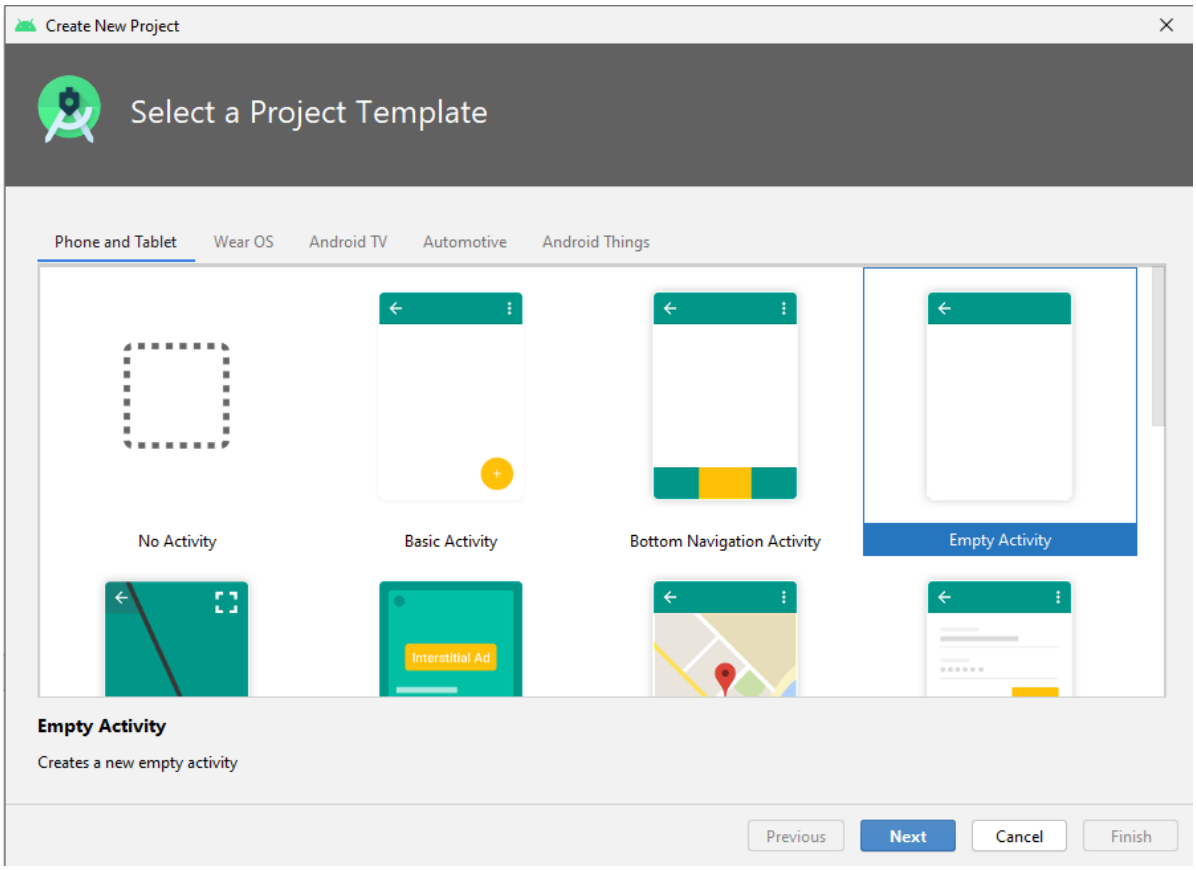


Рисунок 3.3 – Вікно створення нового проекту у Android Studio.

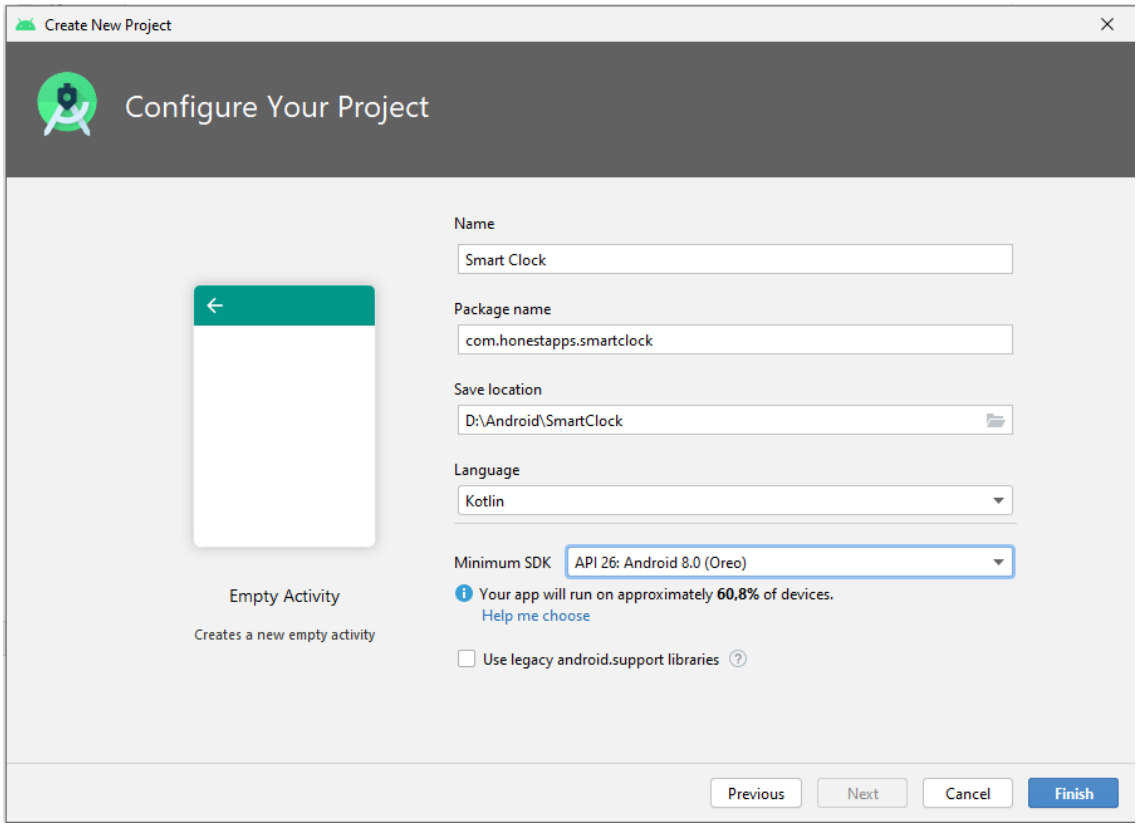


Рисунок 3.4 – Вікно вибору назви та версії SDK для проекту у Android Studio.

Далі потрібно з пустого макету (Рисунок 3.5) зробити головний екран додатку який і буде зустрічати користувачів при вході в додаток.

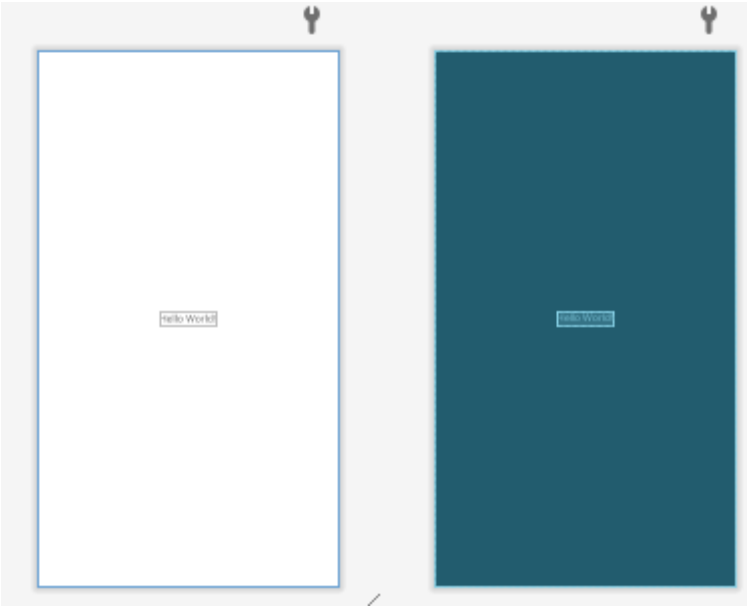


Рисунок 3.5 – Пустий макет головного екрану.

Щоб на ньому показувався весь список будильників потрібно зробити шаблон вигляду кожного будильнику. Для цього створили файл з назвою list_row_classic.xml. Тут розмістили елементи таким чином, щоб ними було комфортно користуватись і при цьому зберігся гарний візуальний вигляд. Згодом в нас вийшов такого вигляду шаблон (Рисунок 3.6). Тут ми розмістили інформацію з часом, на який налаштовано будильник, дні тижня коли буде він спрацьовувати, назву для будильника, яку буде налаштовувати користувач, перемикач стану будильнику та кнопку для налаштування інших параметрів будильника.

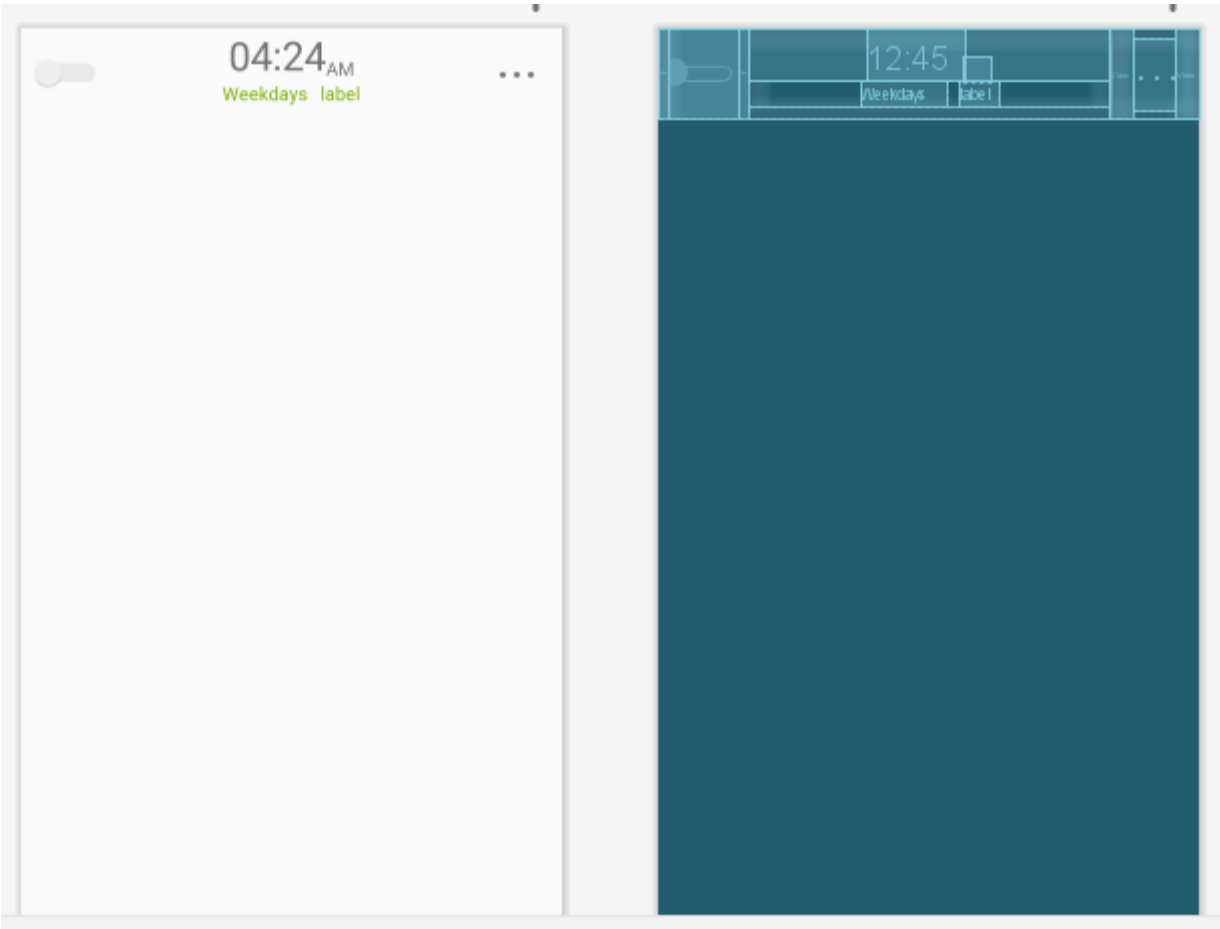


Рисунок 3.6 – Шаблон вигляду будильника у списку.

Після створення шаблону необхідно було налаштувати його відображення, та зв’язку його з головним екраном. Після цього налаштували відображення на головному екрані списку, для вірного відображення в зв’язку з тим, чи є створений будильник, чи немає. Якщо немає, то створити FAB для створення нових будильників (Рисунок 3.7).

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View {
    logger.d("onCreateView $this")

    val view = inflater.inflate(R.layout.list_fragment, container,  attachToRoot: false)

    val listView = view.findViewById(R.id.list_fragment_list) as ListView

    listView.adapter = mAdapter

    listView.isVerticalScrollBarEnabled = false
    listView.setOnCreateContextMenuListener(this)
    listView.choiceMode = AbsListView.CHOICE_MODE_SINGLE

    listView.onItemClickListener = AdapterView.OnItemClickListener { _, listRow, position, _ ->
        val id = mAdapter.getItem(position).id
        uiStore.edit(id, listRow.tag as RowHolder)
    }

    registerForContextMenu(listView)

    setHasOptionsMenu(true)

    val fab: View = view.findViewById(R.id.fab)
    fab.setOnClickListener { uiStore.createNewAlarm() }

    lollipop {
        (fab as FloatingActionButton).attachToListView(listView)
    }

    alarmsSub =
            prefs.listRowLayout()
                    .switchMap { uiStore.transitioningToNewAlarmDetails() }
                    .switchMap { transitioning -> if (transitioning) Observable.never() else store.alarms() }
                    .subscribe { alarms ->
                        val sorted = alarms
                                .sortedWith(Comparators.MinuteComparator())
                                .sortedWith(Comparators.HourComparator())
                                .sortedWith(Comparators.RepeatComparator())
                        mAdapter.clear()
                        mAdapter.addAll(sorted)
                    }

    return view
```

Рисунок 3.7 – Код налаштування відображення головного екрану.

Налаштувати видалення непотрібних будильників також потрібно, тому ми реалізували і цю функцію. Тут саме додали змінення налаштувань та можливість активувати виключений будильник (Рисунок 3.8).

```
override fun onContextItemSelected(item: MenuItem): Boolean {
    val info = item.menuInfo as AdapterContextMenuInfo
    val alarm = mAdapter.getItem(info.position)
    when (item.itemId) {
        R.id.delete_alarm -> {
            // Confirm that the alarm will be deleted.
            AlertDialog.Builder(activity).setTitle("Delete alarm")
                    .setMessage("Delete this alarm?")
                    .setPositiveButton(android.R.string.ok) { d, w -> alarms.delete(alarm) }.setNegativeButton(android.R.string.cancel,  listener: null).show()
            return true
        }

        R.id.enable_alarm -> {
            alarms.enable(alarm, !alarm.isEnabled)
            return true
        }

        R.id.edit_alarm -> {
            uiStore.edit(alarm.id)
            return true
        }

        else -> {
            return super.onContextItemSelected(item)
        }
    }
}
```

Рисунок 3.8 – Код для видалення, активування та редагування будильника.

Ось так в нас виглядає тепер головний екран додатка, коли налаштовано усе необхідне. Тепер саме такий дизайн буде зустрічати користувача при кожному запуску додатка (Рисунок 3.9). Користувач натискаючи на кнопку з намальованими трьома точками зможе попасти в меню налаштування будильника. Реалізацією цього меню ми зайнялись слідом. Для користувача необхідно продумати зручне налаштування кожного будильнику. Обов’язково в ньому має бути присутнім налаштування часу дзвінка будильнику, налаштування днів тижня в які буде спрацьовувати будильник автоматично, можливість змінити мелодію сигналу будильника. Наприклад ця функція знадобиться, якщо людина поставила декілька різних будильників і мала змогу їх розрізняти на слух. Також не мало потрібною функцією можна назвати попередній дзвінок будильника. Це коли користувач хоче, щоб будильник за якийсь час до свого дзвінка зробив попередній дзвінок. Також потрібен підпис для кожного будильника. Наприклад коли користувач захоче зробити нагадування у вигляді будильника. Не завжди звичайні вбудовані в систему нагадування спрацьовують, а дзвінок будильника чути завжди та всюди. Таким буде тепер вікно налаштувань будильника (Рисунок 3.10).

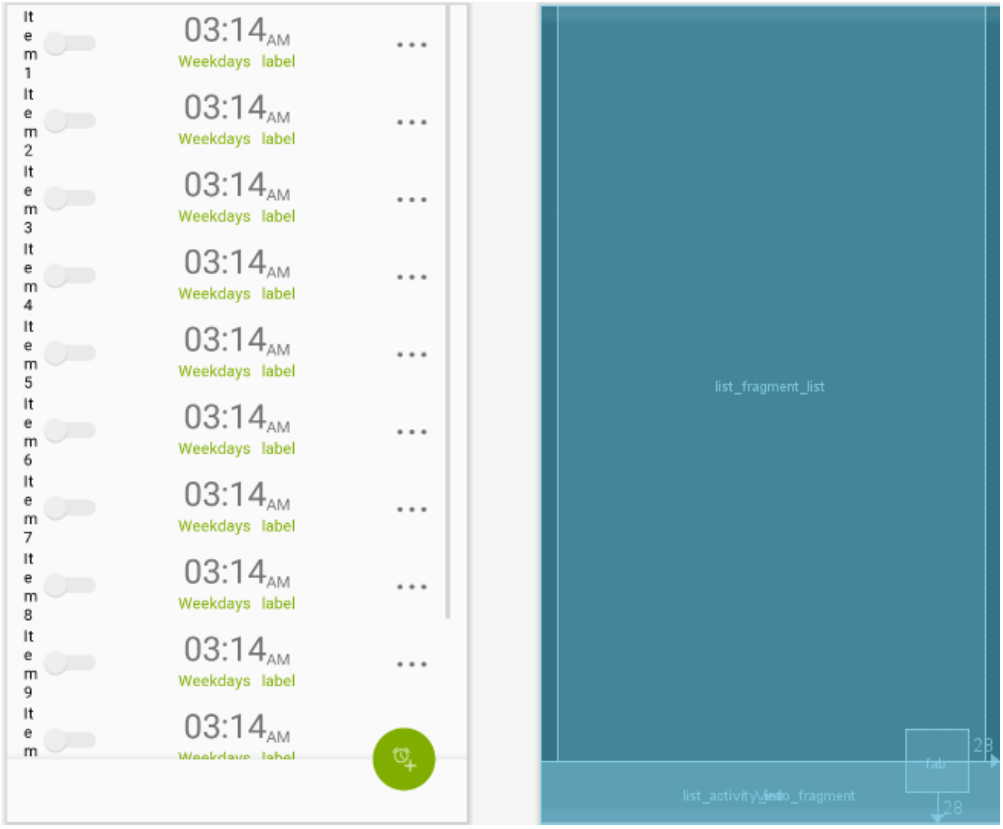


Рисунок 3.9– Головний екран додатку. Відображення на етапі розробки в Android Studio.

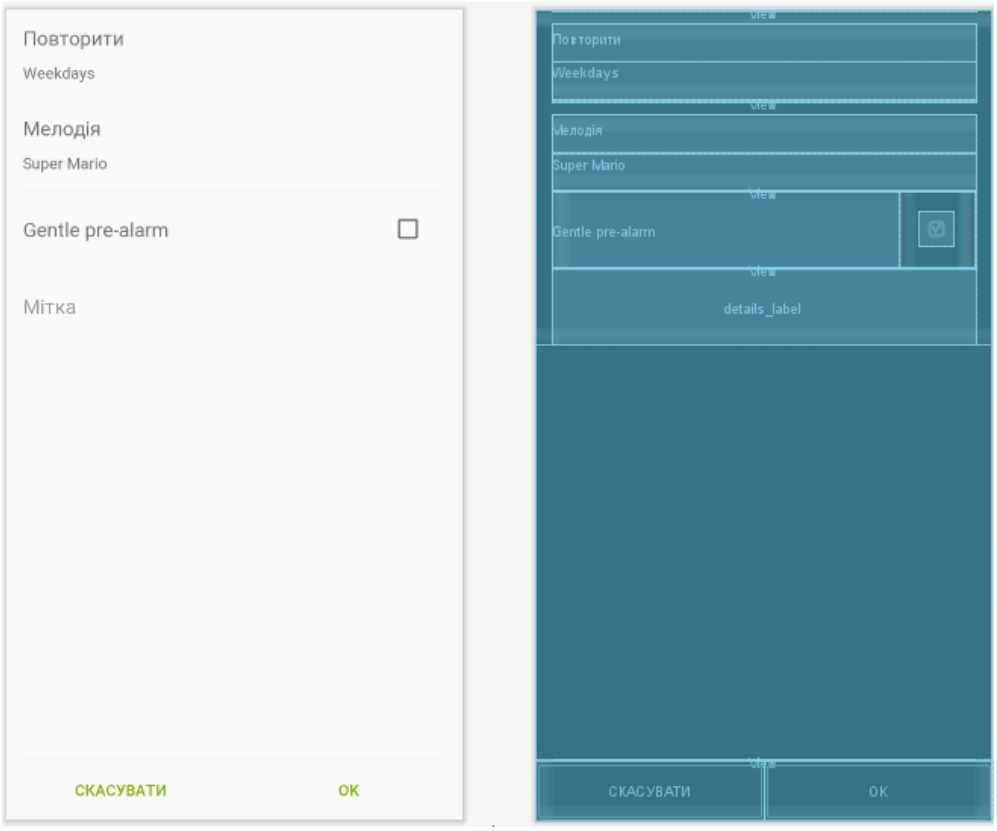


Рисунок 3.10 – Вікно налаштувань будильника

Тепер щоб це все працювало потрібно прописати код для роботи кожного елементу. Розпочавши з налаштування зміни мелодії будильника (Рисунок 3.11), перейшли до налаштування підпису будильника (Рисунок 3.12).

```
mRingtoneRow.setOnClickListener { it: View!
    editor.firstOrError().subscribe { editor ->
        try {
            startActivityForResult(Intent(RingtoneManager.ACTION_RINGTONE_PICKER).apply { this: Intent
                putExtra(RingtoneManager.EXTRA_RINGTONE_EXISTING_URI, editor.alarmtone.ringtoneManagerString())

                putExtra(RingtoneManager.EXTRA_RINGTONE_SHOW_DEFAULT, value: true)
                putExtra(RingtoneManager.EXTRA_RINGTONE_DEFAULT_URI, RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM))

                putExtra(RingtoneManager.EXTRA_RINGTONE_SHOW_SILENT, value: true)
                putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE, RingtoneManager.TYPE_ALARM)
            }, requestCode: 42)
        } catch (e: Exception) {
            Toast.makeText(context, "No Ringtone provider found...", Toast.LENGTH_LONG)
                    .show()
        }
    }
}
```

Рисунок 3.11– Код для налаштування зміни мелодії будильника

```
mLabel.addTextChangedListener(object : TextWatcher {
    override fun afterTextChanged(s: Editable?) {
    }

    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {
    }

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
        editor.take( count: 1)
                .filter { it.label != s.toString() }
                .subscribe { it: AlarmData!
                    modify( reason: "Label") { prev -> prev.copy(label = s.toString(), isEnabled = true) }
                }
                .addToDisposables()
    }
})
```

Рисунок 3.12– Код налаштування підпису будильника.

Тепер потрібно прописати код для роботи налаштування часу, який користувач вибере, також прописати код для попереднього дзвінка будильника (Рисунок 3.13) та налаштування днів тижня (Рисунок 3.14). Попередній дзвінок

повинен бути опціональним. Тобто працювати тільки тоді, коли користувач сам вкаже, що це потрібно.

```
private class PreAlarmFiredState extends AlarmState {
    @Override
    public void enter() {
        broadcastAlarmState(Intents.ALARM_PREALARM_ACTION);
        setAlarm(calculateNextTime(), CalendarType.NORMAL);
    }

    @Override
    protected void onFired() { transitionTo(fired); }

    @Override
    protected void onSnooze() {
        if (getCurrentMessage().obj().isPresent()) {
            //snooze to time with prealarm -> go to snoozed
            transitionTo(snoozed);
        } else {
            transitionTo(preAlarmSnoozed);
        }
    }

    @Override
    public void exit() {
        removeAlarm();
        broadcastAlarmState(Intents.ALARM_DISMISS_ACTION);
    }
}
```

Рисунок 3.13 – Код налаштування попереднього дзвінка будильника

```
fun DaysOfWeek.summary(context: Context): String {
    return toString(context,  showNever: true)
}

fun DaysOfWeek.showDialog(context: Context): Single<DaysOfWeek> {
    return Single.create { emitter ->
        val weekdays = DateFormatSymbols().weekdays
        val entries = arrayOf(weekdays[Calendar.MONDAY], weekdays[Calendar.TUESDAY], weekdays[Calendar.WEDNESDAY],
        var mutableDays = coded
        AlertDialog.Builder(context)
                .setMultiChoiceItems(entries, booleanArray) { _, which, isChecked ->
                    mutableDays = whe
                        isChecked ->
                        else -> mutableDays and (1 shl which).inv()
                    }
                }
                .setPositiveButton(android.R.string.ok) { _, which ->
                    emitter.onSuccess(DaysOfWeek(mutableDays))
                }
                .setOnCancelListener {
                    emitter.onSuccess(DaysOfWeek(mutableDays))
                }
                .create()
                .show()
    }
}
```

val entries: Array<String!>

Рисунок 3.14 – Код налаштування днів тижня та часу будильника.

Для налаштування часу зі сторони користувача, потрібна зручна форма для цього. Користувачу потрібно точно налаштувати години та хвилини. Для його ж зручності ми добавили автоматичні кнопки. Вони зможуть одним натисканням поставити хвилини на 00 або 30 хвилин. Також потрібно продумати те, що в годині немає більше 59 хвилин та те, що відлік починається з 00 хвилин. Користувач також буде мати змогу виправити час, якщо він написав його не вірно, натискаючи на кнопку очистки (Рисунок 3.15).

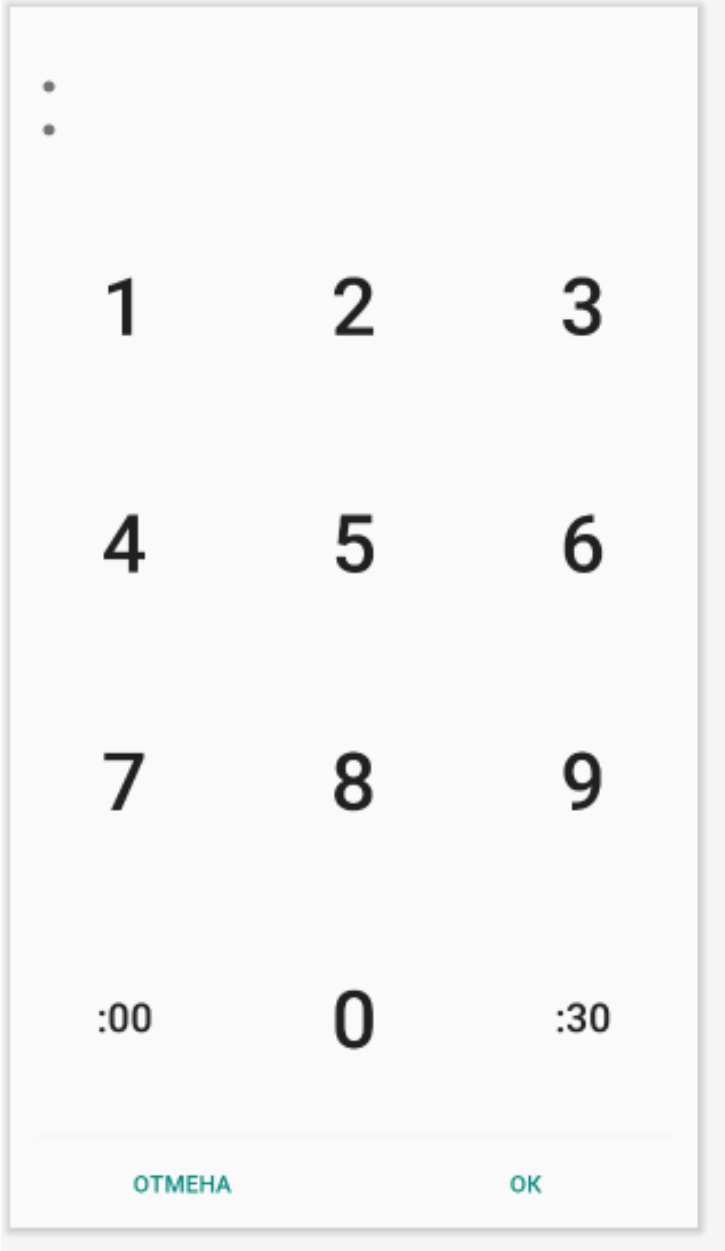


Рисунок 3.15 – Вікно зміни часу будильника

Тепер, щоб це все працювало як задумано, потрібно запрограмувати кожну кнопку на конкретну дію (Рисунок 3.16, Рисунок 3.17, Рисунок 3.18).

```
@Override
protected void onFinishInflate() {
    super.onFinishInflate();

    View v1 = findViewById(R.id.first);
    View v2 = findViewById(R.id.second);
    View v3 = findViewById(R.id.third);
    View v4 = findViewById(R.id.fourth);
    mEnteredTime = (TimerView) findViewById(R.id.timer_time_text);
    mDelete = (ImageButton) findViewById(R.id.delete);
    mDelete.setOnClickListener(this);
    mDelete.setOnLongClickListener((v) -> {
            presenter.reset();
            return true;
    });

    mNumbers[1] = (Button) v1.findViewById(R.id.key_left);
    mNumbers[2] = (Button) v1.findViewById(R.id.key_middle);
    mNumbers[3] = (Button) v1.findViewById(R.id.key_right);

    mNumbers[4] = (Button) v2.findViewById(R.id.key_left);
    mNumbers[5] = (Button) v2.findViewById(R.id.key_middle);
    mNumbers[6] = (Button) v2.findViewById(R.id.key_right);

    mNumbers[7] = (Button) v3.findViewById(R.id.key_left);
    mNumbers[8] = (Button) v3.findViewById(R.id.key_middle);
    mNumbers[9] = (Button) v3.findViewById(R.id.key_right);

    mLeft = (Button) v4.findViewById(R.id.key_left);
    mNumbers[0] = (Button) v4.findViewById(R.id.key_middle);
    mRight = (Button) v4.findViewById(R.id.key_right);

    for (int i = 0; i < 10; i++) {
        mNumbers[i].setOnClickListener(this);
        mNumbers[i].setText(String.format("%d", i));
    }

    mNumbers[0].setTag(TimePickerPresenter.Key.ZERO);
    mNumbers[1].setTag(TimePickerPresenter.Key.ONE);
    mNumbers[2].setTag(TimePickerPresenter.Key.TWO);
    mNumbers[3].setTag(TimePickerPresenter.Key.THREE);
    mNumbers[4].setTag(TimePickerPresenter.Key.FOUR);
    mNumbers[5].setTag(TimePickerPresenter.Key.FIVE);
    mNumbers[6].setTag(TimePickerPresenter.Key.SIX);
```

Рисунок 3.16– Код форми встановлення часу (частина 1).

```
        mLeft.setTag(TimePickerPresenter.Key.LEFT);
        mRight.setTag(TimePickerPresenter.Key.RIGHT);
        mDelete.setTag(TimePickerPresenter.Key.DELETE);

        mLeft.setOnClickListener(this);
        mRight.setOnClickListener(this);

        Resources res = mContext.getResources();
        mAmPmLabel = (TextView) findViewById(R.id.ampm_label);

        if (mIs24HoursMode) {
            mLeft.setText(":00");
            mRight.setText(":30");
            mAmPmLabel.setVisibility(INVISIBLE);
        } else {
            mLeft.setText(AM_PM_STRINGS[0]);
            mRight.setText(AM_PM_STRINGS[1]);
        }

        mLeft.setContentDescription(null);
        mRight.setContentDescription(null);
        views.append(TimePickerPresenter.Key.ONE.ordinal(), mNumbers[1]);
        views.append(TimePickerPresenter.Key.TWO.ordinal(), mNumbers[2]);
        views.append(TimePickerPresenter.Key.THREE.ordinal(), mNumbers[3]);
        views.append(TimePickerPresenter.Key.FOUR.ordinal(), mNumbers[4]);
        views.append(TimePickerPresenter.Key.FIVE.ordinal(), mNumbers[5]);
        views.append(TimePickerPresenter.Key.SIX.ordinal(), mNumbers[6]);
        views.append(TimePickerPresenter.Key.SEVEN.ordinal(), mNumbers[7]);
        views.append(TimePickerPresenter.Key.EIGHT.ordinal(), mNumbers[8]);
        views.append(TimePickerPresenter.Key.NINE.ordinal(), mNumbers[9]);
        views.append(TimePickerPresenter.Key.ZERO.ordinal(), mNumbers[0]);
        views.append(TimePickerPresenter.Key.LEFT.ordinal(), mLeft);
        views.append(TimePickerPresenter.Key.RIGHT.ordinal(), mRight);
        views.append(TimePickerPresenter.Key.DELETE.ordinal(), mDelete);

    }
```

Рисунок 3.17 – Код форми встановлення часу (частина 2).

```
presenter.getState().subscribe((Consumer) (state) → {

        mEnteredTime.setTime(state.getHoursTensDigit(), state.getHoursOnesDigit(), state.getMinutesTensDigit(),
        hours = state.getHours();
        minutes = state.getMinutes();

        for (int i = 0; i < views.size(); i++) {
            views.valueAt(i).setEnabled(false);
        }

        if (state.getAmPm().equals(TimePickerPresenter.AmPm.AM)) {
            mAmPmLabel.setText(AM_PM_STRINGS[0]);
        } else if (state.getAmPm().equals(TimePickerPresenter.AmPm.PM)) {
            mAmPmLabel.setText(AM_PM_STRINGS[1]);
        } else {
            mAmPmLabel.setText(noAmPm);
        }

        for (TimePickerPresenter.Key key : state.getEnabled()) {
            views.get(key.ordinal()).setEnabled(true);
        }
});
```

Рисунок 3.18 – Код форми встановлення часу (частина 3).

Так як персоналізовані налаштування для кожного окремого будильника вже готові, потрібно перейти до створення загальних налаштувань для додатку. Для комфортного користування додатком, користувач має змогу змінити гучність будильника, гучність попереднього будильника та змінити мелодію будильника за замовчуванням (Рисунок 3.19). Також не від'ємною функцією попереднього дзвінка будильника є налаштування часу, коли саме буде спрацьовувати попередній будильник перед основним. Тому реалізували меню з вибором часу. Користувач матиме змогу змінити час дзвінка попереднього будильника на 10 хвилин, 20 хвилин, 30 хвилин, 45 хвилин та за годину до дзвінка основного будильнику. Не треба забувати і про людей, які не люблять різких звуків. Для них реалізували функцію наростаючої гучності. Наростаюча гучність це коли мелодія починає грати спочатку на мінімальній гучності, а потім протягом, в залежності від обраного користувачем, часу стає гучнішою до заданої користувачем гучності у вище згаданому пункті меню. Користувач має вибір, протягом якого часу буде діяти наростання гучності, а саме 10 секунд, 20 секунд, 30 секунд, хвилина та дві хвилини. Якщо користувачу не до вподоби ця функція, він у цьому ж меню має

змогу її відключити (Рисунок 3.20). Там саме для користувачів, які люблять довше поспати та переносити будильники, є зручне меню вибору, через який час знову задзвонити будильнику. В меню є варіанти на 5 хвилин, на 10 хвилин, на 15 хвилин, на 20 хвилин, на 25 хвилин та на півгодини.

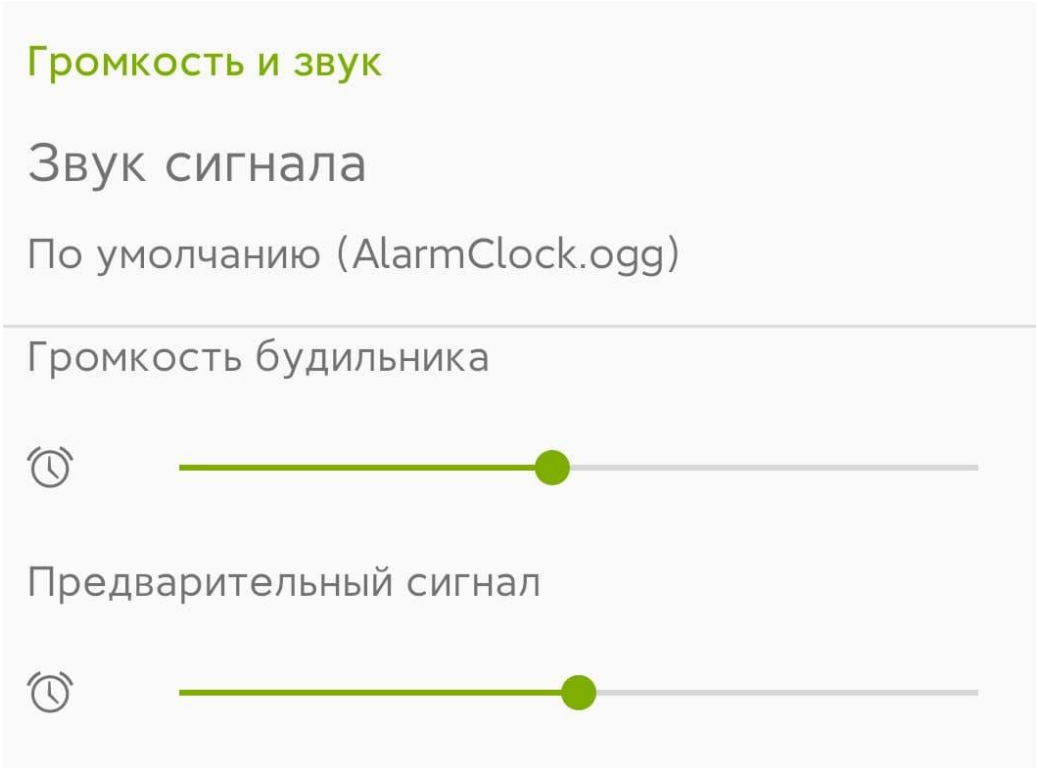


Рисунок 3.19 – Налаштування гучності.

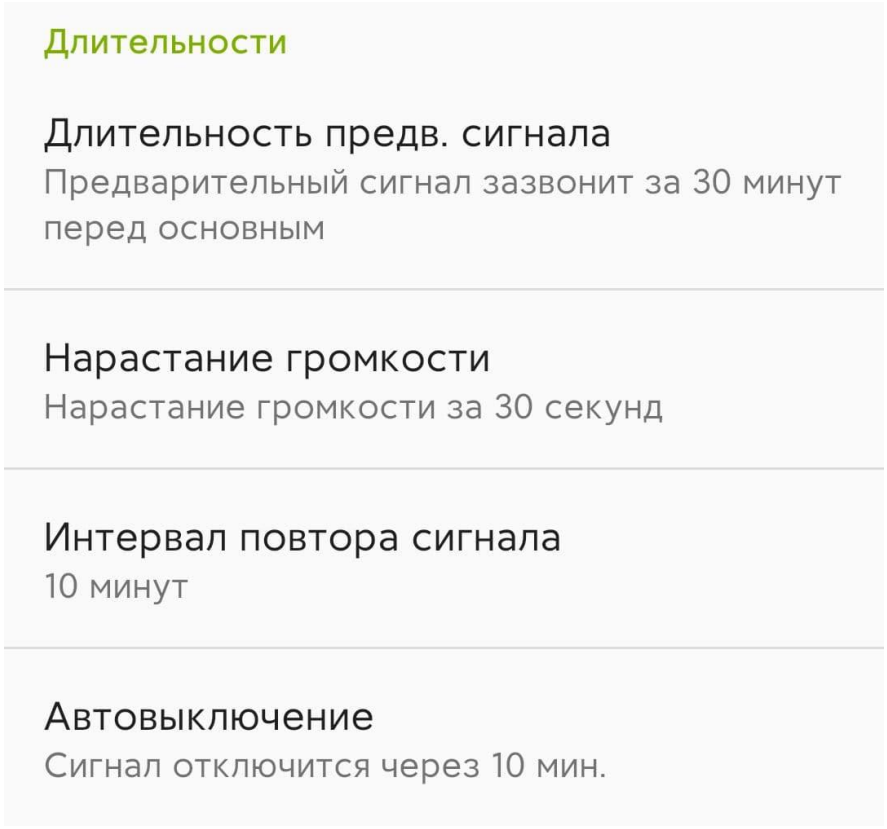


Рисунок 3.20 – Налаштування тривалості.

Для правильної робити цих налаштувань був прописаний код. Кожна функція повинна працювати безперечно точно та без збоїв. Усі ми боїмось проспати ту чи іншу подію. Це може вплинути на подальшу кар’єру в роботі, на успіхи в навчанні чи навіть на особисті стосунки. Налаштування гучності з точки зору коду виглядає так (Рисунок 3.21).

```
private val klaxon: KlaxonPlugin by lazy {
    KlaxonPlugin(
            log = container().logger(),
            playerFactory = {
                PlayerWrapper(
                        context = container().context(),
                        resources = container().context.resources,
                        log = container().logger()
                )
            },
            prealarmVolume = container().rxPrefs().getInteger(KEY_PREALARM_VOLUME, DEFAULT_PREALARM_VOLUME).asObservable(),
            fadeInTimeInMillis = Observable.just( item: 100),
            inCall = Observable.just( item: false),
            scheduler = AndroidSchedulers.mainThread()
    )
}

init {
    layoutResource = R.layout.seekbar_dialog
    // this actually can return null
    this.ringtone = RingtoneManager.getRingtone(context, RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM))
    ringtone?.streamType = AudioManager.STREAM_ALARM
}

override fun onBindView(view: View) {
    super.onBindView(view)

    bindPrealarmSeekBar(view.findViewById<View>(R.id.seekbar_dialog_seekbar_prealarm_volume) as SeekBar)
    bindAudioManagerVolume(view.findViewById<View>(R.id.seekbar_dialog_seekbar_master_volume) as SeekBar)

    view.findViewById<View>(R.id.settings_ringtone).setOnClickListener { it: View!
        context.startActivity(Intent(Settings.ACTION_SOUND_SETTINGS))
    }
    ringtoneSummary = view.findViewById(R.id.settings_ringtone_summary)
    onResume()
}
```

Рисунок 3.21 – Код налаштування гучності

Окрім гучності основного будильника налаштування гучності попереднього дзвінка перед основним також важлива (Рисунок 3.22).

```
private fun bindPrealarmSeekBar(preAlarmSeekBar: SeekBar) {
    val prealarmListener = SeekBarListener()
    preAlarmSeekBar.setOnSeekBarChangeListener(prealarmListener)
    val rxPrefs = container().rxPrefs()
    val log = container().logger()
    val prealarmPreference = rxPrefs.getInteger(KEY_PREALARM_VOLUME, DEFAULT_PREALARM_VOLUME)
    preAlarmSeekBar.max = MAX_PREALARM_VOLUME

    prealarmPreference.asObservable().subscribe { integer -> preAlarmSeekBar.progress = integer!! }

    prealarmListener
            .progressObservable()
            .doOnNext { integer ->
                log.d("Pre-alarm " + integer!!)
                prealarmPreference.set(integer)
                ringtone?.stop()
            }
            .subscribe { it: Int!
                prealarmSampleDisposable.dispose()
                prealarmSampleDisposable = klaxon.go(PluginAlarmData(
                        id = -1,
                        label = "",
                        alarmtone = Alarmtone.Default()
                ), prealarm = true, targetVolume = Observable.just(TargetVolume.FADED_IN))
            }

    prealarmListener
            .progressObservable()
            .debounce( timeout: 3, TimeUnit.SECONDS, AndroidSchedulers.mainThread())
            .subscribe { stopPrealarmSample() }
}
```

Рисунок 3.22 – Код налаштування гучності попереднього дзвінка будильника.

Коли весь основний функціонал готовий, настала пора додати у додаток той самий алгоритм визначення пробудження людини. Логіка робити проста. Коли людина прокидається, та хоч трошки торкається телефону, наприклад подивитись на час, додаток це фіксує, та виводить інформацію у спеціальному меню (Рисунок 3.23).

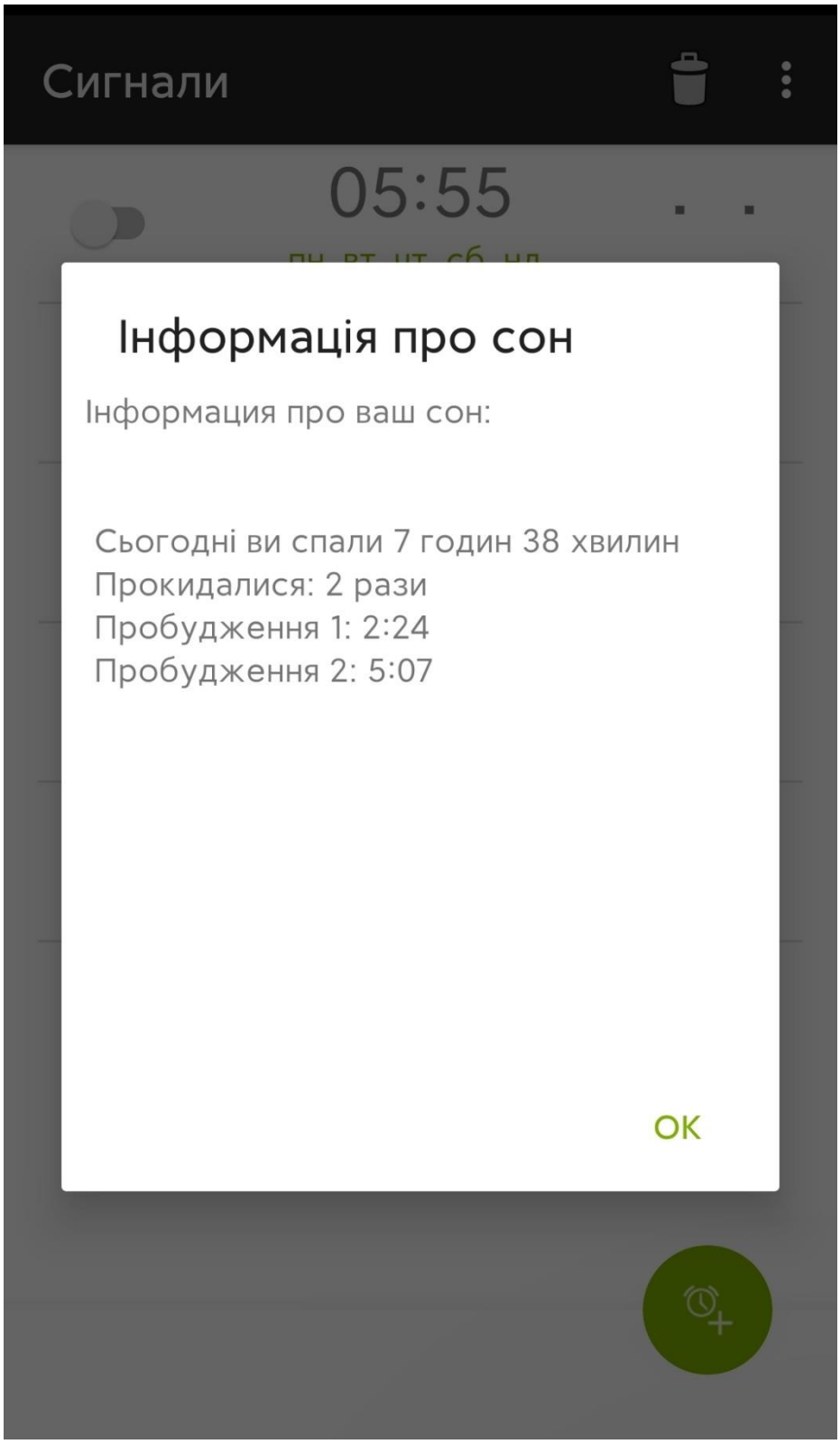


Рисунок 3.23 – Інформація про сон.

Якщо додаток фіксує порушення, то буде виводитись і попередження про те, що необхідно звернутись за допомогою до лікаря (Рисунок 3.24).

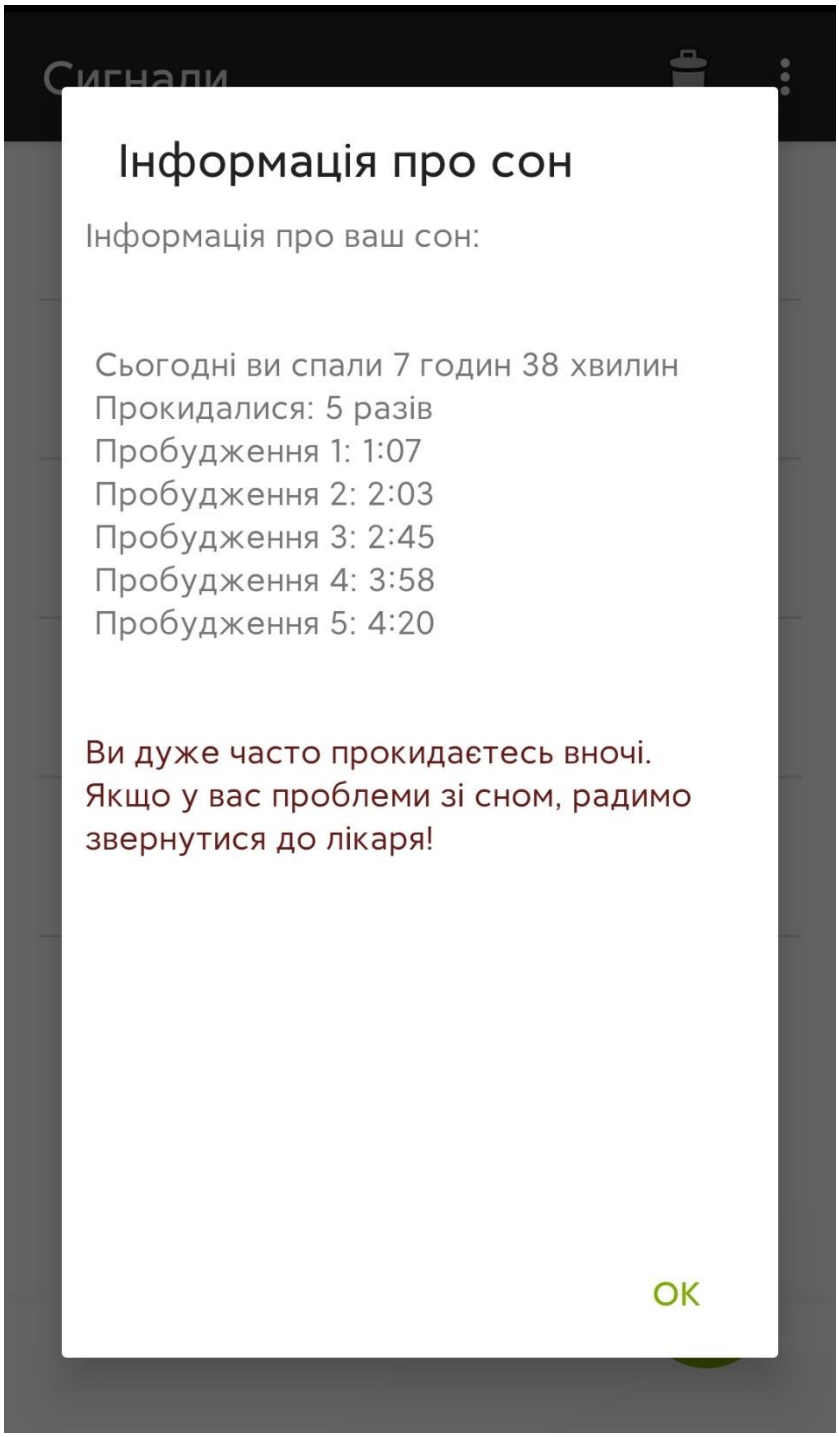


Рисунок 3.24– Інформація про сон, коли виявлені порушення.

Так як інтерфейс без коду не працює, тому під нього був прописаний код, що реалізує усі потрібні функції, користуючись тільки вбудованими в телефон датчиками. (Рисунок 3.25)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.details_fragment_lower);

    screenTimeBroadcastReceiver = new ScreenTimeBroadcastReceiver();
    IntentFilter lockFilter = new IntentFilter();
    lockFilter.addAction(Intent.ACTION_SCREEN_ON);
    lockFilter.addAction(Intent.ACTION_SCREEN_OFF);
    registerReceiver(screenTimeBroadcastReceiver, lockFilter);

    senSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    senAccelerometer = senSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    senMotion = senSensorManager.getDefaultSensor(Sensor.TYPE_MOTION_DETECT);
    senSensorManager.registerListener( listener: this, senAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
}
@Override
public void onSensorChanged(SensorEvent event) {
    Sensor mySensor = event.sensor;

    if (mySensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        float x = event.values[0];
        float y = event.values[1];
        float z = event.values[2];

        long curTime = System.currentTimeMillis();

        if ((curTime - lastUpdate) > 100) {
            long diffTime = (curTime - lastUpdate);
            lastUpdate = curTime;
```

Рисунок 3.25 – Код для реалізації алгоритму визначення пробудження людини.

Висновки

За результатами виконаних теоретичних та практичних досліджень у дипломній роботі розроблено мобільний додаток «Smart Lock» та алгоритм визначення пробудження людини.

Також протягом практичних досліджень було здійснено поглиблення навичок самостійної наукової роботи, розширення наукового світогляду, дослідження проблем практики та вміння пов’язувати їх з обраним теоретичним напрямком дослідження. Також було поглиблено теоретичні знання в сфері розробки мобільних додатків для смартфонів на операційній системі Android, а саме: повністю готовий до роботи мобільний додаток, що працює в парі з алгоритмом визначення пробудження людини.

У цій магістерській роботі представлений алгоритм визначення пробудження людини під назвою ПЧС. Метою ПЧС є точне виявлення та прогнозування часу пробудження та сну людини шляхом видобутку журналу стану екрану смартфона.

Це допоможе людям контролювати свій здоровий сон, та вчасно звернутися до лікаря, якщо виникнуть проблеми. Реалізація ПЧС не залежить від будь-якої актиграфічної або хмарної інфраструктури, що робить її придатною для популярних додатків та забезпечує захист конфіденційності.

Експерименти за даними журналу екранів за 13 днів 5 осіб показали, що ПЧС може ефективно виконувати завдання виявлення без введення додаткових пристроїв або завантаження даних. Я також порівняв його з BES, іншим алгоритмом виявлення сну, і ПЧС перевищує BES за точністю та трудомісткістю. Крім того, моя робота показує, що ПЧС можна застосувати як важливий контекстний компонент для покращення інтелектуального обслуговування на смартфоні.

Мобільний додаток, який також називають мобільним додатком або просто додатком, - це комп’ютерна програма або програмне забезпечення, призначене для роботи на мобільному пристрої, такому як телефон, планшет або годинник.

Додатки спочатку були призначені для сприяння підвищенню продуктивності, наприклад електронної пошти, календаря та контактних баз даних, але суспільний попит на програми спричинив швидке розширення в інших сферах, таких як мобільні ігри, автоматизація роботи на заводі, послуги GPS та локації, відстеження замовлень та квитки. покупки, завдяки чому зараз доступні мільйони програм. Зазвичай програми завантажуються з платформ розповсюдження програм, якими керує власник мобільної операційної системи, наприклад, App Store (iOS) або Google Play Store. Деякі програми безкоштовні, а інші мають ціну, при цьому прибуток розподіляється між творцем програми та платформою розповсюдження. Мобільні програми часто відрізняються від настільних програм, призначених для роботи на настільних комп'ютерах, та веб-програм, які працюють у мобільних веб-браузерах, а не безпосередньо на мобільному пристрої.

Взагалі кажучи, не існує єдиного методу розробки додатків, який був би найкращим для всіх. Все залежить від ваших потреб, бюджету, типу додатка, галузі та багатьох інших факторів.

Якщо ви створюєте ігровий додаток або щось подібне, найкращим варіантом буде власна розробка. Якщо ви робите програму як хобі чи просто для особистого користування, ви, мабуть, зможете відійти від програми для вирізання печива.

Перелік джерел посилань

1. Офіційна презентація Apple iPhone 12 на YouTube [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=KR0g-1hnQPA
2. С. Сеневіратне, А. Сеневіратне, П. Мохапатра та А. Маханті, “Прогнозування властивостей користувачів за допомогою знімка програм, встановлених на смартфоні”, ACM SIGMOBILE Mobile Computing and Communications Review, 2014.
3. А. Андерсен та Р. Карлсен, “Профілювання користувачів за допомогою взаємодії NFC: видобуток інформації про користувачів на основі NFC з мобільних пристроїв та внутрішніх систем”, у Матеріалах 14-го Міжнародного симпозіуму ACM з питань управління мобільністю та бездротового доступу, MobiWac 2016.
4. Сервіс для сну [Електронний ресурс] – Режим доступу до ресурсу: https://pzizz.com/ .
5. Сервіс для сну [Електронний ресурс] – Режим доступу до ресурсу – https://www.sleepcycle.com/
6. Ю. Фан, Ю. Чень, К. Тунг, К. Ву та А. Л. Чен, “Структура для забезпечення профілювання переваг користувачів за допомогою журналів Wi-Fi”, у матеріалах 32-ї Міжнародної конференції з питань інженерії даних IEEE 2016 (ICDE).
7. Р. Баеза-Йейтс, Ф. Сільвестрі, Д. Цзян та Б. Гаррісон, “Прогнозування наступного додатка, яким ви збираєтесь скористатися”, у матеріалах 8-ї Міжнародної конференції ACM з веб-пошуку та видобутку даних, WSDM 2015.
8. Г. Ореллана, С.М. Хелд, П.А. Естевезетал., «Метод класифікації збалансованого сну / неспання на основі актиграфічних даних у підлітків», у матеріалах 36-ї щорічної міжнародної конференції IEEE Товариства з медицини та біології, EMBC 2014.

9. А. Домінгуес, Т. Пайва та Дж. М. Санчес, “Виявлення стану сну та неспання в нічній актиграфії на основі інформації про рух”, IEEE Transactions on Biomedical Engineering.
10. Офіційна документація мови програмування Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/kotlin
11. Г. Ян, Дж. Чен, Х. Тенхунен та Л. Чжен, "Інтелектуальна конструкція електродів для довгострокового моніторингу ЕКГ в домашніх умовах: розробка прототипів з використанням FPAA та FPGA", у матеріалах третьої Міжнародної конференції ICST з широкомасштабних обчислювальних технологій для охорони здоров’я
12. Ж.-К. Мін, А. Доряб, Дж. Віс, С. Аміні, Дж. Ціммерман, та JI Хонг, «Toss 'N' turn: Smartphone as sleep and quality quality detector,» у Матеріалах 32-ї щорічної конференції ACM з людських факторів у Обчислювальні системи, CHI 2014
13. Google API – Режим доступу до ресурсу – https://developers.google.cn/awareness/.
14. В. Коту та Б. Дешпанде, Прогностична аналітика та видобуток даних , 2015.
15. Скільки годин потрібно спати, щоб висипатись та бути здоровим – Режим доступу до ресурсу – https://t1.ua/porady/37570-skilky-hodyn-potribno-spaty-lyudyni-shchob-vysypatys.html.
16. Офіційний сайт для завантаження Java JRE – Режим доступу до ресурсу – https://www.java.com/en/download/.
17. How Much Sleep Do We Really Need – Режим доступу до ресурсу – https://www.sleepfoundation.org/articles/how-much-sleep-do-we-really-need.
18. Платформа розробки – Режим доступу до ресурсу – https://www.android.com/.
19. Магазин додатків – Режим доступу до ресурсу – https://play.google.com/store.
20. Будова смартфонів – Режим доступу до ресурсу – https://hype.tech/@hell-ka/iz-chego-sostoit-smartfon-vzglyad-iznutri-0qewawek.

21. Будова акселерометра – Режим доступу до ресурсу – https://kkite.pnu.edu.ua/mems-%D0%B0%D0%BA%D1%81%D0%B5%D0%BB%D0%B5%D1%80%D0%BE%D0%BC%D0%B5%D1%82%D1%80/
22. Підтримка Android – Режим доступу до ресурсу – https://support.google.com/android/?hl=ru#topic=7313011
23. Нова версія Android – Режим доступу до ресурсу – https://itc.ua/news/google-vypustila-finalnuyu-versiyu-android-11-ona-dostupna-i-dlya-smartfonov-ne-iz-linejki-pixel/

Додаток А

```
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.content.res.Configuration;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.v4.app.FragmentActivity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.TextView;

import com.better.alarm.R;
import com.better.alarm.background.Event;
import com.better.alarm.interfaces.Alarm;
import com.better.alarm.interfaces.IAlarmsManager;
import com.better.alarm.interfaces.Intents;
import com.better.alarm.logger.Logger;
import com.better.alarm.presenter.PickedTime;
import com.better.alarm.presenter.TimePickerDialogFragment;
import com.better.alarm.util.Optional;

import io.reactivex.annotations.NonNull;
import io.reactivex.disposables.Disposable;
import io.reactivex.disposables.Disposables;
import io.reactivex.functions.Consumer;
import io.reactivex.functions.Predicate;

import static com.better.alarm.configuration.AlarmApplication.container;
```

```
import static com.better.alarm.configuration.AlarmApplication.themeHandler;
import static com.better.alarm.configuration.Prefs.LONGCLICK_DISMISS_DEFAULT;
import static com.better.alarm.configuration.Prefs.LONGCLICK_DISMISS_KEY;

/**
 * Alarm Clock alarm alert: pops visible indicator and plays alarm tone. This
 * activity is the full screen version which shows over the lock screen with the
 * wallpaper as the background.
 */
public class AlarmAlertFullScreen extends FragmentActivity {
    protected static final String SCREEN_OFF = "screen_off";

    protected Alarm mAlarm;

    private final IAlarmsManager alarmsManager = container().alarms();
    private final SharedPreferences sp = container().sharedPreferences();

    private boolean longClickToDismiss;

    private Disposable disposableDialog = Disposables.disposed();
    private Disposable subscription;

    @Override
    protected void onCreate(Bundle icicle) {
        setTheme(themeHandler().getIdForName(getClassName()));
        super.onCreate(icicle);

        if (getResources().getBoolean(R.bool.isTablet)) {
            // preserve initial rotation and disable rotation change
            if (getResources().getConfiguration().orientation ==
Configuration.ORIENTATION_PORTRAIT) {
                setRequestedOrientation(getRequestedOrientation());
            } else {
                setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
            }
```

```
            } else {
                setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
            }

            final int id = getIntent().getIntExtra(Intents.EXTRA_ID, -1);
            try {
                mAlarm = alarmsManager.getAlarm(id);

                final Window win = getWindow();
                win.addFlags(WindowManager.LayoutParams.FLAG_SHOW_WHEN_LOCKED);
                // Turn on the screen unless we are being launched from the
                // AlarmAlert
                // subclass as a result of the screen turning off.
                if (!getIntent().getBooleanExtra(SCREEN_OFF, false)) {
                    win.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON
                            | WindowManager.LayoutParams.FLAG_TURN_SCREEN_ON
                            |
WindowManager.LayoutParams.FLAG_ALLOW_LOCK_WHILE_SCREEN_ON);
                }

                updateLayout();

                // Register to get the alarm killed/snooze/dismiss intent.
                subscription = container().store()
                        .getEvents()
                        .filter(new Predicate<Event>() {
                            @Override
                            public boolean test(Event event) throws Exception {
                                return (event instanceof Event.SnoozedEvent && ((Event.SnoozedEvent)
event).getId() == id)
                                        || (event instanceof Event.DismissEvent && ((Event.DismissEvent)
event).getId() == id)
                                        || (event instanceof Event.Autosilenced && ((Event.Autosilenced)
event).getId() == id);
                            }
```

```
            }).subscribe(new Consumer<Event>() {
                @Override
                public void accept(Event event) throws Exception {
                    finish();
                }
            });
    } catch (Exception e) {
        Logger.getDefaultLogger().d("Alarm not found");
    }
}

private void setTitle() {
    final String titleText = mAlarm.getLabelOrDefault();
    setTitle(titleText);
    TextView textView = findViewById(R.id.alarm_alert_label);
    textView.setText(titleText);
}

protected int getLayoutResId() {
    return R.layout.alert_fullscreen;
}

protected String getClassName() {
    return AlarmAlertFullScreen.class.getName();
}

private void updateLayout() {
    LayoutInflater inflater = LayoutInflater.from(this);

    setContentView(inflater.inflate(getLayoutResId(), null));

    /*
     * snooze behavior: pop a snooze confirmation view, kick alarm manager.
     */
    final Button snooze = (Button) findViewById(R.id.alert_button_snooze);
```

```
            snooze.requestFocus();
            snooze.setOnClickListener(new Button.OnClickListener() {
                @Override
                public void onClick(View v) {
                    snoozeIfEnabledInSettings();
                }
            });

            snooze.setOnLongClickListener(new Button.OnLongClickListener() {
                @Override
                public boolean onLongClick(View v) {
                    if (isSnoozeEnabled()) {
                        disposableDialog =
TimePickerDialogFragment.showTimePicker(getSupportFragmentManager())
                                .subscribe(new Consumer<Optional<PickedTime>>() {
                                    @Override
                                    public void accept(@NonNull Optional<PickedTime> picked) {
                                        if (picked.isPresent()) {
                                            mAlarm.snooze(picked.get().getHour(), picked.get().getMinute());
                                        } else {
                                            AlarmAlertFullScreen.this.sendBroadcast(new
Intent(Intents.ACTION_DEMUTE));
                                        }
                                    }
                                });
                        container().store().getEvents().onNext(new Event.MuteEvent());
                        new android.os.Handler().postDelayed(new Runnable() {
                            @Override
                            public void run() {
                                container().store().getEvents().onNext(new Event.DemuteEvent());
                            }
                        }, 10000);
                    }
                    return true;
                }
```

```
    });

    /* dismiss button: close notification */
    final Button dismissButton = (Button) findViewById(R.id.alert_button_dismiss);
    dismissButton.setOnClickListener(new Button.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (longClickToDismiss) {
                dismissButton.setText(getString(R.string.alarm_alert_hold_the_button_text));
            } else {
                dismiss();
            }
        }
    });

    dismissButton.setOnLongClickListener(new Button.OnLongClickListener() {
        @Override
        public boolean onLongClick(View v) {
            dismiss();
            return true;
        }
    });

    /* Set the title from the passed in alarm */
    setTitle();
}

// Attempt to snooze this alert.
private void snoozeIfEnabledInSettings() {
    if (isSnoozeEnabled()) {
        alarmsManager.snooze(mAlarm);
    }
}

// Dismiss the alarm.
```

```
    private void dismiss() {
        alarmsManager.dismiss(mAlarm);

getWindow().addFlags(WindowManager.LayoutParams.FLAG_DISMISS_KEYGUARD);
    }

    private boolean isSnoozeEnabled() {
        return Integer.parseInt(sp.getString("snooze_duration", "-1")) != -1;
    }

    /**
     * this is called when a second alarm is triggered while a previous alert
     * window is still active.
     */
    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);

        Logger.getDefaultLogger().d("AlarmAlert.OnNewIntent()");

        int id = intent.getIntExtra(Intents.EXTRA_ID, -1);
        try {
            mAlarm = alarmsManager.getAlarm(id);
            setTitle();
        } catch (Exception e) {
            Logger.getDefaultLogger().d("Alarm not found");
        }

    }

    @Override
    protected void onResume() {
        super.onResume();
        longClickToDismiss =
PreferenceManager.getDefaultSharedPreferences(this).getBoolean(LONGCLICK_DISMISS_KEY,
```

```
                LONGCLICK_DISMISS_DEFAULT);

        Button snooze = findViewById(R.id.alert_button_snooze);
        View snoozeText = findViewById(R.id.alert_text_snooze);
        if (snooze != null) snooze.setEnabled(isSnoozeEnabled());
        if (snoozeText != null) snoozeText.setEnabled(isSnoozeEnabled());
    }

    @Override
    protected void onPause() {
        super.onPause();
        disposableDialog.dispose();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Logger.getDefaultLogger().d("AlarmAlert.onDestroy()");
        // No longer care about the alarm being killed.
        subscription.dispose();
    }

    @Override
    public void onBackPressed() {
        // Don't allow back to dismiss
    }
}
```

ДОДАТОК Б

```
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.hardware.display.DisplayManager;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.os.Bundle;
import android.util.Log;

import com.better.alarm.R;


public class PreAlarmCore extends Activity implements SensorEventListener{
    private SensorManager senSensorManager;
    private Sensor senAccelerometer;
    private Sensor senMotion;
    private long lastUpdate = 0;
    private float last_x, last_y, last_z;
    private static final int SHAKE_THRESHOLD = 600;
    private ScreenTimeBroadcastReceiver screenTimeBroadcastReceiver;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.details_fragment_lower);
```

```
            screenTimeBroadcastReceiver = new ScreenTimeBroadcastReceiver();
            IntentFilter lockFilter = new IntentFilter();
            lockFilter.addAction(Intent.ACTION_SCREEN_ON);
            lockFilter.addAction(Intent.ACTION_SCREEN_OFF);
            registerReceiver(screenTimeBroadcastReceiver, lockFilter);

            senSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
            senAccelerometer =
senSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
            senMotion = senSensorManager.getDefaultSensor(Sensor.TYPE_MOTION_DETECT);
            senSensorManager.registerListener( this, senAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
        }
        @Override
        public void onSensorChanged(SensorEvent event) {
            Sensor mySensor = event.sensor;

            if (mySensor.getType() == Sensor.TYPE_ACCELEROMETER) {
                float x = event.values[0];
                float y = event.values[1];
                float z = event.values[2];

                long curTime = System.currentTimeMillis();

                if ((curTime - lastUpdate) > 100) {
                    long diffTime = (curTime - lastUpdate);
                    lastUpdate = curTime;

                    float speed = Math.abs(x + y + z - last_x - last_y - last_z)/ diffTime * 10000;

                    if (speed > SHAKE_THRESHOLD) {

                    }

                    last_x = x;
```

```
                last_y = y;
                last_z = z;
            }
        }
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {

    }

    protected void onPause() {
        super.onPause();
        senSensorManager.unregisterListener(this);
    }

    protected void onResume() {
        super.onResume();
        senSensorManager.registerListener(this, senAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
    }

    public class ScreenTimeBroadcastReceiver extends BroadcastReceiver {

        private long startTimer;
        private long endTimer;
        private long screenOnTimeSingle;
        private long screenOnTime;
        private final long TIME_ERROR = 1000;

        public void onReceive(Context context, Intent intent) {
            //Log.i(P.TAG, "ScreenTimeService onReceive");

            if (intent.getAction().equals(Intent.ACTION_SCREEN_ON)) {
                startTimer = System.currentTimeMillis();
```

```
            } else if (intent.getAction().equals(Intent.ACTION_SCREEN_OFF)) {
                endTimer = System.currentTimeMillis();
                screenOnTimeSingle = endTimer - startTimer;

                if (screenOnTimeSingle < TIME_ERROR) {
                    screenOnTime += screenOnTime;
                }

            }
        }
    }
}
```

ДОДАТОК В

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <RelativeLayout

        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="3">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center|center_horizontal"
            android:orientation="vertical">

            <TextView
                android:id="@+id/alarm_alert_label"
                style="?android:attr/buttonBarButtonStyle"
                android:layout_width="match_parent"
                android:layout_height="0px"
                android:layout_weight="1"
                android:clickable="false"
                android:gravity="center|center_horizontal"
                android:paddingTop="8dip"
                android:paddingBottom="8dip"
                android:text="Label"
                android:textColor="@color/colorAccent"
                android:textSize="18sp" />

            <com.better.alarm.view.DigitalClock
```

```
        android:id="@+id/alert_digital_clock"
        android:layout_width="match_parent"
        android:layout_height="0px"
        android:layout_weight="1"
        android:gravity="center_horizontal">

        <TextView
            android:id="@+id/digital_clock_time"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:fontFamily="sans-serif-thin"
            android:textSize="100sp"
            android:textStyle="normal" />

        <TextView
            android:id="@+id/digital_clock_am_pm"
            android:layout_width="40dip"
            android:layout_height="wrap_content"
            android:gravity="bottom|center_horizontal"
            android:textSize="20sp"
            android:textStyle="bold" />
    </com.better.alarm.view.DigitalClock>

    <TextView
        android:id="@+id/alert_text_snooze"
        style="?android:attr/buttonBarButtonStyle"
        android:layout_width="match_parent"
        android:layout_height="0px"
        android:layout_weight="1"
        android:clickable="false"
        android:text="@string/alarm_alert_snooze_text"
        android:textColor="@color/colorAccent"
        android:textSize="24sp" />
</LinearLayout>
```

```
        <Button
            android:id="@+id/alert_button_snooze"
            style="?android:attr/buttonBarButtonStyle"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginLeft="40dp"
            android:layout_marginTop="10dp"
            android:layout_marginRight="40dp"
            android:layout_marginBottom="10dp" />
    </RelativeLayout>

    <View
        android:id="@+id/alert_button_divider"
        android:layout_width="match_parent"
        android:layout_height="1dip"
        android:layout_marginLeft="16dip"
        android:layout_marginRight="16dip"
        android:background="?android:attr/dividerHorizontal" />

    <Button
        android:id="@+id/alert_button_dismiss"
        style="?android:attr/buttonBarButtonStyle"
        android:layout_width="match_parent"
        android:layout_height="0px"
        android:layout_marginLeft="40dp"
        android:layout_marginTop="10dp"
        android:layout_marginRight="40dp"
        android:layout_marginBottom="10dp"
        android:layout_weight="1"
        android:text="@string/alarm_alert_dismiss_text"
        android:textColor="@color/colorAccent"
        android:textSize="24sp" />
</LinearLayout>
```

ДОДАТОК Г

Алгоритм визначення пробудження людини під час сну

Чеснюк М.В..
Науковий керівник – к.т.н.,доцент Медзатий Д.М.
Хмельницький національний університет

Для постачальників послуг смартфонів вкрай важливо визнати характеристики споживачів. Процес розпізнавання цих характеристик зазвичай називають профілем користувача, який забезпечує основу знань для прийняття бізнес-рішень, забезпечує інтелектуальні послуги та забезпечує унікальну конкурентоспроможність. Як основний компонент профілю користувача, час сну може відображати спосіб життя, стан здоров'я та заняття людей. У цій роботі представлений гнучкий алгоритм під назвою BTP (Bedtime Prediction), який призначений для прогнозування часу пробудження та сну за допомогою аналізу стану екрану смартфона. BTP спочатку збирає дані журналу стану екрану смартфона користувача та проводить попередню обробку з низкою допоміжних профілів користувача. Тоді, він виявляє та реєструє час пробудження та сну користувачів за один день шляхом пошуку та поєднання основних періодів гасіння екрану за останні 24 години. Нарешті, BTP передбачає майбутній час сну, узгоджуючи поточну послідовність стану екрана з усіма історичними записами. Застосовуючи BTP, більшість програм, що базуються на нічних та ранкових сценаріях, можуть надавати більш уважні послуги, а не дотримуватися фіксованого часу виконання, наприклад будильника. Експерименти з практичних додатків доводять, що BTP може ефективно прогнозувати час пробудження та час сну без застосування складних алгоритмів машинного навчання або завантаження даних на сервер. а не слідувати фіксованому часу виконання, як будильник.

Процес виявлення та прогнозування АТС може бути придушений багатьма інцидентами. Недостатньо використовувати лише дані журналу стану екрана. У реальних проектах та дослідженнях ми зіткнулися з такими труднощами.

Перший - це ненормальна поведінка. Хтось живе регулярно, але повертається додому дуже пізно або час від часу залишається на вулиці. Цей випадок слід визначити як відхилення в теорії аналізу даних. BTP містить моделі, засновані на знанні шаблонів, тому ці викиди слід виявити та відмовитись.

Друге - несподіване переривання. Коли люди сплять, вхідні дзвінки, отримання повідомлень та сповіщення від програм можуть засвічуватися на екрані, дзвонити або вібрувати, а потім спричиняти пробудження. Також можливо, що люди засинають глибоким сном і ігнорують ці події. Але для даних журналу стану екрану тривалий період гасіння екрану переривається дивною подією.

Третій - вимкнення смартфона. Якщо смартфон вимкнено, стан екрана та допоміжні події відсутні. Деякі люди не люблять режим польоту, вони вимикають смартфон перед сном і запускають його, коли встають. Але гірше, користувач може забути зарядити акумулятор, а потім автоматично вимкнути живлення смартфона. Втрата даних журналу є найбільшою загрозою та ризиком ВТР. На малюнку 3 показано стан екрану випадку вимкнення.

Три вищезазначені труднощі згадуються при виявленні, а четверта стосується передбачення. Зважаючи на різний життєвий досвід, норми часу неспання та сну між людьми пристосовані.

Більшість людей працює в офісі або на заводі, тобто в стабільному цільовому місці. Вони показують ідентифікований час пробудження та розподіл сну протягом робочих днів. Але багато з них - вихідні у вільний стиль. Інші не працюють з понеділка по п'ятницю. Через свої професійні особливості вони демонструють дивні розподіли, такі як ходити на роботу кожні два дні чи три дні роботи та один день відпочинку. Це призводить до великої помилки при прогнозуванні за допомогою простих моделей часових рядів або спробі описати ці закономірності за тижнями та вихідними. Окрім труднощів, ми можемо використати дві переваги для виправлення та вдосконалення нашої моделі. Одне - подія будильника. Для більшості людей вони ходять на роботу в будні. Момент дзвінка будильника - це також час пробудження. Це правило має дуже високу впевненість, і ми можемо перевірити подію, що активується прискорювачем, щоб підтвердити це. Інша подія - активована акселерометром. Коли активовано акселерометр, ми знаємо, що смартфон переміщується і користувач прокинувся. Ми долаємо та зменшуємо ефект чотирьох зазначених вище труднощів за допомогою наступних рішень.

1. Аномальна поведінка. ВТР припускає, що люди сплять вдома. Він відмовився від послідовностей не вдома, щоб уникнути впливу ненормальної поведінки.

2. Несподіване переривання . Порушення сну вирішуються шляхом злиття суміжних періодів стану загасання екрану. Тут ми визначаємо центральну точку сну, приблизно 3 години, найсонніший час доби. Коли людей прокидає подія переривання біля центральної точки сну, вони, швидше за все, залишаться спати через кілька хвилин. Але коли вони менш сонні, наприклад о 5:00, люди можуть вставати по справах.

3. Вимкнення смартфона . Для користувачів, які регулярно та розумно вмикають і вимикають, ВТР розглядає події ввімкнення та вимкнення як час пробудження та час сну. Але ми трактуємо нерегулярне вимкнення живлення як відхилення і відмовляємось від нього.

4. Індивідуальні правила між людьми. Раніше ми вважали, що прогнозування поведінки користувачів має базуватися на робочих днях і вихідних днях, але це заплутано в налаштованих правилах робочого дня, що

призводить до більшої помилки. Потім ми змінили нашу стратегію та порівняли поточну послідовність стану екрана з іншими послідовностями кожного минулого дня. Найбільш схожі послідовності представляють однакову модель поведінки. Оскільки прокидання та лягання спати - це початок і кінець цілоденних занять, вони природно відображаються у цій подібній системі моделей.

Підводячи підсумок, ми ефективно покращуємо точність та доступність послідовності стану екрану при виявленні та прогнозуванні часу пробудження та сну, представляючи низку допоміжних подій. Основний процес BTP показаний на малюнку 1 . BTP спочатку збирає дані журналу стану екрану користувача та інші допоміжні події, а потім здійснює пошук та підключення періодів гасіння головного екрану, щоб отримати щоденний результат виявлення. Тоді прогнозування здійснюється шляхом узгодження поточної послідовності стану екрана з історичними послідовностями, такими як K-Найближчий сусід.

Рисунок 1 - Потік обробки BTP.

Тепер дійшла черга до структури даних. По-перше, ми визначаємо структуру даних. Дані журналу містять інформацію як про стан екрану, так і про допоміжний фактор. Подія складається з дати, моменту та події. Різні події відрізняються кодами подій. Тут код SON являє собою загоряння екрану. Для першого завдання, визначення часу пробудження та сну, ми вводимо набір даних журналу з усіма типами подій зі смартфона протягом багатьох днів.

BTP використовує (1), щоб визначити, чи можна цей період об'єднати з сусідніми періодами; тобто, якщо тимчасовий інтервал між ними коротше ніж T, ми з'єднаємося дві суміжних послідовностей , як один. T є поріг, який визначає, слід поєднати дві послідовності, l використовуються для настройки абсолютного значення, t знаходиться в декількох хвилинах від 12:00 до середини періоду, c знаходиться в декількох хвилинах від 12:00 до часу , що люди мають глибокий сон, і off це значення зміщення.

$$T = 1 \cdot \frac{1}{1+e^{|t-c|/60-off}} \quad (1)$$

У цій роботі представлений алгоритм передбачення сну під назвою BTP. Метою BTP є точне виявлення та прогнозування часу пробудження та сну людини шляхом видобутку журналу стану екрану смартфона. Реалізація BTP не залежить від будь-якої актиграфічної або хмарної інфраструктури, що

робить її придатною для популярних додатків та забезпечує захист конфіденційності.

Перелік посилань

1. С. Сеневіратне, А. Сеневіратне, П. Мохапатра та А. Маханті, "Прогнозування властивостей користувачів за допомогою знімка програм, встановлених на смартфоні", ACM SIGMOBILE Mobile Computing and Communications Review, 2014.

2. А. Андерсен та Р. Карлсен, "Профілювання користувачів за допомогою взаємодії NFC: видобуток інформації про користувачів на основі NFC з мобільних пристроїв та внутрішніх систем", у Матеріалах 14-го Міжнародного симпозіуму ACM з питань управління мобільністю та бездротового доступу, MobiWac 2016.

3. Ю. Фан, Ю. Чень, К. Тунг, К. Ву та А. Л. Чен, "Структура для забезпечення профілювання переваг користувачів за допомогою журналів Wi-Fi", у матеріалах 32-ї Міжнародної конференції з питань інженерії даних IEEE 2016 (ICDE)..

4. G. Orellana, CM Held, PA Estevez et al., «Метод класифікації збалансованого сну / неспання на основі актиграфічних даних у підлітків», у матеріалах 36-ї щорічної міжнародної конференції IEEE Товариства з медицини та біології, EMBC 2014.

Розробка мобільного додатка «Smart Clock»

Науковий керівник: К. Т. Н. Доцент Медзатий Дмитро Миколайович

Виконав: Студент групи ПМм-19-1, Чеснюк Максим

Об’єкт, предмет дослідження

- Об’єктом дослідження є розробка мобільного додатку-будильника.
- Предметом дослідження є алгоритм визначення пробудження людини за допомогою вбудованих датчиків в смартфонах на операційній системі «Android».

Мета, завдання

- Метою дипломної магістерської роботи є розробка мобільного додатку з алгоритмом визначення пробудження людини за допомогою смартфону.

Для досягнення поставленої мети необхідно було вирішити такі завдання:

1. Аналіз особливостей програмування для мобільних додатків.
2. Дослідження моделі поведінки алгоритму.
3. Розробка мобільного додатку та інтеграція алгоритму в нього.
4. Розробка алгоритму визначення пробудження людини.

Аналіз особливостей програмування для мобільних додатків.

Говорячи про «розробку додатків», ми зазвичай говоримо про розробку для мобільних пристроїв - включаючи смартфони та планшети.

У всьому світі мобільні пристрої більшості людей працюють на iOS та Android.

Станом на 2019 рік Android контролює близько 88% ринку мобільних пристроїв у всьому світі, а більшість решти належить Apple з їхніми пристроями iPhone та iPad. Кількість користувачів Android зросла з 1,8 мільярда пристроїв у вересні 2015 року до понад 2,7 мільярда сьогодні. Оскільки все більше і більше світового населення виходить в Інтернет протягом наступного десятиліття, Android буде продовжувати зростати.

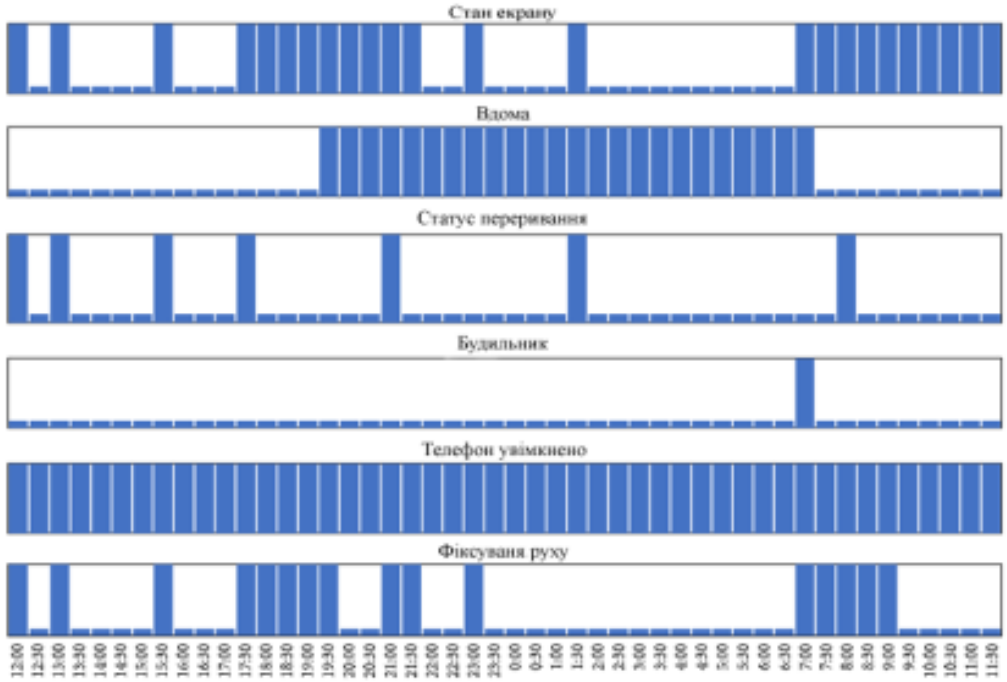
Дослідження проблеми

Проблеми зі сном часто розглядаються як симптом психічних захворювань, а не як причинний фактор. Однак все більша кількість доказів свідчить про те, що вони є і причиною, і симптомом психічних захворювань. Безсоння є важливим провісником великого депресивного розладу ; мета-аналіз 170000 людей показав, що безсоння на початку періоду дослідження показало більш ніж дворазове збільшення ризику для великого депресивного розладу. Деякі дослідження також вказують на кореляцію між безсонням та тривогою, посттравматичним стресовим розладом та суїцидом. Порушення сну може збільшити ризик розвитку психозу та погіршити тяжкість психотичних епізодів.

Прогнозування часу сну

Збір та попередня обробка даних

Враховуючи, що в добі 1440 хвилин, ми ділимо різні події на шість послідовностей довжиною 1440 хвилин. Для стану екрану 0 означає вимкнений екран, а 1 - загоряння екрану. Для стану екрану та стану живлення ми відібрали першу секунду стану хвилини. Однак для іншої послідовності, якщо якась подія відбулася за хвилину, статус буде 1. Рисунок демонструє графік роботи.

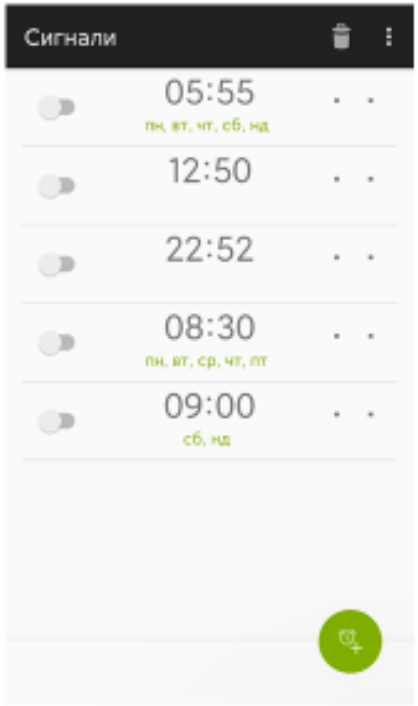


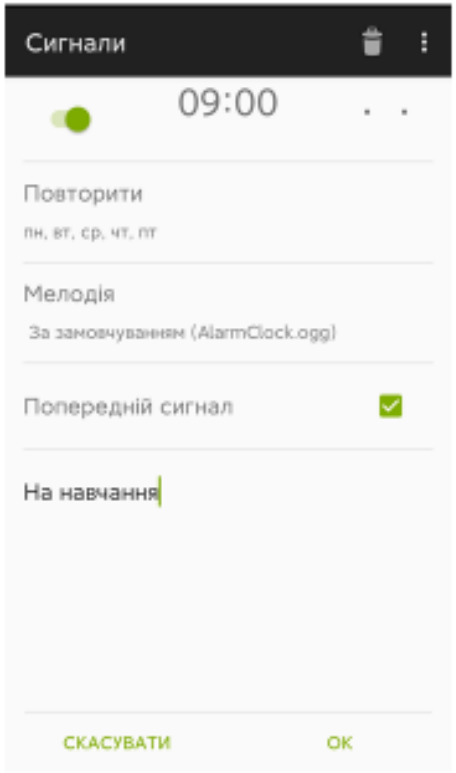
Формула виявлення пробудження

$$T=1\cdot 1/(1+e^{(|t-c|/60-off)})$$

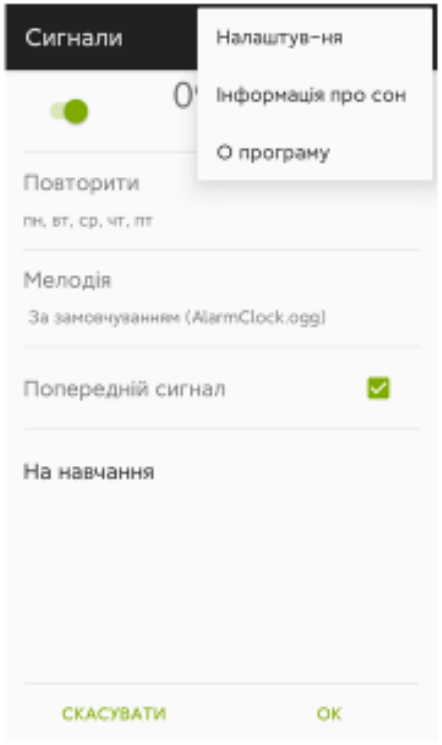
де T – поріг;
l - використовуються для настройки абсолютного значення;
t - знаходиться в декількох хвилинах від 12:00 до середини періоду;
c - знаходиться в декількох хвилинах від 12:00 до часу, що люди мають глибокий сон
off - значення зміщення.

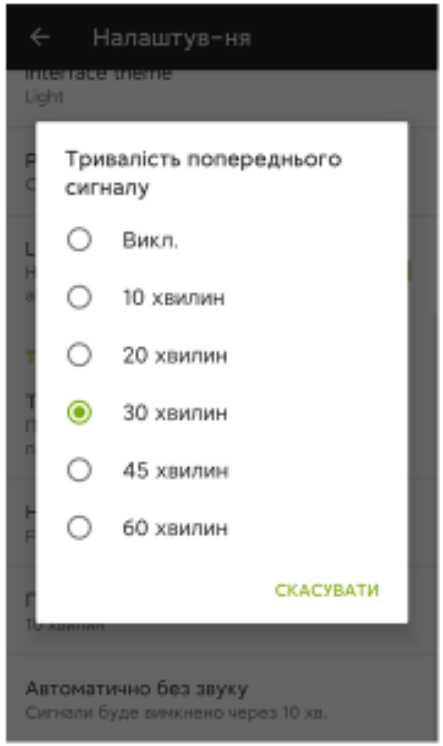
Розробка додатку

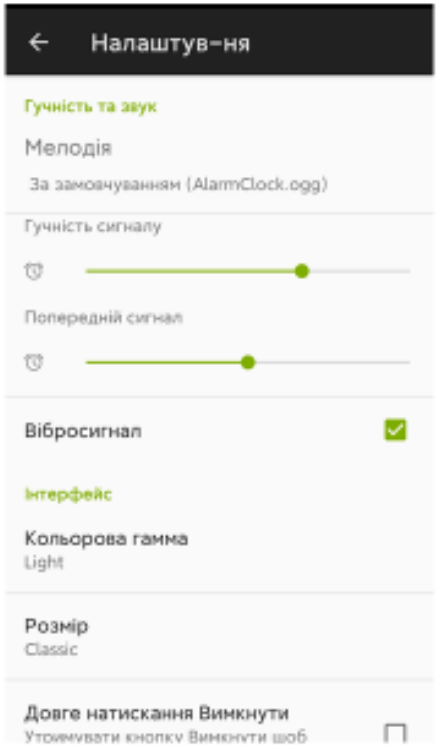




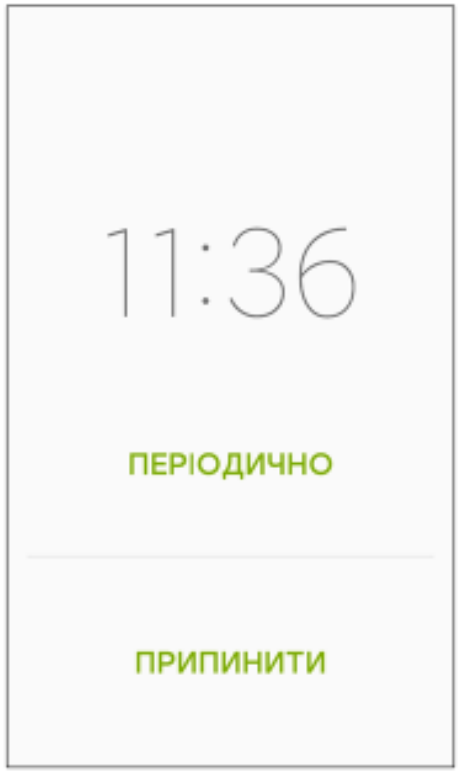
Розробка додатку

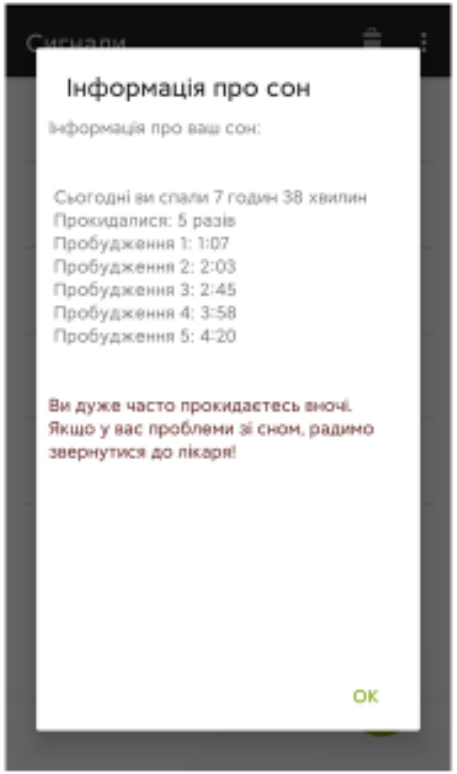


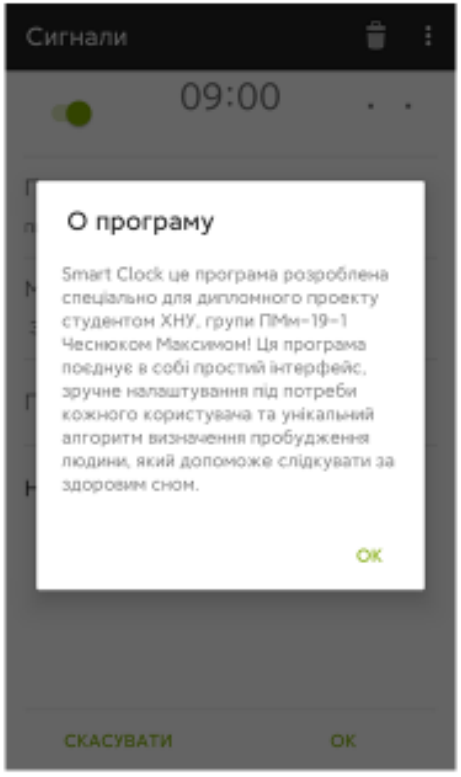




Розробка додатку







Висновки

За результатами виконаних теоретичних та практичних досліджень у дипломній роботі розроблено мобільний додаток «Smart Lock» та алгоритм визначення пробудження людини .

У цій магістерській роботі представлений алгоритм визначення пробудження людини під назвою ПЧС. Метою ПЧС є точне виявлення часу пробудження та сну людини шляхом видобутку журналу стану датчиків смартфона.

Експерименти за даними журналу екранів за 13 днів 5 осіб показали, що ПЧС може ефективно виконувати завдання виявлення без введення додаткових пристроїв або завантаження даних.

За темою дослідження опублікоано тези Чеснюк М.В.
Алгоритм визначення пробудження людини під час сну //
«Інтелектуальний потенціал – 2020» - збірник наукових праць молодих науковців і студентів - Хмельницький : ПВНЗ УЕП, 2020. – Частина 1.

UNICHECK

Имя пользователя:
Kafedra TMIT KhNU

ID проверки:
1005326328

Дата проверки:
02.12.2020 10:58:08 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
02.12.2020 11:52:29 EET

ID пользователя:
100005657

Название файла: Чеснюк_ПМм-19-1

Количество страниц: 86 Количество слов: 13328 Количество символов: 95769 Размер файла: 2.63 MB ID файла: 100544921

Обнаружены модификации текста (могут влиять на процент совпадений)

1.43%
Совпадения

Наибольшее совпадение: 0.65% с источником из Библиотеки (ID файла: 1005444316)

0.94% Источники из Интернета | 150 ..... Страница 88

0.74% Источники из Библиотеки | 30 ..... Страница 88

0.73% Цитат

Цитаты | 8 ..... Страница 89

Не найдено ни одной ссылки

0% Исключений

Нет исключенных источников

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Подозрительное форматирование | 17 страниц

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 1.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 8%**

ID: 81302 Название: Розробка мобільного додатка «Smart Clock» Добавлено в БД: 2020-11-25 Авторы: Чеснюк Максим Валерійович Руководители: Медзатий Дмитро Миколайович Консультанты: Оппоненты:	Документ		Суммарное совпадение по Базе Данных	
	Символы	Лексемы	Символы	Лексемы
	90691	813	1738 (2%)	26 (3%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

освітнього ступеня «магістр»

Магістр Чеснюк Максим Валерійович

Тема Розробка мобільного додатку «Smart Clock»

Спеціальність 113 «Прикладна математика»
спеціалізація «Математика та статистика»

Обсяг дипломної роботи освітнього ступеня «магістр»:

кількість листів креслень _____; кількість сторінок записки 105

1. Короткий зміст ДР та прийнятих рішень В рамках магістерської роботи проведено аналіз моделей, методів і алгоритмів виявлення пробудження людини. Розроблено власний алгоритм виявлення пробудження людини. На його основі створено власний додаток-будильник, який виявляє пробудження людини та у випадку порушення сну повідомляє про це.

2. Висновок про відповідність ДР дипломному завданню Дипломна робота освітнього ступеня «магістр» у повній мірі відповідає поставленому завданню як в теоретичній, так і в практичній частині дипломної роботи.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі обгрунтовується актуальність теми роботи, дається аналіз досліджуваної проблеми і обгрунтовується застосовуваний підхід до її вирішення, формулюються цілі і завдання дослідження, описується наукова новизна і практична значимість отриманих результатів. У першому розділі якісно та в повній мірі проаналізовано сучасний стан ринку мобільних операційних систем, засобів розробки мобільних додатків та необхідність створення алгоритму визначення пробудження людини. Наступні розділи присвячені розробці моделі алгоритму визначення пробудження людини, методу виявлення. Розглянуто питання застосування розробленого методу. Розроблене програмне забезпечення для мобільних телефонів на операційній системі Android.

4. Позитивні сторони проекту Дипломна робота містить ряд інноваційних рішень, зокрема, розроблено метод визначення пробудження людини на основі вбудованих датчиків в усі сучасні телефони на операційній системі Android, що дозволить слідкувати за здоровим сном, та в разі виявлення відхилень, повідомить про це користувача

5. Негативні сторони проекту Розробка представленого алгоритму дозволяє досягти мети, тобто допомагати слідкувати за сном людині, лише після декількох ночей проведених з алгоритмом на базі додатка «Smart Clock», щоб алгоритм пристосувався до поведінки людини.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення виконане відповідно до теми дипломної роботи з дотриманням стандартів. В загальному графічне оформлення виконане на достатньому рівні. Пояснювальна записка відповідає нормам для її оформлення.

7. Відгук про роботу в цілому В загальному дипломна робота заслуговує позитивної оцінки. Весь матеріал дипломної роботи структурований, чіткий та послідовний. Усі розділи роботи послідовні та логічні, що дозволяє чітко розуміти викладений матеріал в рамках тематики дипломної роботи. Графічний матеріал дозволяє наочно побачити доцільність та ефективність рішень, які були прийняті за основу для досягнення поставленої задачі.

8. Інші зауваження.

9. Оцінка дипломної роботи Розглянувши позитивні та негативні сторони представленої дипломної роботи, можна зробити висновок, що вона заслуговує оцінку «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Гурман Іван Васильович, доцент кафедри інженерії програмного забезпечення.

« 02 » грудня 2020 . Гурман І.В. (підпис)

ХНУ. Зам. 54. тир. 6000. 2007

Завідувачу кафедри ТМІТ
д-р.техн.наук Підченку С.К.

Чесюк Максим Валерійович
ПІБ здобувача вищої освіти

ФПКТС, 2 курсу, групи ПМм-19-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіатоповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів(Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

25.11.2020
дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ ТЕЛЕКОМУНІКАЦІЙ, МЕДІЙНИХ ТА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Розробка мобільного додатку «Smart Clock»

Автор: Чеснюк Максим Валерійович

Спеціальність: 113 – прикладна математика

Освітня програма: освітньо-професійна

Науковий керівник: Медзатий Дмитро Миколайович, к.т.н доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	+
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розмішені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
2) більшість джерел запозичення дублюють одне одного;

4.12.2020р
Дата

Підпис