Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Зюзіна Олега Сергійовича*
(ПІБ)

академічної групи *122-19-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка мобільного додатку для відстеження амбулаторного лікування пацієнта за допомогою мови програмування Python та фреймворку Kivy*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Спірінцев В.В.*			
розділів:				
спеціальний	*доц. Спірінцев В.В.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-19-2* (група) *Зюзіна Олега Сергійовича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка мобільного додатку для відстеження амбулаторного лікування пацієнта за допомогою мови програмування Python та фреймворку Kivy*

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проектно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *доц. Спірінцев В.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Зюзін О.С.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 79 с., 11 рис., 3 дод., 27 джерела.

Об’єкт розробки: мобільний додаток для збереження та відображення інформації про лікування пацієнта, яку він вводить, та відстеження цього лікування шляхом керованого виводу повідомлень.

Мета кваліфікаційної роботи полягає в розробці мобільного додатку з використанням мови програмування Python та фреймворку Kivy для відстеження лікування пацієнта, який сприятиме поліпшенню якості медичної допомоги, співпраці між лікарями та пацієнтами, а також забезпеченню контролю над лікуванням та дотриманням рекомендацій лікаря.

У вступі розглянуті первині мотиви для створення додатку, коротко проаналізована предмета область, наведено обґрунтування актуальності розробки та поставлені певні задачі, у зв’язку з якими буде проводитись розробка.

У першому розділі наведені та розглянуті предметна галузь, галузь призначення та підстави для розробки додатку, було поставлено завдання на кваліфікаційну роботу та розглянуто вимоги до технічних та програмних засобів, до інформаційної безпеки та програмної сумісності.

У другому розділі було визначено програмне та технічне забезпечення, потрібне для роботи з програмою. Також наведене функціональне призначення, зазначено про математичні методи, описані технології, що використовувались для розробки програми, та описана сама розроблена система.

В економічному розділі за допомогою формул та розрахунків, зроблених за ними, було обчислено трудомісткість, витрати та час на розробку програмного продукту.

Допомога користувачу щодо нагадування про вживання необхідних заходів для відновлення здоров'я, занесення цих даних та забезпечення їх цілісності зумовлює практичне значення розробки додатку.

Актуальність розробленої програми визначає відсутність єдиного, прийнятого на державному рівні, додатку, що оцифровує сучасний спосіб збереження списку необхідних пацієнту заходів для відновлення його здоров’я.

У майбутній розробці проекту рекомендується розширити функціонал, покращити візуальне представлення (інтерфейс) та оптимізувати існуючий функціонал.

Список ключових слів: ПРОДУКТ, PYTHON, KIVY, WORA, ТЕХНОЛОГІЯ, ГРАФІЧНИЙ ІНТЕРФЕЙС, МОВА ПРОГРАМУВАННЯ, БІБЛІОТЕКА, МОДУЛЬ.

ABSTRACT

Explanatory note: 79 p., 11 figures, 3 appendixes, 27 sources.

Object of development: a mobile application for storing and displaying information about the patient's treatment, which he enters, and tracking this treatment through controlled output of messages.

The purpose of the qualification work is to develop a mobile application using the Python programming language and the Kivy framework to track patient treatment, which will help improve the quality of medical care, cooperation between doctors and patients, as well as ensure control over treatment and compliance with doctor's recommendations.

The introduction discusses the primary motivations for creating the application, briefly analyzes the subject area, provides a justification for the relevance of the development, and sets out certain tasks in connection with which the development will be carried out.

In the first section, the subject area, purpose area and grounds for developing the application are presented and considered, the task for qualification work is set and the requirements for hardware and software, information security and software compatibility are considered.

In the second section, the software and hardware required to work with the application were identified. The functional purpose is also given, mathematical methods are mentioned, technologies used to develop the program are described, and the developed system itself is described.

In the economic section, the labor intensity, costs, and time for developing the software product were calculated using formulas and calculations based on them.

Assisting the user in reminding them to take the necessary measures to restore their health, recording this data, and ensuring its integrity is the practical significance of developing the application.

The relevance of the developed application is determined by the absence of a single application adopted at the state level that digitizes a modern way to keep a list of measures necessary for a patient to restore his or her health.

In the future development of the project, it is recommended to expand the functionality, improve the visual representation (interface) and optimize the existing functionality.

List of keywords: PRODUCT, PYTHON, KIVY, WORA, TECHNOLOGY, GRAPHICAL INTERFACE, PROGRAMMING LANGUAGE, LIBRARY, MODULE.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

WORA – Write Once Run Everywhere;

HTML – Hypertext Markup Language;

Wi-Fi – Wireless Fidelity;

OpenGL - Open Graphics Library;

CGI – Computer-Generated Imagery;

SQL – Structured Query Language;

JDBC - Java Data Base Connectivity;

GNU – GNU’s not UNIX;

JNI – Java Native Interface;

JVM – Java Virtual Machine;

API - Application Programming Interface;

OpenSSL – Open Secure Socket Layer;

JSON - JavaScript Object Notation.

ЗМІСТ

ВСТУП

Сучасна медицина активно використовує мобільні додатки, оскільки вони мають значний потенціал у полі охорони здоров'я. Проте, для досягнення ще більшого успіху, необхідно поліпшити якість та зручність цих додатків. Можна зосередитися на вдосконаленні функцій і забезпеченні комфорту для користувачів. Медичні додатки повинні бути надійними, ефективними і зручними, щоб сприяти ширшому прийняттю і застосуванню цих технологій в медичній практиці.

Наприклад, дослідження показали, що використання мобільних додатків для відстежування фізичної активності збільшує мотивацію користувачів до здорового способу життя. Крім того, аналізи показали, що медичні додатки, які надають інформацію про симптоми та лікування різних захворювань, забезпечують швидше діагностування і поліпшують результати лікування пацієнтів.

Мета кваліфікаційної роботи полягає в розробці мобільного додатку для відстеження лікування пацієнта, який сприятиме поліпшенню якості медичної допомоги, співпраці між лікарями та пацієнтами, а також забезпеченню контролю над лікуванням та дотриманням рекомендацій лікаря.

Цей додаток спрямований на використання у медичній галузі, зокрема в системах електронного здоров'я.

Розроблюючи додаток, автор сподівається надати поштовх для молодих або досвідчених спеціалістів у напрямку розробки додатку, що зможе стати стандартом для забезпечення якісного, ефективного та надійного лікування.

Наразі є відсутнім єдиний додаток, який був би розроблений з метою покращення спілкування між пацієнтом та лікарем та був прийнятим як єдина заміна дискомфортної та неякісної паперової версії рішення цього питання на державному рівні. Однак, у деяких лікарнях вже практикуються такі додатки, десктопні чи мобільні, і явно покращують умови надання медичних послуг, а

також покращують зберігання даних про лікарів та пацієнтів. Цим зумовлюється актуальність розробки даного додатку.

Для досягнення визначеної мети необхідне вирішення наступних завдань:

- аналізувати предметну галузь та знайти ймовірні аналоги додатку;
- визначити призначення додатку та галузь його застосування;
- розробити план розробки додатку та його структуру;
- визначити функціонал додатку;
- визначити технології, потрібні для створення додатку;
- проаналізувати вимоги до функціональних, технічних та програмних характеристик;
- опираючись на обрані технології, розробити архітектуру програми;
- розробити додаток;

Розроблений додаток повинен працювати на мобільних пристроях, які задовольняють рекомендовані технічні та програмні вимоги.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Сьогодні мобільні додатки мають велику популярність, оскільки вони пропонують багато переваг для бізнесу та полегшують наші повсякденні завдання. Вони покращують комунікацію і розширюють можливості користувачів. Мобільні додатки важливі як для користувачів, так і для компаній. Вони дозволяють скористатися різноманітними послугами та отримати необхідну інформацію в будь-який час і місце. Вони спрощують життя і відкривають нові можливості.

Існує безліч практичних мов програмування для створення мобільних додатків з графічним інтерфейсом для Android. Найпоширеніші з них - Flutter, Java та Kotlin:

– Java: протягом багатьох років Java є стандартною мовою для створення додатків для Android. Вона пропонує значну екосистему інструментів, бібліотек та ресурсів, створених спеціально для розробки під Android. Для створення графічних користувацьких інтерфейсів за допомогою Android SDK (Software Development Kit), Java забезпечує потужну функціональність. Вона пропонує різноманітні елементи інтерфейсу та потужні інструменти для створення цікавих і візуально привабливих додатків.

– Kotlin: JetBrains представляє Kotlin, найсучаснішу мову програмування, як заміну Java для розробки для Android. Вона пропонує більш компактний та виразний синтаксис і повністю сумісна з Java. Kotlin створена для збільшення продуктивності розробників та підвищення стандартів їхньої роботи. Його легка адаптація для розробки під Android є результатом його безперешкодної взаємодії з кодом Java та фреймворками, які вже використовуються. За останні роки популярність

Kotlin зросла, і зараз вона часто розглядається як основна мова для Android розробників.

– Flutter: використовуючи єдину кодову базу, ви можете створювати нативні мобільні додатки для платформ Android та iOS за допомогою фреймворку Google, відомого як Flutter. Він використовує розроблену Google мову програмування Dart. Ви можете створювати естетично привабливі інтерфейси, що легко налаштовуються, за допомогою великої бібліотеки попередньо розроблених віджетів Flutter. Flutter є практичним варіантом для розробки додатків завдяки функції гарячого перезавантаження, яка дозволяє миттєво змінювати код та проводити швидкі ітерації. Крім того, продуктивність Flutter часто високо оцінюють за те, що він малює віджети прямо на екрані за допомогою рушія рендерингу, створюючи швидкі та витончені інтерфейси.

Також можна розробляти мобільні додатки з графічним інтерфейсом для Android за допомогою Python. Існує декілька фреймворків та інструментів, які дозволяють створювати додатки для Android, використовуючи Python:

– Kivy: фреймворк Python з відкритим вихідним кодом для створення мультисенсорних додатків називається Kivy. Він включає власну підтримку Android, а також інших операційних систем, таких як iOS, Windows і macOS. Kivy пропонує широкий вибір елементів інтерфейсу користувача (UI) і приймає введення з клавіатури, миші та дотику. Він не покладається на специфічні для платформи елементи інтерфейсу, а використовує власний графічний рушій.

– BeeWare: використовуючи набір інструментів і модулів BeeWare, ви можете створювати нативні користувацькі інтерфейси в Python для різних платформ, включаючи Android. Ви можете створювати додатки, які виглядатимуть на кожному пристрої як рідні, використовуючи рівень абстракції, який він пропонує над нативними компонентами інтерфейсу. BeeWare надає різноманітні інструменти для створення додатків для Android, включаючи Briefcase для пакування Python-додатків у вигляді

окремих виконуваних файлів та Toga для створення крос-платформних наборів інструментів інтерфейсу користувача.

– PySide/PyQt: PySide та PyQt - це прив'язки Python до фреймворку Qt, який є потужним і широко використовуваним інструментарієм для побудови графічних інтерфейсів. Хоча ці зв'язки в першу чергу орієнтовані на десктопні додатки, їх також можна використовувати для розробки додатків для Android. За допомогою таких інструментів, як PySide6-Android, ви можете пакувати та розгортати Qt-додатки на основі Python на Android.

В розробці мобільного додатку даної кваліфікаційної роботи буде використовуватись саме фреймворк Kivy, який є найпопулярнішим, а це означає, що найефективнішим, рішенням для розробки мобільних додатків під Android на мові Python.

Створення мобільних додатків для моніторингу амбулаторного лікування зараз є актуальною та гаряче обговорюваною темою у світі медичних технологій. Це пов'язано зі зростанням інтересу до електронної охорони здоров'я, зміною способів надання медичних послуг, а також збільшенням використання смартфонів та інших мобільних пристроїв.

Основними цілями створення мобільного додатку для відстеження амбулаторного лікування є покращення комунікації між пацієнтами та медичним персоналом, підвищення рівня медичного обслуговування, а також надання пацієнтам практичного та корисного інструменту для ведення особистого медичного журналу.

Наразі існує декілька мобільних додатків для відстеження амбулаторного лікування, які надають широкий спектр можливостей та функцій, наприклад:

– Medisafe Pill Reminder (рис.1.1);
– MyTherapy (рис.1.2);
– Pillo: Medication Reminder (рис.1.3);

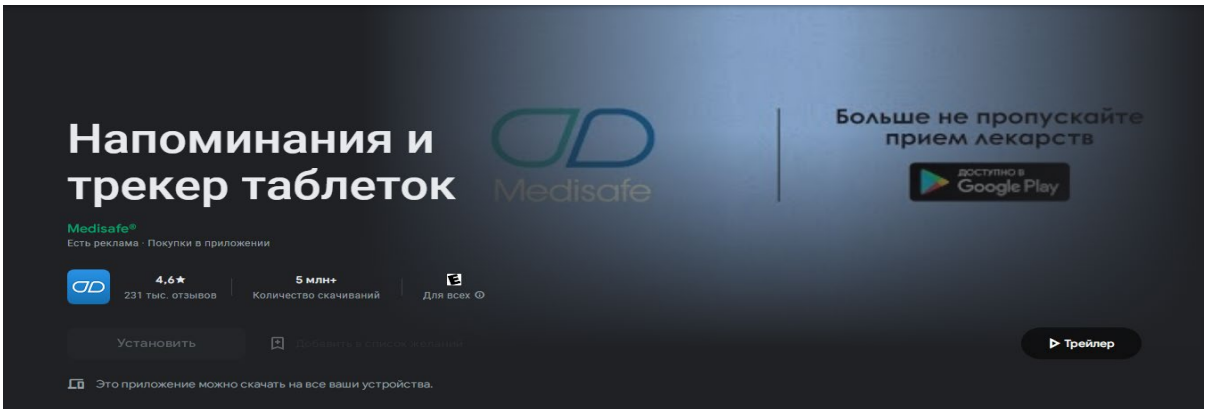

Рис.1.1. Додаток Medisafe Pill Reminder в Play Market

Medisafe Pill Reminder [25] - це додаток для смартфонів, який допомагає управляти прийомом ліків та нагадує про необхідні дози вчасно. Він розроблений з метою полегшення процесу лікування та забезпечення правильного прийому ліків.

Переваги додатку:

– Нагадування про прийом ліків. Додаток надає нагадування про час і дозу ліків, що допомагає уникнути пропуску дози та забезпечити правильне приймання ліків вчасно. Це особливо корисно для людей, які повинні приймати багато різних ліків в різний час;

– Система сповіщень. Додаток може надсилати сповіщення про прийом ліків не тільки на смартфон, але й на інші пристрої, такі як годинник або планшет. Це дозволяє зберігати нагадування незалежно від того, де ви знаходитесь.

– Управління курсом лікування. Додаток дозволяє створювати розклади та курси лікування, вказуючи тривалість терапії та дози. Це спрощує ведення медичного журналу та забезпечує точне виконання призначень;

– Взаємодія зі спільнотою. Додаток має функцію, яка дозволяє користувачам обмінюватися досвідом та підтримувати один одного. Це може бути особливо корисно для людей, які хочуть отримати підтримку і поради від інших, що приймають аналогічні ліки.

Недоліки:

– Залежність від технології. Для використання додатку потрібно мати доступ до смартфона або іншого пристрою, а також Інтернет-з'єднання. Це означає, що якщо батарея розряджена або немає доступу до мережі, нагадування можуть бути пропущені;

– Складність використання. Деякі користувачі можуть вважати додаток складним у використанні або потребують деякого часу, щоб звикнути до його функцій та налаштувань;

– Приватність та безпека. Використання додатку пов'язане з обробкою медичних даних. Користувачі повинні бути обережними та переконатися, що їхні дані захищені від несанкціонованого доступу та зловживання;

– Обмежені можливості безкоштовної версії. Додаток пропонує платну підписку з додатковими функціями. Безкоштовна версія може мати обмежені можливості та наявність реклами.

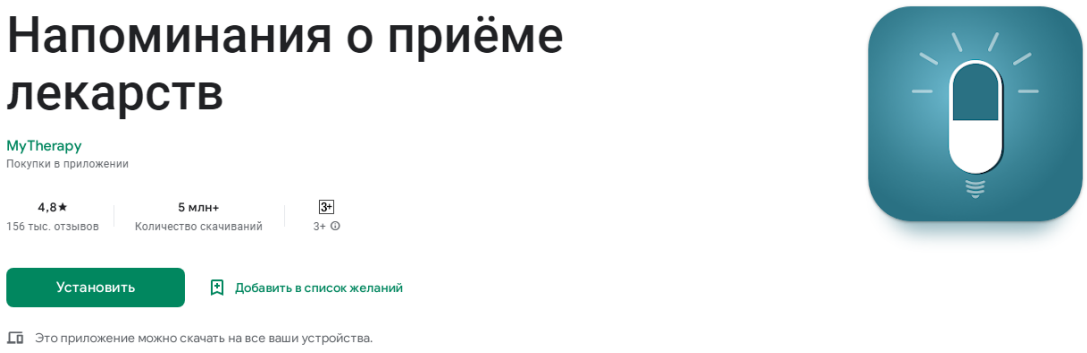

Рис.1.2. Додаток MyTherapy в Play Market

MyTherapy [26] - це мобільний додаток для керування здоров'ям, який призначений для допомоги людям у веденні їхнього розкладу прийому ліків, контролю за симптомами, моніторингу стану здоров'я та спілкування з медичним персоналом. Основною метою додатка є полегшення життя людей,

які страждають від хронічних захворювань або повинні регулярно приймати ліки.

Переваги:

– Планування та нагадування. Додаток дозволяє користувачам створювати персоналізований розклад прийому ліків, додавати нагадування та отримувати сповіщення на мобільний пристрій. Це допомагає уникнути пропуску доз та покращує дотримання лікування;

– Система слідкування за прийомом ліків. Додаток дає змогу вести журнал щодо прийому ліків, де користувачі можуть відмічати, коли вони прийняли свою дозу. Це дозволяє зберігати детальну інформацію про прийом ліків та надає можливість показати цю інформацію медичному персоналу;

– Симптоми та настрій. Користувачі можуть вести журнал щодо своїх симптомів та настрою, щоб стежити за їхнім станом здоров'я. Це може бути корисно для моніторингу ефективності лікування та надання медичним фахівцям об'єктивних даних;

– Взаємодія з медичним персоналом. Додаток надає можливість спілкуватися з медичним персоналом, надсилати повідомлення, задавати питання та отримувати рекомендації щодо лікування. Це допомагає покращити комунікацію між пацієнтом та лікарем.

Недоліки:

– Залежність від технології. Додаток вимагає наявності мобільного пристрою та Інтернет-з'єднання, щоб правильно функціонувати. Це може бути проблемою для людей, які не мають доступу до цих ресурсів або не володіють навичками використання сучасних технологій;

– Приватність та безпека. Використання мобільних додатків для зберігання медичних даних може створювати ризики для приватності та безпеки інформації. Необхідно впевнитися, що додаток має надійні заходи захисту даних та дотримується відповідних стандартів;

– Обмежені функції. Хоча додатку має багато корисних функцій, він може бути обмеженим в певних аспектах лікування. Наприклад, деякі пацієнти

можуть потребувати більш спеціалізованого функціоналу, що не доступний в цьому додатку;

– Вартість. Деякі функції додатку можуть бути доступні лише за додаткову плату або за підпискою. Це може бути фінансовим обмеженням для деяких користувачів.

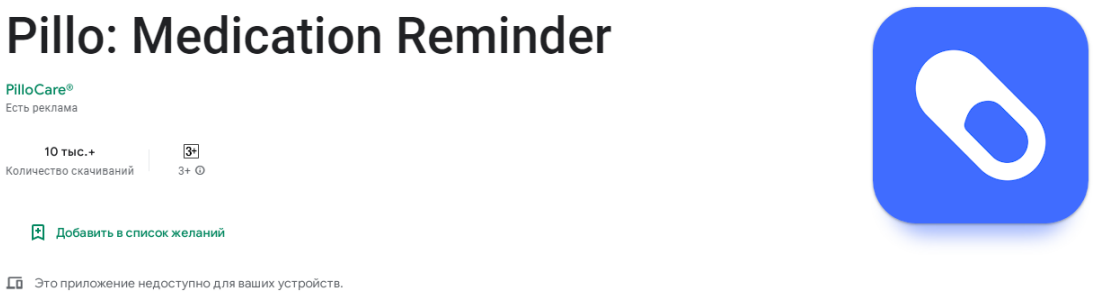

Рис.1.3. Додаток Pillo: Medication Reminder в Play Market

Pillo Medication Reminder [27] - це додаток, створений для мобільних пристроїв, щоб нагадувати про прийом лікарських препаратів. Його головна мета - допомогти людям зберігати регулярний графік прийому ліків і уникати пропусків доз. Додаток дозволяє встановлювати нагадування і надсилати сповіщення, щоб пам'ятати про важливість вчасного лікування. Це корисний інструмент для тих, хто має хронічні захворювання або приймає ліки за рецептом.

Переваги:

– Нагадування про прийом ліків. Додаток надсилає попередження та нагадування на мобільний пристрій у встановлені часи, що допомагає користувачам не забувати про прийом препаратів;

– Персоналізовані налаштування. Користувачі можуть налаштувати свій особистий розклад прийому ліків, включаючи часи, дози та назви препаратів. Це дозволяє адаптувати додаток до індивідуальних потреб кожного користувача;

– Система нагадування для близьких. Додаток також має можливість надсилати сповіщення членам сім'ї або близьким про прийом ліків користувача. Це особливо корисно для старших людей або людей з обмеженими можливостями, які потребують підтримки і контролю з боку родичів або доглядачів;

– Перегляд історії: Pillo Medication Reminder зберігає історію прийому ліків, що дозволяє користувачам відстежувати свій прогрес та дотримання розкладу лікування. Це може бути корисною функцією для допомоги медичним працівникам або для відстеження подальшого лікування.

Недоліки:

– Залежність від мобільного пристрою. Для користування додатком потрібний мобільний пристрій;

– Технічні проблеми. Як і в будь-якому мобільному додатку, можуть виникати технічні проблеми, такі як збої, помилки або несправність додатку. Це може призвести до ненадійності нагадувань та неправильного прийому ліків;

– Вартість та доступність. Деякі функції додатку можуть бути платними або доступними лише в підписці.

Вищенаведені аналоги мають високі оцінки серед користувачів, що означає, що на даний момент це еталонні приклади додатків такого типу.

З них для розробки була запозичена функція виведення повідомлень для нагадування про прийом ліків.

Також при розробці програми рекомендується усунути недоліки:

- Технічні проблеми. Варто максимально знизити шанс збою або обробити усі можливі сценарії збою для забезпечення стабільності роботи додатку.
- Вартість. Додаток планується до випуску без обмежень до функціоналу та вийти безкоштовним.

1.2. Призначення розробки та галузь застосування

Додаток розроблено з метою спрощення взаємодії лікаря та пацієнта, зокрема передачі списку необхідних заходів для лікування конкретної хвороби пацієнта.

Призначення розробки:

- перетворення паперових лікарняних призначень на цифрові;
- полегшення передачі переліку необхідних заходів та/або засобів для відновлення здоров’я від лікаря до пацієнта;
- більш надійне зберігання даних;
- повідомлення про час прийому ліків або вчинення заходу.

Додаток застосовується у галузі здоров’я та медицини. Така галузь на даний час вже має багато рішень для прискорення роботи лікарень, більш структурованого та безпечного зберігання даних про пацієнтів та лікарів, але все ще потребує якісних, а головне дешевих варіантів для вирішення цих задач.

1.3. Підстави для розробки

Підставами для виконання кваліфікаційної роботи є:

- освітня програма 122 «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;
- завдання на кваліфікаційну роботу на тему «Розробка мобільного додатку для відстеження амбулаторного лікування пацієнта за допомогою мови програмування Python та фреймворку Kivy»;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 350-с від 16.05.2023 р.

1.4. Постановка завдання

Метою створення додатку є впровадження цифрової мобільної системи, що дозволяє зберігати введену пацієнтом інформацію щодо лікарських засобів або заходів, що були приписані лікарем для відновлення здоров`я пацієнта.

Для реалізації зазначеного додатку були сформовані такі завдання:

- проаналізувати предметну галузь та галузь застосування;
- знайти та провести аналіз аналогів;
- сформулювати призначення розробки;
- розробити план розробки додатку та його структури;
- обрати технології, за якими буде розроблюватись додаток, та дослідити їх;
- сформувати функціональні характеристики та необхідні вимоги до них;
- визначити вимоги до технічних та програмних засобів;
- за планами структури та архітектури і за визначеними функціональними характеристиками розробити програму;
- провести тестування програми.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Розроблений продукт кваліфікаційної роботи має задовольняти наступні вимоги:

- має можливість зареєструвати користувача;
- має можливість зберегти авторизаційні дані та увійти за ними користувачу;
- має можливість створювати змінювані «примітки», що будуть мати інформацію про певний лікарський засіб або захід;

– має можливість переходити з екрану створення «примітки» до інформаційного екрану про ліки, заходи та хвороби, та навпаки;

– має можливість налаштовувати та виводити повідомлення про прийом ліків або відвідування процедур, якщо користувач вказав, що хоче їх отримувати.

1.5.2. Вимоги до інформаційної безпеки

Забезпечення інформаційної безпеки вимагає комплексу стратегічних заходів для захисту цінної інформації. Ці заходи мають на меті запобігти несанкціонованому доступу, неправомірному використанню, незаконному розголошенню, некоректній зміні або незаконному видаленню даних. Важливими аспектами безпеки даних є конфіденційність, цілісність та доступність інформації.

У швидкозмінному світі, особливо з технологічної та цифрової точки зору, загрози інформаційній безпеці можуть набувати різних форм і джерел. Ці загрози можуть походити зовні, наприклад, через хакерські атаки, кібершпигунство або фішинг, або зсередини організації, наприклад, через несанкціонований доступ співробітників до конфіденційної інформації. Розуміння і визначення цих загроз підкреслюють необхідність використання інтегрованого та міцного підходу до захисту інформаційних ресурсів.

Будь-яке програмне забезпечення, призначене для пошкодження або зловживання комп'ютерами, викрадення конфіденційних даних або втручання в повсякденну роботу, відоме як шкідливе програмне забезпечення. Згідно зі звітом McAfee[4], шкідливі програми були основною причиною загроз безпеці мобільних додатків за останні кілька років. Для більш ефективного захисту від атак шкідливих програм, важливо усвідомлювати ризики та дотримуватися найкращих практик безпеки.

Запобігання шкідливому програмному забезпеченню має велике значення і включає кілька важливих кроків. Один з них - постійне оновлення

програмного забезпечення до останньої версії. Виробники постійно виправляють виявлені уразливості та випускають патчі для захисту від нових загроз.

Щоб зменшити ризик завантаження шкідливого програмного забезпечення, слід завантажувати додатки тільки з офіційних джерел, таких як офіційні магазини програм для мобільних пристроїв. Ці магазини проводять перевірку додатків перед їх публікацією, що забезпечує додатковий рівень безпеки.

Варто уникати натискання на посилання або завантаження файлів з ненадійних джерел, оскільки вони можуть містити шкідливий код або створювати доступ для зловмисників до вашого пристрою.

Перед встановленням будь-якого додатка слід завжди перевіряти запитувані дозволи. Якщо запитані дозволи виглядають зайвими або несумісними з функціональністю додатку, краще утриматися від його встановлення.

Важливо не встановлювати програмне забезпечення з невідомих джерел або посилань, які надходять у повідомлення електронної пошти або соціальних мереж. Ці посилання можуть містити шкідливе програмне забезпечення, яке створене з метою злому вашої системи або крадіжки конфіденційної інформації.

Для додаткового рівня захисту існує можливість встановлення антивірусного програмного забезпечення на своєму мобільному пристрої. Воно здатне виявляти та блокувати шкідливе програмне забезпечення, забезпечуючи додатковий захист.

Також варто бути обережним при підключенні до невідомих або ненадійних Wi-Fi мереж. У таких мережах зловмисники можуть перехоплювати ваші дані або розповсюджувати шкідливе програмне забезпечення. Використовуйте довірені мережі з надійними захисними механізмами, щоб забезпечити безпеку своїх даних.

Незахищене зберігання даних[3] - це практика, коли приватна або конфіденційна інформація залишається відкритою або доступною. У контексті мобільних додатків незахищене зберігання даних може статися, коли додатки зберігають приватну інформацію користувача без необхідного шифрування або захисту, наприклад, паролі, фінансову інформацію, дані про особу або іншу конфіденційну інформацію у файловій системі або базі даних пристрою.

Незахищене зберігання даних користувачів у мобільних додатках робить їх вразливими для хакерів, витоків даних і небажаного доступу. Це може призвести до компрометації конфіденційних даних користувачів, крадіжки особистих даних, фінансового шахрайства та інших видів кіберзлочинів.

Приклади зловживань незахищеним зберіганням даних у мобільних додатках:

– крадіжка облікових даних: якщо додаток зберігає ім'я користувача та пароль у відкритому вигляді, зловмисник, який отримав доступ до пристрою, може легко отримати облікові дані користувача та використати їх для входу в обліковий запис користувача. Це може призвести до крадіжки особистих даних, фінансового шахрайства або інших форм кіберзлочинності;

– несанкціонований доступ: якщо додаток зберігає особисту інформацію користувача без належного шифрування, зловмисник, який отримав доступ до пристрою, може прочитати або скопіювати дані. Це може призвести до крадіжки особистих даних, атак соціальної інженерії або інших форм кіберзлочинності;

– встановлення шкідливого програмного забезпечення: якщо додаток зберігає дані користувача без належного шифрування, зловмисник, який отримав доступ до пристрою, може вставити в дані шкідливий код для виконання довільних команд або завантажити та встановити шкідливе програмне забезпечення. Це може призвести до повної компрометації пристрою та даних користувача;

– маніпулювання даними: якщо додаток зберігає фінансові дані користувача без належного шифрування, зловмисник, який отримує доступ до

пристрою, може змінити дані для переказу грошей або здійснення несанкціонованих транзакцій. Це може призвести до фінансових втрат і погіршення кредитної історії користувача.

Для запобігання виникнення перелічених способів маніпулювання слабкозахищеними або незахищеними даними у розроблюваному додатку має бути реалізований механізм захисту даних, уведених користувачем, у вигляді шифрування цих даних. Також рекомендується притримуватись правил інформаційної захищеності, вказаних вище.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для нормального функціонування мобільного додатку платформа повинна виконувати такі технічні нормативи:

- сенсорний екран смартфону (тачскрін);
- 100 МБ вільного місця в пам’яті;
- процесор з тактовою частотою не менше ніж 2 ГГц;
- не менше 4 ГБ оперативної пам’яті;

Ці характеристики є рекомендованими. Це означає, що при таких характеристиках або краще, додаток буде забезпечувати безперервну та коректну роботу.

1.5.4. Вимоги до інформаційної та програмної сумісності

Програмне забезпечення пристрою повинно мати операційну систему Android 12, що є рекомендацією для коректної та стабільної роботи продукту кваліфікаційної роботи.

Для розробки мобільного додатку необхідні такі компоненти:

- середа розробки (використана середа Python IDLE);
- Python версії 3+ (використана версія – Python 3.7.2);

– сімейство потрібних бібліотек фреймворку Kivy (OpenGL версії 2.0 або вище для можливості роботи з фреймворком);

– бібліотека cryptography та мова Rust для її роботи;

– бібліотека pyjnius;

– бібліотека datetime;

– бібліотека os;

– бібліотека json.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Призначенням розробленого додатку є дозвіл до введення, відображення і збереження інформації про лікарські засоби та заходи, що вводяться користувачем, та інформування користувача про загальну інформацію щодо них. Також програма інформує користувача про певні заходи згідно введеним даним.

Загалом, розроблений продукт кваліфікаційної роботи має конкретні функції:

– реєстрація та авторизація користувача;

– можливість введення інформації про розклад прийому певного лікарського засобу;

– відображення й збереження введених даних;

– відображення загальної інформації про різні лікарські засоби та заходи (на даний момент у програми наведена обмежена кількість лікарських засобів та заходів як приклад роботи);

– побудова повідомлення на основі переваг користувача та їх відображення;

– захист збережених даних шляхом їх шифрування;

– зрозумілий дизайн;

– стабільний робочий стан.

2.2. Опис застосованих математичних методів

Оскільки особливості предметної галузі розв’язуваної задачі не передбачають застосування математичних методів, при розробці даного додатку математичні методи не використовувалися.

Математичні методи та функції, що використовуються для виставлення розмірів елементів або задання їх позицій та інших функціональних алгоритмів, прописані у застосованих бібліотеках.

2.3. Опис використаних технологій та мов програмування

Перелік основних технологічних аспектів, потрібних для розробки додатку:

- мова програмування Python;
- фреймворк Kivy;
- бібліотека pyjnius;
- мова програмування Java;
- бібліотека cryptography;
- бібліотека datetime;
- бібліотека os;
- бібліотека json.

Мова програмування Python [5-6, 24].

Згідно з опитуванням StackOverflow Python була названа другою найулюбленішою та найбільш швидкозростаючою мовою програмування серед розробників після Rust. Будучи універсальною мовою, вона є кращим вибором для підприємців, які шукають проекти з машинного навчання та Data Science.

З точки зору виходу на ринок праці, студенти віддають перевагу Python, оскільки її легко зрозуміти та кодувати. Більше того, багато організацій використовують Python для багатьох своїх проектів.

Переваги:

- Python - мова високого рівня. Мови високого рівня більш читабельні. Вони використовують осмислені імена змінних і мають осмислений синтаксис. Не потрібно розуміти базову операційну систему. У цьому відношенні Python схожий на інші мови програмування, включаючи

JavaScript, Rust і C++, але є ще більш зрозумілим і читабельним. На протилежному кінці спектру знаходиться мова асемблера. Асемблерний код посилається на адреси пам'яті і використовує інструкції машинної мови;

– Python підтримує об'єктно-орієнтоване програмування (ООП). Python - це ООП мова з підтримкою класів, методів, успадкування та інкапсуляції. На відміну від Java, Python не нав'язує ООП-модель, а об'єктно-орієнтовані принципи проектування є абсолютно необов'язковими. Тому Python можна використовувати суто в імперативному/процедурному режимі для коротких програм і простих утиліт;

– велика спільнота. Спільнота Python може похвалитися великою кількістю кодерів, розробників, професіоналів та студентів на одній платформі, які допомагають один одному та отримують від цього найкраще;

– гнучкість і розширюваність. Python надзвичайно гнучкий і може бути розширений для розробки на інших мовах. Розробники можуть писати код на C та C++ і створювати нові функції на мові з динамічною типізацією;

– обширні бібліотеки. Python має великий набір бібліотек і містить код для різних цілей, таких як створення документації, регулярні вирази, веб-браузери, модульне тестування, CGI, бази даних, маніпулювання зображеннями тощо. Таким чином, він усуває необхідність писати весь код вручну;

– Python - це мова загального призначення: Вузькоспеціалізовані мови призначені для однієї конкретної мети. Наприклад, SQL використовується лише для спілкування з реляційними системами баз даних. Проте Python є мовою загального призначення і має широкий спектр застосувань;

– мультиплатформеність. Python є портативним, що означає, що його можна запускати на будь-якій іншій платформі. Тут вам потрібно написати код лише один раз, і ви можете запустити його де завгодно. Це називається WORA (Write Once Run Anywhere). Це полегшує розробникам

роботу з Python, оскільки їм не потрібно вносити в нього зміни, якщо вони захочуть запустити його на іншій платформі;

– відсутність процесу компіляції. Python - це інтерпретована мова, і програми автоматично компілюються під час виконання. Програму можна запускати одразу після написання. Немає окремого компілятора, немає трудомісткого етапу компіляції і немає непрозорих помилок компілятора.

Недоліки:

– повільніша за скомпільовані мови. Python набагато повільніша порівняно з такими мовами, як C та Java. Python інтерпретується та динамічно типізується, тому компілятор часу виконання має багато роботи. Він повинен постійно перевіряти тип кожної змінної. Це означає, що Python - не найкращий вибір для сценаріїв, де швидкість є критично важливою;

– безпека. Python не є безпечним на 100%, тому для забезпечення безпеки коду потрібно вжити додаткових заходів;

– Python не оптимізований для зменшення споживання пам'яті. Python може використовувати вдесятеро більше оперативної пам'яті, ніж програма, написана більш економною мовою. Однак це частково плата за гнучкість і простоту використання. Крім того, збирач сміття в Python не може зібрати всі відкинуті ресурси одразу, що зменшує обсяг доступної пам'яті. Python не є хорошим вибором для середовищ з обмеженою пам'яттю;

– невелика популярність у мобільних та настільних додатках. Через те, що Python є дещо повільною мовою і використовує багато пам'яті, вона не набула значного поширення в мобільному просторі. Існує кілька інструментів розробки на Python для мобільних додатків, але вони більш обмежені, ніж фреймворки для інших мов. Ситуація трохи краща у клієнтських десктопах, але для фронтенд-додатків Python все ще не надто популярний;

– не оптимізований для доступу до баз даних. Працювати з базами даних у Python складніше, ніж у деяких інших мовах. Python не має потужного, високоякісного, простого у використанні інтерфейсу, подібного до Java Database Connectivity (JDBC). Його все ще можна використовувати, якщо

читання і запис до бази даних є відносно простими. Але це не найкращий вибір для додатків, які мають складну взаємодію з великою корпоративною базою даних.

Для розробки додатку була обрана мова програмування Python для демонстрації можливості швидкої розробки мобільних додатків на ній або розробки приблизної архітектури та структури мобільних додатків.

Фреймворк Kivy [7].

Kivy - це багатоплатформна бібліотека розробки графічного інтерфейсу з відкритим вихідним кодом для Python, яка може працювати на iOS, Android, Windows, OS X та GNU/Linux. Вона допомагає розробляти додатки, які використовують інноваційний мультисенсорний інтерфейс. Основна ідея Kivy полягає в тому, щоб дозволити розробнику створити додаток один раз і використовувати його на всіх пристроях, роблячи код придатним для багаторазового використання і розгортання, дозволяючи швидко і легко проектувати взаємодію і швидко створювати прототипи.

Цей фреймворк містить всі елементи для створення додатків, такі як:

– широка підтримка пристроїв введення, таких як миша, клавіатура, TUIO та специфічні для ОС мультисенсорні події;

– графічна бібліотека, що використовує лише OpenGL ES 2;

– широкий вибір віджетів, створених з підтримкою мультитач;

– проміжна мова Kv, яка використовується для легкого створення власних віджетів.

Переваги:

– заснований на Python, який є надзвичайно потужним, враховуючи його багату бібліотеку;

– можна написати код один раз і використовувати його на всіх пристроях;

– прості у використанні віджети з підтримкою мультитач;

– працює краще, ніж кросплатформенні альтернативи HTML5.

Недоліки:

– незрозумілий на вигляд інтерфейс користувача;

– більший обсяг компонування (через необхідність врахування інтерпретатора Python);

– недостатня підтримка спільноти (спільнота Kivy не є особливо великою);

– нестача хороших прикладів і документації;

– доступні кращі та багатші на спільноту альтернативи, якщо зосередитися лише на мобільних крос-платформних пристроях.

Бібліотека pyjnius [8].

Pyjnius є модулем Python, який спрощує доступ до Java-класів в середовищі Python шляхом використання Java Native Interface (JNI). Цей модуль створює зв'язок між Python і Java, дозволяючи взаємодіяти між кодом обох мов. Розробники Python високо цінують цей модуль, оскільки він дозволяє використовувати Java-бібліотеки та фреймворки у їх Python-проектах. Завдяки Pyjnius розробники можуть зручно та ефективно використовувати функціонал Java в своїх програмах, не переписуючи весь код на Java або Python. Це робить Pyjnius важливим інструментом для інтеграції цих двох популярних мов програмування, що розширює можливості розробників у програмуванні.

Переваги:

– доступ до широкого спектру бібліотек та фреймворків Java;

– можливість використання Java коду в Python проектах;

– покращена продуктивність у порівнянні з іншими мостами Python-Java, такими як Jython;

– pyjnius активно підтримується і має велику спільноту користувачів.

Недоліки:

– pyjnius потребує робочої інсталяції Java в системі;

– pyjnius може бути складнішим у налаштуванні та використанні, ніж інші мости між Python та Java, такі як JPype;

– pyjnius може бути повільнішим, ніж використання Java напряму;

– pyjnius має обмеження на кількість локальних посилань, які можна створювати, що може викликати проблеми при роботі з великими об'єктами Java;

– pyjnius може бути сумісним не з усіма методами та бібліотеками Java.

Мова програмування Java [9-10].

Незважаючи на те, що було відкрито багато нових мов, слава Java ніколи не згасає. Вже понад 20 років Java займає лідируючу позицію серед усіх інших мов. Більшість експертів не можуть заперечити той факт, що Java є однією з найпотужніших і найефективніших мов, коли-небудь створених, і є найбільш широко використовуваною мовою програмування в багатьох галузях. Причина в тому, що вона має багато переваг і можливостей, які допомагають розробникам з легкістю вирішувати складні реальні проблеми. Java також допомагала розробляти різноманітне програмне забезпечення в минулому і продовжує допомагати в розробці в сьогоденні.

Переваги:

– проста і зрозуміла. Будь-яку мову можна вважати простою, якщо її легко вивчити і зрозуміти. Синтаксис Java простий, його легко писати, вивчати, підтримувати та розуміти, код легко налагоджувати. Більше того, Java менш складна, ніж такі мови, як C та C++, тому що багато складних особливостей цих мов були вилучені з Java, такі як концепція явних вказівників, класи зберігання, перевантаження операторів та багато іншого;

– Java є об'єктно-орієнтованою мовою програмування. Java - це об'єктно-орієнтована мова, яка допомагає нам підвищити гнучкість і можливість повторного використання коду. Використовуючи концепцію ООП, ми можемо легко повторно використовувати об'єкт в інших програмах. Це також допомагає підвищити безпеку, зв'язуючи дані та функції в єдине ціле і не дозволяючи зовнішньому світу отримати до них доступ. Це також допомагає організувати великі модулі в менші, щоб їх було легко зрозуміти;

– захищена мова. Java зменшує загрози та ризики безпеки, уникаючи використання явних вказівників. Покажчик зберігає в пам'яті адресу іншого значення, що може призвести до несанкціонованого доступу до пам'яті. Ця проблема вирішується шляхом видалення концепції вказівників. Крім того, в Java для кожного додатку існує менеджер безпеки, який дозволяє визначати правила доступу до класів;

– дешева та економічна в обслуговуванні. Програми на Java дешеві у розробці та підтримці, оскільки вони залежать від певної апаратної інфраструктури для запуску. Ми можемо легко виконати їх на будь-якій машині, що зменшує додаткові витрати на підтримку;

– незалежна від платформи. Java пропонує дуже ефективну перевагу своїм користувачам, надаючи можливість незалежності від платформи, тобто функцію Write Once Run Anywhere (WORA). Скомпільований код, тобто байт-код java, не залежить від платформи і може працювати на будь-якій машині незалежно від операційної системи;

– Java - це мова програмування високого рівня, оскільки вона зрозуміла людині. Вона схожа на людську мову і має дуже простий і легкий у підтримці синтаксис, який схожий на синтаксис мови C++, але в більш простому вигляді;

– Java забезпечує автоматичне очищення даних. У Java існує автоматичне керування пам'яттю, яке здійснюється за допомогою віртуальної машини Java (JVM). Кожного разу, коли об'єкти більше не використовуються програмами і не посилаються ні на що, на них не потрібно робити посилання або видаляти шляхом явного програмування. Java автоматично видаляє невикористані об'єкти за допомогою процесу автоматичного очищення (Garbage Collection);

– стабільна мова. Програми на Java є більш стабільними у порівнянні з програмами на інших мовах. Більше того, нова версія Java випускається в найкоротші терміни з більш досконалими функціями, що робить її більш стабільною.

Недоліки:

– повільна і має низьку продуктивність. Java споживає багато пам'яті і працює значно повільніше, ніж рідні мови, такі як C або C++. Вона також повільна порівняно з іншими мовами, такими як C та C++, тому що кожен код повинен бути інтерпретований на машинний рівень. Така низька продуктивність пов'язана з додатковим рівнем компіляції та абстрагування в JVM. Більше того, іноді збирач сміття призводить до низької продуктивності Java, оскільки він споживає більше процесорного часу;

– надає не надто привабливий вигляд та відчуття графічного інтерфейсу. Хоча для створення графічного інтерфейсу в Java існує багато конструкторів графічного інтерфейсу, вони не підходять для створення складних інтерфейсів. Існує багато невідповідностей при їх використанні. Існує багато популярних фреймворків, таких як Swing, SWT, JavaFX, JSF для створення GUI. Але вони недостатньо розвинені для розробки складного інтерфейсу;

– Java не надає можливості резервного копіювання. Java в основному працює на зберігання і не фокусується на резервному копіюванні даних. Це основний недолік, через який вона втрачає популярність і рейтинги серед користувачів;

– Java вимагає значного обсягу пам'яті. Java вимагає значного або переважного обсягу пам'яті у порівнянні з іншими мовами, такими як C та C++. Під час виконання збірки сміття може негативно вплинути на ефективність використання пам'яті та продуктивність системи;

– довгі та складні коди. Коди на Java багатослівні, тобто в них багато слів і багато довгих і складних речень, які важко читати і розуміти. Це може знизити читабельність коду. Java зосереджується на тому, щоб бути більш керованою, але в той же час їй доводиться йти на компроміс з надто складними кодами та довгими поясненнями до кожної речі.

Бібліотека cryptography [11-14].

Cryptography - це практика захисту корисної інформації під час передачі з одного комп'ютера на інший або зберігання даних на комп'ютері. Cryptography займається шифруванням відкритого тексту в зашифрований і розшифровуванням зашифрованого тексту в відкритий. Python підтримує пакет cryptography, який допомагає шифрувати та розшифровувати дані. Модуль fernet пакету cryptography має вбудовані функції для генерації ключа, шифрування відкритого тексту у зашифрований та розшифрування зашифрованого тексту у відкритий, використовуючи методи encrypt та decrypt відповідно. Модуль fernet гарантує, що дані, зашифровані за його допомогою, не можуть бути надалі маніпульовані або прочитані без ключа. Потребується установка мультипарадигмальної системної мови програмування Rust для роботи з модулем cryptography. Більш детально з Rust можна дізнатись за посиланням [23].

Переваги бібліотеки:

– cryptography надає широкий спектр криптографічних алгоритмів, включаючи симетричне шифрування, асиметричне шифрування, хешування та цифрові підписи;

– cryptography активно підтримується і має хорошу документацію;

– cryptography розроблена так, щоб бути простою у використанні і надає високорівневі API, які абстрагуються від багатьох низькорівневих деталей криптографії;

– cryptography побудовано на основі бібліотеки OpenSSL, яка широко використовується і була ретельно протестована.

Недоліки:

– cryptography може бути складною і важкою у використанні для розробників, які не знайомі з криптографією;

– cryptography не є панацеєю і не може гарантувати безпеку системи сама по собі. Важливо використовувати криптографію в поєднанні з іншими заходами безпеки, такими як безпечне кодування та безпечний дизайн системи;

– cryptography може бути повільною та ресурсоємною, особливо при використанні певних криптографічних алгоритмів.

Бібліотека datetime [15-18].

Модуль Python datetime включає в себе класи для роботи з датою і часом. Ці класи надають ряд функцій для роботи з датами, часом і часовими інтервалами. date і datetime є об'єктами в Python, тому при маніпулюванні ними здійснюється маніпулювання об'єктами, а не рядками або часовими мітками.

Переваги:

– гнучкість. Модуль datetime надає широкий спектр функцій і методів для роботи з датами і часом, що дозволяє легко виконувати складні операції;

– вбудований модуль. Модуль datetime є частиною стандартної бібліотеки Python, а це означає, що він доступний за замовчуванням у будь-якій інсталяції Python;

– маніпулювання датами та часом. Модуль datetime надає класи для маніпулювання датами та часом. З його допомогою можна виконувати арифметичні операції над датами і часом, форматувати їх і конвертувати в різні часові пояси;

– об'єктно-орієнтований підхід. Модуль datetime є об'єктно-орієнтованим, а це означає, що він забезпечує більш інтуїтивно зрозумілий і послідовний спосіб роботи з датами і часом.

Недоліки:

– обмежена підтримка часових поясів. Модуль datetime не надає повного рішення для роботи з часовими поясами. Може бути важко впоратися зі зміною літнього часу та іншими проблемами, пов'язаними з часовими поясами;

– обмежена підтримка високосних секунд. Модуль datetime не надає можливості обробляти високосні секунди, які додаються до деяких шкал часу, щоб синхронізувати їх з обертанням Землі;

– обмежений діапазон. Модуль datetime має обмежений діапазон дат, які він може представляти. Він може відображати дати від 1 січня 1 року нашої ери до 31 грудня 9999 року нашої ери, але не може відображати дати за межами цього діапазону;

– обмежена точність. Модуль datetime має кінцеву точність у мікросекунди. Якщо вам потрібно працювати з більш точними значеннями часу, можливо, доведеться використовувати іншу бібліотеку;

– продуктивність. Модуль datetime може бути повільним при роботі з великими наборами даних або при виконанні складних обчислень. У деяких випадках може знадобитися використання більш продуктивної бібліотеки.

Бібліотека os [19-20].

Модуль os у Python є частиною стандартної бібліотеки мови програмування. При імпортуванні він дозволяє користувачеві взаємодіяти з рідною операційною системою, в якій наразі працює Python. Простіше кажучи, він надає користувачеві простий спосіб взаємодії з деякими функціями ОС, які стають в нагоді у повсякденному програмуванні.

Цей модуль також містить два підмодулі, модуль os.sys та os.path. Надані модулем os функції можна використовувати для виконання широкого спектру завдань. Серед найпоширеніших - виконання команд командного інтерпретатора, керування файлами і каталогами, запуск процесів тощо.

Переваги:

– надає спосіб взаємодії з операційною системою у незалежний від платформи спосіб. Це означає, що один і той самий код можна використовувати на різних операційних системах;

– дозволяє виконувати різноманітні операції з файлами, такі як створення, видалення, перейменування та переміщення файлів і каталогів;

– надає можливості для виконання системних команд і скриптів;

– містить інформацію про систему, таку як поточний робочий каталог, змінні оточення та конфігурацію системи.

Недоліки:

– може бути складним у використанні і вимагає хорошого розуміння операційної системи;

– може залежати від платформи, тобто деякі функції можуть бути доступні не на всіх операційних системах;

– може працювати повільніше, ніж інші модулі з подібною функціональністю, особливо при виконанні файлових операцій.

Бібліотека json [21-22].

JSON в Python - це стандартний формат, натхненний JavaScript, для обміну даними та передачі даних у текстовому форматі через мережу. JSON представляє об'єкти як пари ім'я/значення, подібно до словника Python. Як правило, JSON має рядковий або текстовий формат. Він може використовуватися API та базами даних і представляє об'єкти у вигляді пар ім'я/значення. JSON розшифровується як JavaScript Object Notation.

Цей модуль надає всі необхідні інструменти для роботи з об'єктами JSON, включаючи синтаксичний аналіз, серіалізацію, десеріалізацію та багато іншого.

Переваги:

– легкий перехід між контейнером і значенням (JSON в Python і Python в JSON);

– легко зрозумілий;

– широко використовується і добре підтримується більшістю мовами програмування;

– легкий та швидкий;

– можна використовувати для представлення складних структур даних.

Недоліки:

– не може обробляти довільні об'єкти;

– погано обробляє вкладеність, залишаючись при цьому читабельним і легким для аналізу;

– не підтримує коментування.

2.4. Опис структури системи та алгоритмів її функціонування

Задля зображення структури системи (рис. 2.1) буде наведено структуру папок проекту:

– папка «diplom», яка містить в собі файл розширення .py, який являє собою саму програму, та дві папки: «data» і «saved_data»;

– папка «data», що містить у собі приклад структурованого збереження інформації про лікарські засоби та заходи у вигляді файлів типу .txt;

– папка «saved_data», в якій зберігається: файл «labels.json», який містить у собі інформацію про збережені інформаційні віджети, що були додані користувачем, які відтворюються при новому запуску додатку, та папка «authorization_data»;

– папка «authorization_data», в якій є файл «crypt_key.txt», в якому зберігається ключ, за допомогою якого шифруються авторизаційні дані. Також в цю папку додаються файли типу .json з зашифрованими авторизаційними даними (логін та пароль).

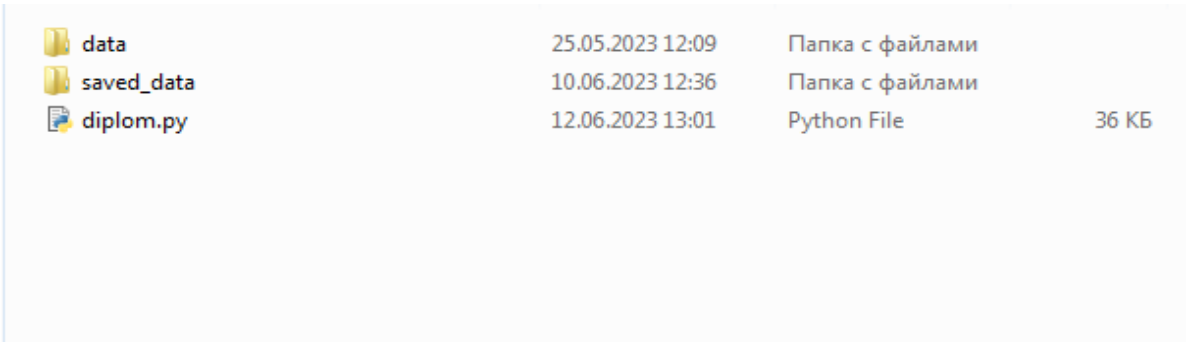

Рис. 2.1. Зображення структури систему у стандартному файловому менеджері Windows 7

Збереження даних та їх відтворення при запуску програми відбувається за допомогою схожих друг на друга алгоритмів взаємодії з файлами типу .txt та .json.

Один з прикладів алгоритмів збереження даних, націлений на збереження та шифрування авторизаційних даних користувача, наведений нижче у вигляді функції:

```
…
def save_data(self, instance):
    self.remove_previous_labels()

    if self.login_register.text == '':
        label = MyLabel(text='Fill the login field, please',
                        color=(1, 0, 0, 1),
                        size_hint=(.2, .1),
                        pos_hint={'x': .25, 'y': .3},
                        font_size=22)
        self.root.add_widget(label)
        self.labels.append(label)
        return

    if self.password_register.text == '':
        label = MyLabel(text='Fill the password field, please',
                        color=(1, 0, 0, 1),
                        size_hint=(.2, .1),
                        pos_hint={'x': .55, 'y': .3},
                        font_size=22)
        self.root.add_widget(label)
        self.labels.append(label)
        return

    if len(self.password_register.text) < 8:
        label = MyLabel(text = 'Password must have at least 8 characters',
                        color=(1, 0, 0, 1),
                        size_hint=(.2, .1),
                        pos_hint={'x': .55, 'y': .3},
                        font_size=22)
        self.root.add_widget(label)
        self.labels.append(label)
        return

    crypted_login = encryption(self.login_register.text.encode(), crypt_key)
    crypted_password = encryption(self.password_register.text.encode(), crypt_key)
    data = {f'{crypted_login}': f'{crypted_login}',
            f'{crypted_password}': f'{crypted_password}'}

    with open(f'saved_data/authorization_data/{self.login_register.text}.json', 'w') as file:
        json.dump(data, file)
```

```
        self.change_to_log_in(instance)
…
```

Ще один приклад, який демонструє алгоритм перевірки даних на достовірність при спробі увійти за ними у вигляді функції:

```
…
def check_data(self, instance):
        self.remove_previous_labels()

        try:
            with open(f'saved_data/authorization_data/{self.login_input.text}.json', 'r') as file:
                data = json.load(file)
                data_list = list(data.keys())
                temp_list = [data_list[0].split("""")[1],
                             data_list[1].split("""")[1]]
                decrypted_password = str(decryption(temp_list[1], crypt_key)).split("""")[1]
                if self.password_input.text == decrypted_password:
                    self.app.switch_screen('Main menu')
                else:
                    label = MyLabel(text='Invalid password',
                                    color=(1, 0, 0, 1),
                                    size_hint=(.2, .1),
                                    pos_hint={'x': .4, 'y': .3},
                                    font_size=22)
                    self.root.add_widget(label)
                    self.labels.append(label)
                    self.password_input.text = ''
                    return
        except(FileNotFoundError, json.JSONDecodeError):
            label = MyLabel(text='There is no account with this login',
                            color=(1, 0, 0, 1),
                            size_hint=(.2, .1),
                            pos_hint={'x': .4, 'y': .3},
                            font_size=22)
            self.root.add_widget(label)
            self.labels.append(label)
            self.login_input.text = ''
            self.password_input.text = ''
            return
…
```

Ключ, що використовується для шифрування та дешифрування даних, береться з відповідного файлу, який був згенерований раніше за допомогою модулю cryptography.Fernet, та використовується у двох функціях, виконуючих наведені вище дії:

```
…
with open('saved_data/authorization_data/crypt_key.txt', 'rb') as file:
    crypt_key = file.read()

def encryption(text, keylock):
```

```
    key = Fernet(keylock)
    return key.encrypt(text)

def decryption(crypted_text, keylock):
    key = Fernet(keylock)
    return key.decrypt(crypted_text)
…
```

Повідомлення будуються за допомогою бібліотеки pyjnius, яка дозволяє брати доступ до класів Java, відповідаючих за ту чи іншу частину, потрібну для виконання цільових функцій.

Нижче наводиться приклад використання модулів pyjnius та datetime для побудови таргетованого повідомлення, що має надсилатися у відповідний до обраного користувачем часу доби, у вигляді фрагментів коду:

```
…
PythonActivity = autoclass('org.kivy.android.PythonActivity')
Notification = autoclass('android.app.Notification')
NotificationManager = autoclass('android.app.NotificationManager')
Context = autoclass('android.content.Context')
…
…
def schedule_notification(self, instance, time):
        try:
            debugged_time = int(time)
            Clock.schedule_once(self.schedule_notification_callback,
self.get_notification_delay(debugged_time))
        except ValueError:
            pass


    def get_notification_delay(self, time):
        now = datetime.now()
        target_time = now.replace(hour = time,
                                  minute = 0,
                                  second = 0,
                                  microsecond = 0)

        if now > target_time:
            target_time = target_time.replace(day = target_time.day + 1)
        delta = target_time - now
        return delta.total_seconds()

    def schedule_notification_callback(self, dt):
        self.show_notification(dt)
        Clock.schedule_once(self.schedule_notification_callback, self.get_notification_delay())

    def show_notification(self, dt):
        notification = Notification()
        notification.icon = PythonActivity.mActivity.getApplicationInfo().icon
```

```
    notification.tickerText = 'Notification'
    notification.contentTitle = 'Important notification'
    notification.contentText = 'Time to take a pill!'
    notification.flags |= Notification.FLAG_AUTO_CANCEL
    notification_manager =
PythonActivity.mActivity.getSystemService(Context.NOTIFICATION_SERVICE)
    notification_manager.notify(1, notification)
…
```

Як можна помітити, модуль pyjnius дійсно дозволяє використовувати Java класи при розробці на мові Python, але для розуміння того, які саме класи потрібно взяти або ж як ними правильно користуватися, повинні бути присутніми знання мови Java.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Оскільки мобільний додаток має адаптивний розмір вікна, то за один з вхідних параметрів можна вважати розмір вікна пристрою, на якому виконується запуск додатку.

Перелік вхідних даних:

- логін та пароль користувача;
- дані про вживання певного лікарського засобу;
- дотики або кліки мишкою під час роботи з візуальними компонентами програми;
- розмір екрану базового пристрою (для адаптивного розміру вікна програми та адаптивного розміщення візуальних компонентів програми);
- операційна система базової системи;
- програмні характеристики базової системи (для роботи фреймворку Kivy);
- час повідомлення.

Перелік вихідних даних:

- візуальні компоненти програми;

– збереження даних про вживання певного лікарського засобу;
– збереження авторизаційних даних (пароль та логін);
– координати розміщення візуальних компонентів;
– інформація про певні хвороби, засоби або заходи.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Продукт кваліфікаційної роботи рекомендується використовувати на мобільних пристроях з версією Android не нижче 8.0. Можливе використання на десктопних операційних системах типу Windows, але при цьому не гарантується коректна робота функцій повідомлень програми.

Також рекомендована наявність 4-х або більше ГБ оперативної пам`яті через навантаження на неї при роботі стосунку у зв’язку з написанням програми на мові Python.

Інтерфейс програми розроблювався під роздільні здатності екрану типу 720x1600. При використанні програми на десктопних моніторах не гарантується правильне відображення інтерфейсу програми.

Параметри пристрою, на якому здійснювалося тестування програми:

– операційна система Windows 7;
– 1.9 ГГц тактової частоти процесора;
– 4 ГБ оперативної пам`яті;
– роздільна здатність екрану 1366x768;
– миша;
– клавіатура;

Параметри мобільного пристрою, на якому здійснювалося тестування програми:

– операційна система Android 12.0;
– 2.3-1.8 ГГц тактової частоти процесора;

– 4 ГБ оперативної пам`яті;
– тачскрін.

2.6.2. Використані програмні засоби

Розробка проекту здійснювалася на мові Python з використанням фреймворку Kivy, а також модулів json, os, datetime, pyjnius та cryptography.

Для написання коду програми була обрана вбудована середа розробки IDLE. Також потребувалася система управління пакетами Python pip.

Необхідними програмними засобами для коректної роботи додатку є мова Rust для роботи модулю шифрування та дистрибутив Java (версія, використана при тестуванні) для роботи з модулем pyjnius.

2.6.3. Виклик та завантаження програми

Для початку роботи програми на мобільному пристрої необхідно скомпілювати .apk файл з усіма структурними та програмними складовими. Після цього установити програму за допомогою стандартного інсталятора Android шляхом відкриття скомпільованого файлу. Після цього потрібно запустити програму. Це рекомендований спосіб.

Ще один спосіб полягає у відкритті .py файлу у середі розробки, компіляція та виконання коду програми. Для цього на пристрої повинні бути встановлені потрібні програмні засоби.

2.6.4. Опис інтерфейсу користувача

При запуску додатку відображається екран реєстрації (якщо ще немає зареєстрованого користувача) (рис.2.2).

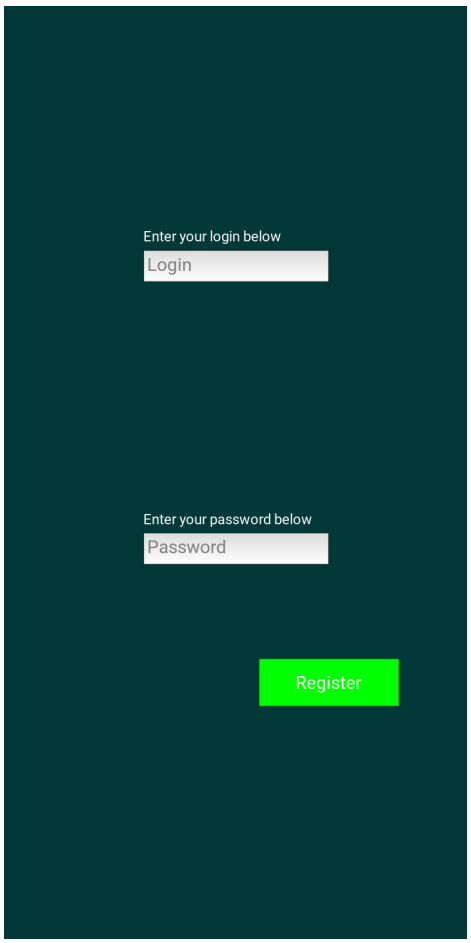

Рис. 2.2. Екран реєстрації

На екрані реєстрації потребується внесення реєстраційних даних (логіну та пароля). Логін може бути яким завгодно. Кнопка «Register» виконує функції збереження даних, їх шифрування та перехід до іншого екрану. Якщо поля логіну та паролю, тільки логіну або тільки паролю залишаться порожніми при натисканні кнопки «Register», буде виведене прохання заповнити.

Є вимога до пароля – він повинен бути довжиною не менше 8-ми символів. Якщо довжина не буде відповідати вимозі при натисканні кнопки «Register», буде виведене відповідне повідомлення.

При успішній реєстрації користувача він побачить екран входу (рис.2.3), в якому зможе ввести свої авторизаційні дані.

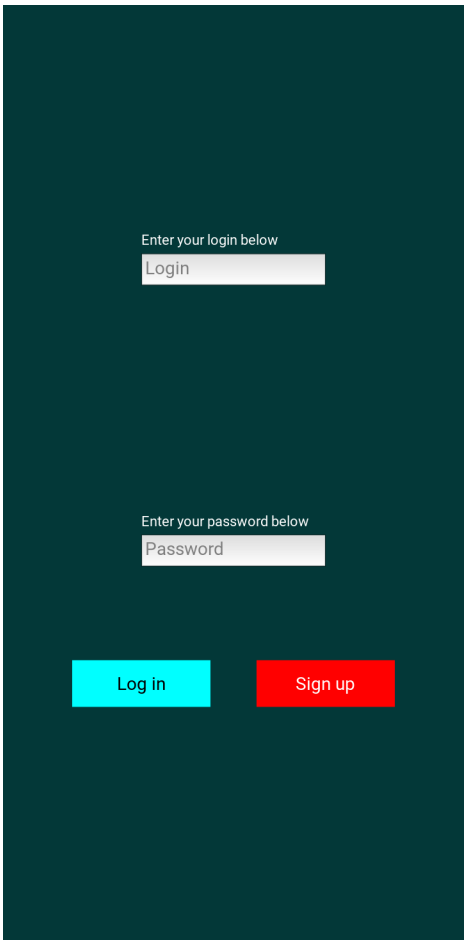

Рис. 2.3. Екран входу

За допомогою натискання кнопки «Sign up» користувач може повернутися до екрану реєстрації, якщо, наприклад, захоче створити ще один профіль. У такому випадку виконується перехід до екрану реєстрації з додатковою кнопкою «Back», яка дозволяє повернутися до екрану входу.

Якщо користувач при введенні логіну або пароля допустить помилку або залишить поле пустим та натисне кнопку «Log in», виведеться текст з проханням заповнити або виправити відповідні поля.

При заповненні полів достовірними даними та натисканні кнопки входу, користувач попаде в головний екран (рис.2.4).

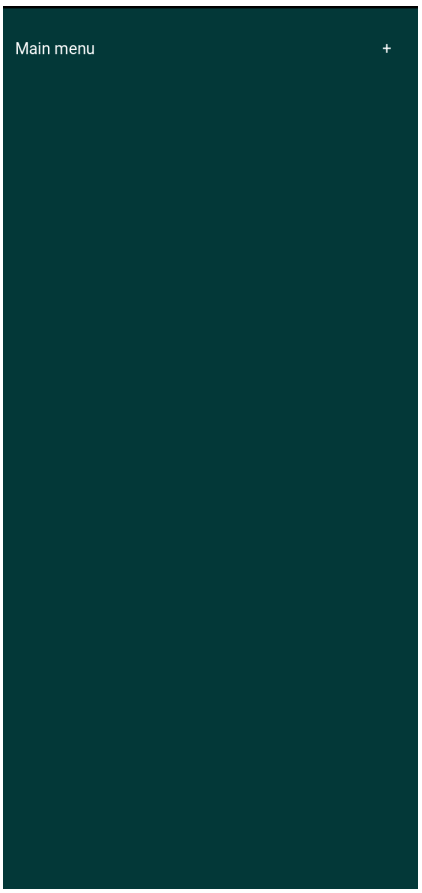

Рис. 2.4. Головний екран

Опинившись на головному екрані, користувач має вибір з двох кнопок: «Main menu» та «+».

Кнопка «+» дає змогу створювати «примітку» про один з лікарських засобів або заходів, вибираючи за допомогою кнопки «Choose variant» та кнопки «Choose *обрана опція*», яка створюється після вибору опції попередньої кнопки та містить свій набір опцій, вибір якої залежить від користувача. Після завершення вибору текстове поле заповнюється стислою інформацію про обраний варіант з його назвою. Також можна користувач може доповнити цю інформацію як забажає. Кнопка «X» дозволяє повернутися назад до головного екрану. Перераховане вище розташовано на новому сформованому після натискання кнопки «+» екрані. Усе, що описано в даному абзаці, можна побачити на рис.2.5 та 2.6.

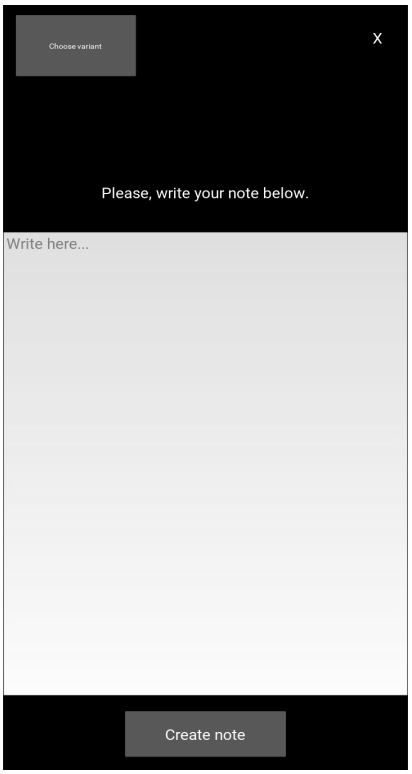

Рис. 2.5. Екран заповнювання «примітки»

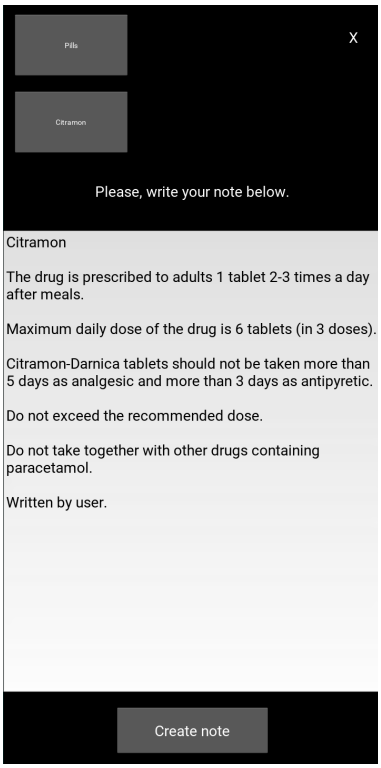

Рис. 2.6. Приклад заповненого екрану

Кнопка «Create note», як зрозуміло з назви, створює «примітку» на головному екрані. Примітка складається з трьох частин: текстове поле з інформацією, що ввів користувач; під ним розташовано міні-меню для вибору часу повідомлення та чекбокса, який вмикає або вимикає побудову повідомлень; над полем розташувалася кнопка, що дозволяє прибрати усі три складові примітки (рис.2.7).

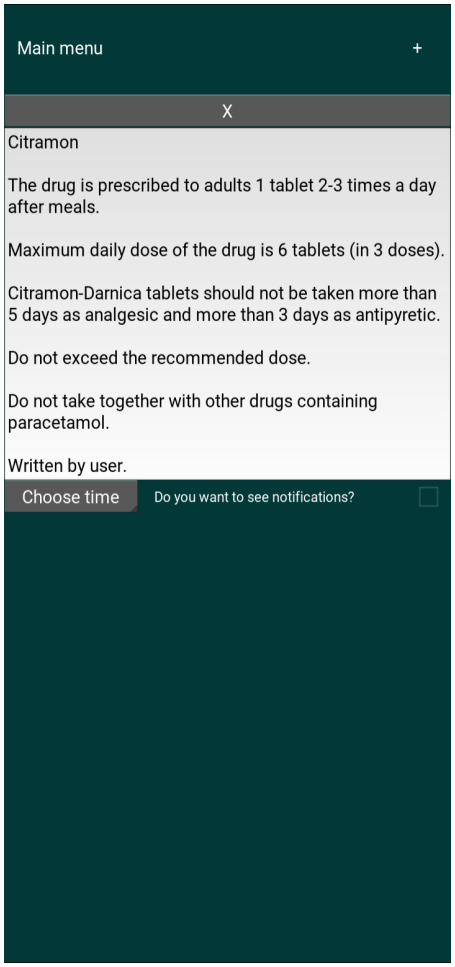

Рис. 2.7. Приклад створеної «примітки»

Кнопка «Main menu» відображає ім’я поточного екрану та дозволяє вибрати, на який екран треба переключитись, шляхом випадаючого списку при її натисканні. Список містить дві опції: «Main menu» та «Info menu».

Коли користувач вибирає опцію «Info menu», інформаційний екран замінює собою головний екран.

Мета інформаційного екрану – дозволяти користувачу переглядати інформацію за усіма хворобами, лікарськими засобами та процедурами. Реалізацію цього можна побачити на рис. 2.8.

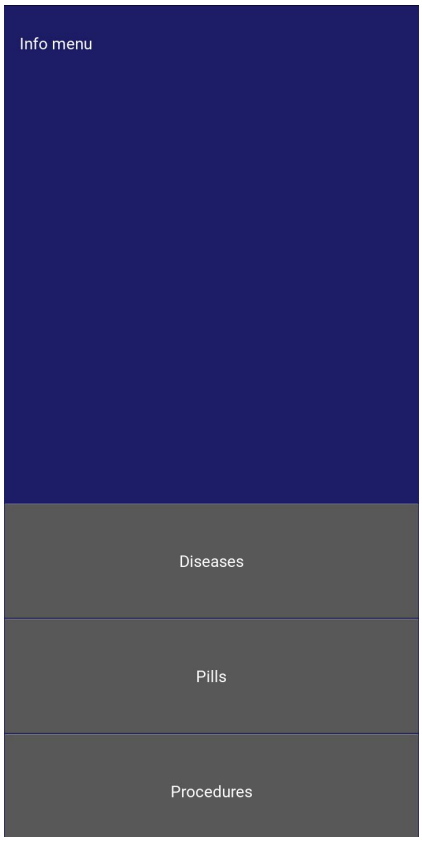

Рис. 2.8. Інформаційний екран

Кнопка «Go back» призначена для повернення на вибір назад та не створюється тільки на виборі між хворобами, таблетками та процедурами.

Аналогічно відбувається для вибору будь-якої кнопки, змінюється тільки кількість опцій.

Інформація щодо тої чи іншої опції скорочена. У майбутній розробці рекомендується написати парсер, який буде брати всю інформацію з відповідних джерел.

Дизайн додатку не є останнім варіантом, але є достатнім, щоб зобразити поточний функціонал програми.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Визначення трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 857;
2. коефіцієнт складності програми – 1,3;
3. коефіцієнт корекції програми в ході її розробки – 0,3;
4. годинна заробітна плата програміста– 116,5 грн.;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,5;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1;
7. вартість машино-годин ЕОМ – 0,15 грн./год.

Годинна заробітна плата програміста розраховувалася за даними середньої заробітної плати за останні три місяця (березень, квітень та травень 2023 року: https://www.work.ua/ru/salary-it/), яка дорівнює 20 500 грн. При 8-ми часовому робочому дні за місяць в середньому буде пророблено близько 176 годин, тому заробітна плата програміста в час буде дорівнювати 116,5 грн.

При розрахунку вартості машино-годин бралися витрати за електроенергію, споживану ЕОМ при роботі над програмою:

Витрати на електроенергію за час користування = 1,68 грн./кіловат (https://index.minfin.com.ua/tariff/electric/2021-10-01/);

потужність ЕОМ = 90 ват = 0,09 кіловат;

Вартість за споживання електроенергії при роботі = 1,68 * 0,09 = 0,15 грн./год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_O + t_U + t_A + t_N + t_{OTL} + t_D \text{ , людино-годин,} \quad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ,яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \times C \times (1 + p), \quad (3.2)$$

де q - передбачуване число операторів;

C - коефіцієнт складності програми;

p - коефіцієнт кореляції програми в ході її розробки.

$$Q = 857 * 1,3 * (1 + 0,3) = 1448,33;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q*B}{(75...85)*k}, \text{ людино-годин,} \quad (3.3)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього

опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи зданої спеціальності.

$$t_u = \frac{1448{,}33 * 1{,}5}{80*1} = 27{,}16 \text{ людино-годин},$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25)*k}, \text{ людино-годин}, \quad (3.4)$$

$$t_a = \frac{1448{,}33}{20*1} = 72{,}41 \text{ людино-годин}.$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)*k}, \text{ людино-годин}, \quad (3.5)$$

$$t_n = \frac{1448{,}33}{25*1} = 57{,}93 \text{ людино-годин},$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4...5)*k}, \text{ людино-годин}, \quad (3.6)$$

$$t_{\text{отл}} = \frac{1448{,}33}{4*1} = 362{,}08 \text{ людино-годин},$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1{,}5 * t_{\text{отл}}, \text{ людино-годин}, \quad (3.7)$$

$$t_{\text{отл}}^{k} = 1{,}5 * 362{,}08 = 543{,}12 \text{ людино-годин}.$$

Витрати праці на підготовку документації:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \text{ людино-годин}, \quad (3.8)$$

$$t_{\text{д}} = 96{,}56 + 72{,}42 = 168{,}98 \text{ людино-годин}.$$

де $t_{\text{др}}$ - трудомісткість підготовки матеріалів і рукопису.

$$t_{\text{др}} = \frac{Q}{(15...20)*k}, \text{ людино-годин}, \quad (3.9)$$

$$t_{\text{др}} = \frac{1448{,}33}{15*1} = 96{,}56 \text{ людино-годин}.$$

$t_{\text{до}}$ - трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 * t_{др}, \text{ людино-годин,} \quad (3.10)$$

$$t_{до} = 0{,}75 * 96{,}56 = 72{,}42 \text{ людино-годин,}$$

Розрахуємо трудомісткість розробки ПЗ:

$t = 50 + 27{,}16 + 72{,}41 + 57{,}93 + 362{,}08 + 168{,}98 = 738{,}56$ людино-годин.

За розрахунком, на розробку програми необхідно 738,56 людино-годин.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ *Кпо* включають витрати на заробітну плату виконавця програми *Зз/п* і витрат машинного часу, необхідного на налагодження програми на ЕОМ

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.} \quad (3.11)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t * С_{ПР}, \text{ грн.} \quad (3.12)$$

де: *t* - загальна трудомісткість, людино-годин;

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година

$$З_{ЗП} = 738{,}56 * 116{,}5 = 86\,042{,}24 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} * С_{мч}, \text{ грн,} \quad (3.13)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год.

$С_{мч}$ - вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 362{,}08 * 0{,}15 = 54{,}31 \text{ грн,}$$

Розрахуємо витрати на створення ПЗ:

$$K_{ПО} = 86\,042{,}24 + 54{,}31 = 86\,096{,}55 \text{ грн.}$$

Визначені в такий спосіб витрати на створення програмного забезпеченняє частиною одноразових капітальних витрат на створення АСУП.

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k * F_p}, \text{ міс}, \quad (3.14)$$

де B_k – число виконавців;

F_p – місячний фонд робочого часу (при 40-годинному робочому тижні F_p = 176 годин)

$$T = \frac{738,56}{1*176} = 4,19 \text{ міс.}$$

Висновок: трудомісткість розробки ПЗ дорівнює 738,56 людино-годин, на розробку ПЗ буде витрачено 86 096,55 грн, а очікуваний період створення ПЗ – 4,19 місяців.

ВИСНОВКИ

Метою створення додатку було впровадження цифрової мобільної системи, що дозволяє зберігати введену пацієнтом інформацію щодо лікарських засобів або заходів, що були приписані лікарем для відновлення здоров`я пацієнта.

В ході розробки продукту кваліфікаційної роботи було вирішено такі задачі:

– проаналізовано предметну галузь та галузь застосування;

– знайдено та проведено аналіз аналогів;

– сформульовано призначення розробки;

– розроблено план розробки додатку та його структури;

– обрано та досліджено технології, за якими буде розроблюватись додаток;

– сформовано функціональні характеристики та необхідні вимоги до них;

– визначено вимоги до технічних та програмних засобів;

– за планами розробки додатку та структури і за визначеними функціональними характеристиками розроблено програму;

– проведено тестування програми.

Розробка додатку виконувалась за допомогою мови програмування Python та фреймворку Kivy, а також модулів cryptography, pyjnius, datetime, os та json.

Під час проведення тестування додатку, було перевірено роботу функціоналу, відображення графічного інтерфейсу та захист даних від інформаційних загроз.

Розроблених функцій та дизайну недостатньо для того, щоб конкурувати з відомими аналогами, тому до майбутньої розробки пропонується додати: вдосконалення дизайну, вдосконалення захисту даних від доступу третіх осіб, дороблення інформаційної панелі (занесення актуальної інформації щодо усіх

лікарських засобів та заходів), додання більшої варіативності щодо існуючого функціоналу, створення функції обміну повідомленнями між пацієнтом та лікарем, створення бази даних для більш зручного зберігання даних та інше.

В ході роботи над економічною частиною було розраховано трудомісткість розробленого додатку, що становить 738,56 людино-годин, була визначена вартість розробки додатку, що становить 86 096,55 грн, а також встановлено час на розробку програми, що становить 4,19 місяців.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 7 Mobile App Security Risks and How to Mitigate Them [Електронний ресурс] // URL: https://www.cypressdatadefense.com/blog/mobile-app-security-risks/ (дата звернення: 10.06.2023);
2. The Top 10 Mobile App Security Threats That Put Your Data at Risk [Електронний ресурс] // URL: https://www.makeuseof.com/top-mobile-app-security-threats/ (дата звернення: 10.06.2023);
3. INSECURE DATA STORAGE IN MOBILE APPS [Електронний ресурс] // URL: https://cqr.company/web-vulnerabilities/insecure-data-storage-in-mobile-apps/ (дата звернення: 10.06.2023);
4. McAfee Mobile Threat Report [Електронний ресурс] // URL: https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf (дата звернення: 10.06.2023);
5. Pros and Cons of Python Programming Language [Електронний ресурс] // URL: https://www.pixelcrayons.com/blog/python-pros-and-cons/ (дата звернення: 10.06.2023);
6. A Programmers' Guide to Python: Advantages & Disadvantages [Електронний ресурс] // URL: https://www.linode.com/docs/guides/pros-and-cons-of-python/ (дата звернення: 10.06.2023);
7. What is Kivy? [Електронний ресурс] // URL: https://www.geeksforgeeks.org/what-is-kivy/ (дата звернення: 10.06.2023);
8. PyJNIus [Електронний ресурс] // URL: https://pypi.org/project/pyjnius/ (дата звернення: 10.06.2023);
9. Pros and Cons of Java | Advantages and Disadvantages of Java [Електронний ресурс] // URL: https://data-flair.training/blogs/pros-and-cons-of-java/ (дата звернення: 10.06.2023);
10. Advantages and Disadvantages of Java [Електронний ресурс] // URL: https://techvidvan.com/tutorials/pros-and-cons-of-java/ (дата звернення: 10.06.2023);

11. Fernet (symmetric encryption) using Cryptography module in Python [Електронний ресурс] // URL: https://www.geeksforgeeks.org/fernet-symmetric-encryption-using-cryptography-module-in-python/ (дата звернення: 10.06.2023);

12. Comparing the Usability of Cryptographic APIs [Електронний ресурс] // URL: https://www.cl.cam.ac.uk/~rja14/shb17/fahl.pdf (дата звернення: 10.06.2023);

13. Fernet (symmetric encryption) [Електронний ресурс] // URL: https://cryptography.io/en/latest/fernet/ (дата звернення: 10.06.2023);

14. What is fernet and when should you use it? [Електронний ресурс] // URL: https://www.comparitech.com/blog/information-security/what-is-fernet/ (дата звернення: 10.06.2023);

15. Python Datetime Module [Електронний ресурс] // URL: https://www.nbshare.io/notebook/914454993/Python-Datetime-Module/ (дата звернення: 10.06.2023);

16. Python datetime module [Електронний ресурс] // URL: https://www.geeksforgeeks.org/python-datetime-module/ (дата звернення: 10.06.2023);

17. datetime — Basic date and time types [Електронний ресурс] // URL: https://docs.python.org/3/library/datetime.html (дата звернення: 10.06.2023);

18. Pendulum: Probably The Best Python DateTime Library [Електронний ресурс] // URL: https://towardsdatascience.com/pendulum-one-of-the-most-useful-python-libraries-you-have-ever-seen-e2ecc365c8c0 (дата звернення: 10.06.2023);

19. OS Module in Python: All You Need to Know [Електронний ресурс] // URL: https://www.edureka.co/blog/os-module-in-python (дата звернення: 10.06.2023);

20. What Is Python's OS Module and How Do You Use It? [Електронний ресурс] // URL: https://www.makeuseof.com/python-os-system/ (дата звернення: 10.06.2023);

21. Python JSON: Encode(dumps), Decode(loads) & Read JSON File [Електронний ресурс] // URL: https://www.guru99.com/python-json.html (дата звернення: 10.06.2023);

22. Python JSON [Електронний ресурс] // URL: https://www.geeksforgeeks.org/python-json/ (дата звернення: 10.06.2023);

23. Rust Programming Language: Knows about Pros & Cons [Електронний ресурс] // URL: https://rlogicaltech.medium.com/rust-programming-language-knows-about-pros-cons-da92391aa465 (дата звернення: 10.06.2023);

24. Think Python: How to Think Like a Computer Scientist [Електронна книга] // URL: https://greenteapress.com/thinkpython2/html/index.html (дата звернення: 10.06.2023);

25. MedisafeApp [Електронний ресурс] // URL: https://medisafeapp.com/ (дата звернення: 10.06.2023);

26. Medication Reminder and Pill Tracker App – MyTherapy [Електронний ресурс] // URL: https://www.mytherapyapp.com/ (дата звернення: 10.06.2023);

27. Pillo: Medication Reminder [Електронний ресурс] // URL: https://play.google.com/store/apps/details?id=xyz.rtrvr.pillo&hl=en_US (дата звернення: 10.06.2023).

ДОДАТОК А

ЛІСТІНГ ПРОГРАМИ

```
#diplom.py
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.gridlayout import GridLayout
from kivy.uix.scrollview import ScrollView
from kivy.uix.checkbox import CheckBox
from kivy.uix.dropdown import DropDown
from kivy.uix.spinner import Spinner
from kivy.uix.button import Button
from kivy.uix.popup import Popup
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.core.window import Window
from kivy.uix.screenmanager import ScreenManager, Screen, FadeTransition
from jnius import autoclass
from cryptography.fernet import Fernet
from datetime import datetime
import os
import json

PythonActivity = autoclass('org.kivy.android.PythonActivity')
Notification = autoclass('android.app.Notification')
NotificationManager = autoclass('android.app.NotificationManager')
Context = autoclass('android.content.Context')

with open('saved_data/authorization_data/crypt_key.txt', 'rb') as file:
    crypt_key = file.read()

def encryption(text, keylock):
    key = Fernet(keylock)
    return key.encrypt(text)

def decryption(crypted_text, keylock):
    key = Fernet(keylock)
    return key.decrypt(crypted_text)


class MyLabel(Label):
    def __init__(self, **kwargs):
        super(MyLabel, self).__init__(**kwargs)
        self.bind(size=self.on_size)

    def on_size(self, instance, value):
        self.texture_update()
        self.texture_size = self.size
        self.text_size = self.size
```

```
class AuthorizationScreen(Screen):
    def __init__(self, **kwargs):
        super(AuthorizationScreen, self).__init__(**kwargs)

        self.app = App.get_running_app()

        self.root = FloatLayout()

        self.label1 = MyLabel(text = 'Enter your login below',
                              size_hint = (.4, .1),
                              pos_hint = {'x': .3, 'y': .74},
                              font_size = 22,
                              halign = 'left',
                              valign = 'bottom')
        self.label2 = MyLabel(text = 'Enter your password below',
                              size_hint = (.4, .1),
                              pos_hint = {'x': .3, 'y': .44},
                              font_size = 22,
                              halign = 'left',
                              valign = 'bottom')

        self.login_input = DynamicSizedTextInput(hint_text = 'Login',
                                         size_hint = (.4, None),
                                         pos_hint = {'x': .3, 'y': .7},
                                         multiline = False)
        self.password_input = DynamicSizedTextInput(password = True,
                                           hint_text = 'Password',
                                           size_hint = (.4, None),
                                           pos_hint = {'x': .3, 'y': .4},
                                           multiline = False)

        self.log_in_btn = Button(text = 'Log in',
                                 background_normal = '',
                                 background_color = (0, 1, 1, 1),
                                 color = (0, 0, 0, 1),
                                 size_hint = (.3, .05),
                                 pos_hint = {'x': .15, 'y': .25},
                                 on_release = self.check_data)
        self.register_btn = Button(text = 'Sign up',
                                   background_normal = '',
                                   background_color = (1, 0, 0, 1),
                                   size_hint = (.3, .05),
                                   pos_hint = {'x': .55, 'y': .25},
                                   on_release = self.change_to_registration)

        self.back_btn = Button(text = 'Back',
                               color = (1, 1, 1, 1),
                               background_normal = '',
                               background_color = (.16, .16, .16, 1),
                               size_hint = (.3, .05),
                               pos_hint = {'x': .15, 'y': .25},
```

```
                    on_release = self.change_to_log_in)
        self.save_data_btn = Button(text = 'Register',
                                    background_normal = '',
                                    background_color = (0, 1, 0, 1),
                                    size_hint = (.3, .05),
                                    pos_hint = {'x': .55, 'y': .25},
                                    on_release = self.save_data)

        self.login_register = DynamicSizedTextInput(hint_text = 'Login',
                                                    size_hint = (.4, None),
                                                    pos_hint = {'x': .3, 'y': .7},
                                                    multiline = False)
        self.password_register = DynamicSizedTextInput(password = True,
                                                       hint_text = 'Password',
                                                       size_hint = (.4, None),
                                                       pos_hint = {'x': .3, 'y': .4},
                                                       multiline = False)

        self.labels = []

        if len(os.listdir('saved_data/authorization_data')) > 1:
            self.change_to_log_in(self.back_btn)
        else:
            self.root.add_widget(self.label1)
            self.root.add_widget(self.label2)
            self.root.add_widget(self.login_register)
            self.root.add_widget(self.password_register)
            self.root.add_widget(self.save_data_btn)

        self.add_widget(TextInput(readonly = True,
                                  background_color = (.01, .22, .22, 1),
                                  size_hint = (1, 1),
                                  background_disabled_normal = '',
                                  disabled = True))
        self.add_widget(self.root)

    def change_to_log_in(self, instance):
        for child in self.root.children[:]:
            self.root.remove_widget(child)

        self.root.add_widget(self.label1)
        self.root.add_widget(self.label2)
        self.root.add_widget(self.login_input)
        self.root.add_widget(self.password_input)
        self.root.add_widget(self.log_in_btn)
        self.root.add_widget(self.register_btn)

    def change_to_registration(self, instance):
        for child in self.root.children[:]:
            self.root.remove_widget(child)

        self.root.add_widget(self.label1)
```

```
        self.root.add_widget(self.label2)
        self.root.add_widget(self.login_register)
        self.root.add_widget(self.password_register)
        self.root.add_widget(self.back_btn)
        self.root.add_widget(self.save_data_btn)

    def save_data(self, instance):
        self.remove_previous_labels()

        if self.login_register.text == '':
            label = MyLabel(text='Fill the login field, please',
                            color=(1, 0, 0, 1),
                            size_hint=(.2, .1),
                            pos_hint={'x': .25, 'y': .3},
                            font_size=22)
            self.root.add_widget(label)
            self.labels.append(label)
            return

        if self.password_register.text == '':
            label = MyLabel(text='Fill the password field, please',
                            color=(1, 0, 0, 1),
                            size_hint=(.2, .1),
                            pos_hint={'x': .55, 'y': .3},
                            font_size=22)
            self.root.add_widget(label)
            self.labels.append(label)
            return

        if len(self.password_register.text) < 8:
            label = MyLabel(text = 'Password must have at least 8 characters',
                            color=(1, 0, 0, 1),
                            size_hint=(.2, .1),
                            pos_hint={'x': .55, 'y': .3},
                            font_size=22)
            self.root.add_widget(label)
            self.labels.append(label)
            return

        crypted_login = encryption(self.login_register.text.encode(), crypt_key)
        crypted_password = encryption(self.password_register.text.encode(), crypt_key)
        data = {f'{crypted_login}': f'{crypted_login}',
                f'{crypted_password}': f'{crypted_password}'}

        with open(f'saved_data/authorization_data/{self.login_register.text}.json', 'w') as file:
            json.dump(data, file)

        self.change_to_log_in(instance)

    def remove_previous_labels(self):
        for label in self.labels:
            self.root.remove_widget(label)
```

```
        self.labels = []

    def check_data(self, instance):
        self.remove_previous_labels()

        try:
            with open(f'saved_data/authorization_data/{self.login_input.text}.json', 'r') as file:
                data = json.load(file)
                data_list = list(data.keys())
                temp_list = [data_list[0].split("""")[1],
                             data_list[1].split("""")[1]]
                decrypted_password = str(decryption(temp_list[1], crypt_key)).split("""")[1]
                if self.password_input.text == decrypted_password:
                    self.app.switch_screen('Main menu')
                else:
                    label = MyLabel(text='Invalid password',
                                    color=(1, 0, 0, 1),
                                    size_hint=(.2, .1),
                                    pos_hint={'x': .4, 'y': .3},
                                    font_size=22)
                    self.root.add_widget(label)
                    self.labels.append(label)
                    self.password_input.text = ''
                    return
        except(FileNotFoundError, json.JSONDecodeError):
            label = MyLabel(text='There is no account with this login',
                            color=(1, 0, 0, 1),
                            size_hint=(.2, .1),
                            pos_hint={'x': .4, 'y': .3},
                            font_size=22)
            self.root.add_widget(label)
            self.labels.append(label)
            self.login_input.text = ''
            self.password_input.text = ''
            return


class InfoScreen(Screen):
    def __init__(self, **kwargs):
        super(InfoScreen, self).__init__(**kwargs)

        self.app = App.get_running_app()

        self.root = BoxLayout(orientation = 'vertical',
                              spacing = 5)

        self.lay1 = FloatLayout(size_hint = (1, .1))
        self.lay2 = BoxLayout(orientation = 'vertical')

        self.screen_drop = DropDown()
        self.screen_btn = Button(text = self.name,
                                 size_hint = (.15, 1),
```

```
                    pos_hint = {'x': 0, 'y': 0},
                    background_color = (.11, .11, .4, 1),
                    background_normal = '',
                    on_release = self.screen_drop.open)

btn1 = Button(text = 'Main menu',
              size_hint_y = None,
              font_size = 16,
              on_release = self.screen_drop.dismiss)
btn2 = Button(text = 'Info menu',
              size_hint_y = None,
              font_size = 16,
              on_release = self.screen_drop.dismiss)

btn1.bind(on_release = lambda instance: self.app.switch_screen(instance.text))
btn2.bind(on_release = lambda instance: self.app.switch_screen(instance.text))

self.btn1 = Button(text = 'Diseases',
                   size_hint_y = None,
                   height = 200)
self.btn2 = Button(text = 'Pills',
                   size_hint_y = None,
                   height = 200)
self.btn3 = Button(text = 'Procedures',
                   size_hint_y = None,
                   height = 200)

self.btn1_1 = Button(text = 'Gastritis',
                     size_hint_y = None,
                     height = 200)
self.btn1_2 = Button(text = 'Migraine',
                     size_hint_y = None,
                     height = 200)
self.btn1_3 = Button(text = 'Sore throat',
                     size_hint_y = None,
                     height = 200)

self.btn2_1 = Button(text = 'Citramon',
                     size_hint_y = None,
                     height = 200)
self.btn2_2 = Button(text = 'Loperamid',
                     size_hint_y = None,
                     height = 200)
self.btn2_3 = Button(text = 'Septefril',
                     size_hint_y = None,
                     height = 200)
self.btn2_4 = Button(text = 'Paracetamol',
                     size_hint_y = None,
                     height = 200)
self.btn2_5 = Button(text = 'Pankreatin',
                     size_hint_y = None,
                     height = 200)
```

```
self.btn3_1 = Button(text = 'Fluorography', size_hint_y = None, height = 200)

self.label1_1 = MyLabel(halign = 'left',
                        valign = 'top',
                        text = 'Information about gastritis',
                        font_size = 22)
self.label1_2 = MyLabel(halign = 'left',
                        valign = 'top',
                        text = 'Information about migraine',
                        font_size = 22)
self.label1_3 = MyLabel(halign = 'left',
                        valign = 'top',
                        text = 'Information about sore throat',
                        font_size = 22)

self.label2_1 = MyLabel(halign = 'left',
                        valign = 'top',
                        text = 'Information about citramon',
                        font_size = 22)
self.label2_2 = MyLabel(halign = 'left',
                        valign = 'top',
                        text = 'Information about loperamid',
                        font_size = 22)
self.label2_3 = MyLabel(halign = 'left',
                        valign = 'top',
                        text = 'Information about septefril',
                        font_size = 22)
self.label2_4 = MyLabel(halign = 'left',
                        valign = 'top',
                        text = 'Information about paracetamol',
                        font_size = 22)
self.label2_5 = MyLabel(halign = 'left',
                        valign = 'top',
                        text = 'Information about pankreatin',
                        font_size = 22)

self.label3_1 = MyLabel(halign = 'left',
                        valign = 'top',
                        text = 'Information about fluorography',
                        font_size = 22)

self.list1 = [self.btn1_1, self.btn1_2, self.btn1_3]
self.list2 = [self.btn2_1, self.btn2_2, self.btn2_3, self.btn2_4, self.btn2_5]
self.list3 = [self.btn3_1]

self.list1_1 = [self.label1_1]
self.list1_2 = [self.label1_2]
self.list1_3 = [self.label1_3]

self.list2_1 = [self.label2_1]
self.list2_2 = [self.label2_2]
```

```
        self.list2_3 = [self.label2_3]
        self.list2_4 = [self.label2_4]
        self.list2_5 = [self.label2_5]

        self.list3_1 = [self.label3_1]

        self.back_btn = Button(text = 'Go back',
                               on_release = self.go_back,
                               size_hint = (1, None),
                               height = 100)
        self.saved_btns1 = []
        self.saved_btns2 = []

        self.btn1.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list1))
        self.btn2.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list2))
        self.btn3.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list3))

        self.btn1_1.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list1_1))
        self.btn1_2.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list1_2))
        self.btn1_3.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list1_3))

        self.btn2_1.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list2_1))
        self.btn2_2.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list2_2))
        self.btn2_3.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list2_3))
        self.btn2_4.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list2_4))
        self.btn2_5.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list2_5))

        self.btn3_1.bind(on_press = lambda instance: self.remove_and_add_widgets(instance,
self.list3_1))

        self.screen_drop.add_widget(btn1)
        self.screen_drop.add_widget(btn2)

        self.lay2.add_widget(self.btn1)
        self.lay2.add_widget(self.btn2)
        self.lay2.add_widget(self.btn3)

        self.lay1.add_widget(self.screen_btn)

        self.root.add_widget(self.lay1)
        self.root.add_widget(self.lay2)
```

```
        self.add_widget(TextInput(readonly = True,
                                  background_color = (.11, .11, .4, 1),
                                  size_hint = (1, 1),
                                  background_disabled_normal = '',
                                  disabled = True))
        self.add_widget(self.root)

    def remove_and_add_widgets(self, instance, children_list):
        if len(self.saved_btns1) > 0:
            for child in self.lay2.children[:]:
                self.saved_btns2.append(child)
                self.lay2.remove_widget(child)

            self.lay2.add_widget(self.back_btn)

            for child in children_list:
                self.lay2.add_widget(child)
        else:
            for child in self.lay2.children[:]:
                self.saved_btns1.append(child)
                self.lay2.remove_widget(child)

            self.lay2.add_widget(self.back_btn)

            for child in children_list:
                self.lay2.add_widget(child)

    def go_back(self, instance):
        if len(self.saved_btns1) > 0 and len(self.saved_btns2) > 0:
            for child in self.lay2.children[:]:
                self.lay2.remove_widget(child)

            for child in reversed(self.saved_btns2):
                self.lay2.add_widget(child)

            self.saved_btns2 = []
        else:
            for child in self.lay2.children[:]:
                self.lay2.remove_widget(child)

            for child in reversed(self.saved_btns1):
                self.lay2.add_widget(child)

            self.saved_btns1 = []


class DynamicSizedTextInput(TextInput):
    def __init__(self, **kwargs):
        super(DynamicSizedTextInput, self).__init__(**kwargs)
        self.size_hint_y = None
        if self.text != '':
```

```
            self.height = self.minimum_height
        else:
            self.height = 50

        self.bind(on_text = self.on_text_change)

    def on_text_change(self, instance):
        self.height = self.minimum_height


class MainScreen(Screen):
    def __init__(self, **kwargs):
        super(MainScreen, self).__init__(**kwargs)

        self.app = App.get_running_app()

        self.root = BoxLayout(orientation = 'vertical',
                              spacing = 5)

        self.output_fields = []

        self.lay1 = FloatLayout(size_hint = (1, .1))
        self.lay2 = ScrollView()
        self.lay_of_lay2 = GridLayout(cols = 1,
                                      size_hint_y = None)
        self.lay_of_lay2.bind(minimum_height=self.lay_of_lay2.setter('height'))
        self.lay_of_lay2.bind(minimum_width=self.lay_of_lay2.setter('width'))


        self.info_enter = FloatLayout()
        self.dropbtn2 = None

        self.add_btn = Button(text = '+',
                              size_hint = (.15, 1),
                              pos_hint = {'x': .85, 'y': 0},
                              background_color = (.01, .22, .22, 1),
                              background_normal = '',
                              on_release = self.add_note)
        self.add_btn.bind(on_press = self.fill_info_enter)
        self.screen_drop = DropDown()
        self.screen_btn = Button(text = self.name,
                                 size_hint = (.15, 1),
                                 pos_hint = {'x': 0, 'y': 0},
                                 background_color = (.01, .22, .22, 1),
                                 background_normal = '',
                                 on_release = self.screen_drop.open)

        btn1 = Button(text = 'Main menu',
                      size_hint_y = None,
                      font_size = 16,
                      on_release = self.screen_drop.dismiss)
        btn2 = Button(text = 'Info menu',
```

```
                size_hint_y = None,
                font_size = 16,
                on_release = self.screen_drop.dismiss)

        btn1.bind(on_release = lambda instance: self.app.switch_screen(instance.text))
        btn2.bind(on_release = lambda instance: self.app.switch_screen(instance.text))

        self.screen_drop.add_widget(btn1)
        self.screen_drop.add_widget(btn2)

        self.lay1.add_widget(self.add_btn)
        self.lay1.add_widget(self.screen_btn)
        self.lay2.add_widget(self.lay_of_lay2)

        self.root.add_widget(self.lay1)
        self.root.add_widget(self.lay2)

        self.add_widget(TextInput(readonly = True,
                                  background_color = (.01, .22, .22, 1),
                                  size_hint = (1, 1),
                                  background_disabled_normal = '',
                                  disabled = True))
        self.add_widget(self.root)

    def add_note(self, instance):
        self.enter_data.text = ''
        self.add_widget(self.info_enter)

    def on_text_change(self, instance):
        self.output_fields.remove(instance)
        self.output_fields.append(instance)

    def schedule_notification(self, instance, time):
        try:
            debugged_time = int(time)
            Clock.schedule_once(self.schedule_notification_callback,
self.get_notification_delay(debugged_time))
        except ValueError:
            pass


    def get_notification_delay(self, time):
        now = datetime.now()
        target_time = now.replace(hour = time,
                                  minute = 0,
                                  second = 0,
                                  microsecond = 0)

        if now > target_time:
            target_time = target_time.replace(day = target_time.day + 1)
        delta = target_time - now
        return delta.total_seconds()
```

```
    def schedule_notification_callback(self, dt):
        self.show_notification(dt)
        Clock.schedule_once(self.schedule_notification_callback, self.get_notification_delay())

    def show_notification(self, dt):
        notification = Notification()
        notification.icon = PythonActivity.mActivity.getApplicationInfo().icon
        notification.tickerText = 'Notification'
        notification.contentTitle = 'Important notification'
        notification.contentText = 'Time to take a pill!'
        notification.flags |= Notification.FLAG_AUTO_CANCEL
        notification_manager                                                    =
PythonActivity.mActivity.getSystemService(Context.NOTIFICATION_SERVICE)
        notification_manager.notify(1, notification)

    def create_note(self, instance):
        output_field = DynamicSizedTextInput()
        X_label_button = Button(text = 'X',
                                size_hint_y = None,
                                height = 50,
                                on_release = self.delete_note)
        notify_field = FloatLayout(size_hint_y = None,
                                   height = 50)

        time_chooser = Spinner(text = 'Choose time',
                               values = ('9:00', '12:00', '15:00', '18:00', '21:00'),
                               size_hint = (.3, 1),
                               pos_hint = {'x': .15})
        check_box = CheckBox(size_hint = (.1, 1),
                             pos_hint = {'x': .9},
                             on_release     =     lambda     instance:     self.schedule_notification(instance,
time_chooser.text.split(':')[0])
        notification_label = MyLabel(text = 'Do you want to see notifications?',
                                     size_hint = (.3, 1),
                                     pos_hint = {'x': .55},
                                     font_size = 22)

        notify_field.add_widget(check_box)
        notify_field.add_widget(time_chooser)
        notify_field.add_widget(notification_label)

        output_field.text = self.enter_data.text

        self.lay_of_lay2.add_widget(X_label_button)
        self.lay_of_lay2.add_widget(output_field)
        self.lay_of_lay2.add_widget(notify_field)

        output_field.bind(minimum_height = output_field.setter('height'))
        output_field.bind(on_text = self.on_text_change)
        self.lay_of_lay2.bind(minimum_height = self.lay_of_lay2.setter('height'))
        self.lay_of_lay2.bind(minimum_width = self.lay_of_lay2.setter('width'))
```

```
        self.output_fields.append(output_field)

        self.remove_widget(self.info_enter)

    def delete_note(self, instance):
        index = self.lay_of_lay2.children.index(instance)

        if index > 0:
            related_field = self.lay_of_lay2.children[index - 1]

            self.lay_of_lay2.remove_widget(related_field)
            self.lay_of_lay2.bind(minimum_height=self.lay_of_lay2.setter('height'))
            self.lay_of_lay2.bind(minimum_width=self.lay_of_lay2.setter('width'))

            self.output_fields.remove(related_field)

        self.lay_of_lay2.remove_widget(instance)

    def fill_info_enter(self, instance):
        self.drop = DropDown()
        self.drop.bind(on_dismiss = self.second_drop_creation)
        self.enter_data = TextInput(hint_text = 'Write here...',
                                    size_hint = (1, .6),
                                    pos_hint = {'y': .1})
        self.dropbtn = Button(text = 'Choose variant',
                              size_hint = (.3, .08),
                              pos_hint = {'x': .03, 'y': .9},
                              font_size = 14)

        pills_btn = Button(text='Pills',
                           size_hint_y = None,
                           height = 50,
                           font_size = 14)
        procedures_btn = Button(text = 'Procedures',
                                size_hint_y = None,
                                height = 50,
                                font_size = 14)

        pills_btn.bind(on_press = self.drop_selector)
        procedures_btn.bind(on_press = self.drop_selector)

        self.drop.add_widget(pills_btn)
        self.drop.add_widget(procedures_btn)

        self.info_enter.add_widget(TextInput(readonly = True,
                                             background_color = (0, 0, 0, 1),
                                             disabled = True))
        self.info_enter.add_widget(self.dropbtn)
        self.info_enter.add_widget(Label(text = 'Please, write your note below.',
                                         size_hint = (1, .1),
                                         pos_hint = {'y': .7}))
```

```
        self.info_enter.add_widget(self.enter_data)
        self.info_enter.add_widget(Button(text = 'Create note',
                                  on_release = self.create_note,
                                  size_hint = (.4, .06),
                                  pos_hint = {'x': .3, 'y': .02}))
        self.info_enter.add_widget(Button(text = 'X',
                                  background_color = (0, 0, 0, 0),
                                  size_hint = (.15, .1),
                                  pos_hint = {'x': .85, 'y': .9},
                                  on_release = self.remove_info_enter))

        self.dropbtn.bind(on_release = self.drop.open)

    def remove_info_enter(self, instance):
        self.remove_widget(self.info_enter)

    def drop_selector(self, instance):
        self.dropbtn.text = instance.text
        self.drop.dismiss()

    def drop2_selector(self, instance):
        if instance.text == 'Citramon':
            with open('data/Pills/Citramon.txt', 'r') as file:
                for line in file:
                    self.enter_data.text = self.enter_data.text + line
        if instance.text == 'Loperamid':
            with open('data/Pills/Loperamid.txt', 'r') as file:
                for line in file:
                    self.enter_data.text = self.enter_data.text + line
        if instance.text == 'Septefril':
            with open('data/Pills/Septefril.txt', 'r') as file:
                for line in file:
                    self.enter_data.text = self.enter_data.text + line
        if instance.text == 'Paracetamol':
            with open('data/Pills/Paracetamol.txt', 'r') as file:
                for line in file:
                    self.enter_data.text = self.enter_data.text + line
        if instance.text == 'Pankreatin':
            with open('data/Pills/Pankreatin.txt', 'r') as file:
                for line in file:
                    self.enter_data.text = self.enter_data.text + line
        if instance.text == 'Fluorography':
            with open('data/Procedures/Fluorography.txt', 'r') as file:
                for line in file:
                    self.enter_data.text = self.enter_data.text + line

        self.dropbtn2.text = instance.text

        self.drop2.dismiss()

    def second_drop_creation(self, instance):
        if self.dropbtn.text == 'Pills':
```

```
        if self.dropbtn2 is not None:
            self.dropbtn2.unbind(on_release = self.drop2.open)
            self.info_enter.remove_widget(self.dropbtn2)

        self.dropbtn2 = Button(text = 'Choose pill',
                               size_hint = (.3, .08),
                               pos_hint = {'x': .03, 'y': .8},
                               font_size = 14)
        self.drop2 = DropDown()

        self.pill1 = Button(text = 'Citramon',
                            size_hint_y = None,
                            height = 50,
                            font_size = 14)
        self.pill2 = Button(text = 'Loperamid',
                            size_hint_y = None,
                            height = 50,
                            font_size = 14)
        self.pill3 = Button(text = 'Septefril',
                            size_hint_y = None,
                            height = 50,
                            font_size = 14)
        self.pill4 = Button(text = 'Paracetamol',
                            size_hint_y = None,
                            height = 50,
                            font_size = 14)
        self.pill5 = Button(text = 'Pankreatin',
                            size_hint_y = None,
                            height = 50,
                            font_size = 14)

        self.drop2.add_widget(self.pill1)
        self.drop2.add_widget(self.pill2)
        self.drop2.add_widget(self.pill3)
        self.drop2.add_widget(self.pill4)
        self.drop2.add_widget(self.pill5)

        self.info_enter.add_widget(self.dropbtn2)

        self.pill1.bind(on_press = self.drop2_selector)
        self.pill2.bind(on_press = self.drop2_selector)
        self.pill3.bind(on_press = self.drop2_selector)
        self.pill4.bind(on_press = self.drop2_selector)
        self.pill5.bind(on_press = self.drop2_selector)

        self.dropbtn2.bind(on_release = self.drop2.open)

    elif self.dropbtn.text == 'Procedures':
        if self.dropbtn2 is not None:
            self.dropbtn2.unbind(on_release = self.drop2.open)
            self.info_enter.remove_widget(self.dropbtn2)
```

```
        self.dropbtn2 = Button(text = 'Choose procedure',
                               size_hint = (.3, .08),
                               pos_hint = {'x': .03, 'y': .8},
                               font_size = 14)
        self.drop2 = DropDown()

        self.procedure1 = Button(text = 'Fluorography',
                                 size_hint_y = None,
                                 height = 50,
                                 font_size = 14)

        self.drop2.add_widget(self.procedure1)

        self.info_enter.add_widget(self.dropbtn2)

        self.procedure1.bind(on_press = self.drop2_selector)

        self.dropbtn2.bind(on_release = self.drop2.open)

    def add_label(self, text):
        field = DynamicSizedTextInput(text = text)
        X_label_button = Button(text = 'X',
                                size_hint_y = None,
                                height = 50,
                                on_release = self.delete_note)
        notify_field = FloatLayout(size_hint_y = None,
                                   height = 50)

        time_chooser = Spinner(text = 'Choose time',
                               values = ('9:00', '12:00', '15:00', '18:00', '21:00'),
                               size_hint = (.3, 1),
                               pos_hint = {'x': .15})
        check_box = CheckBox(size_hint = (.1, 1),
                             pos_hint = {'x': .9},
                             on_release = lambda instance: self.schedule_notification(instance,
time_chooser.text.split(':')[0])
        notification_label = MyLabel(text = 'Do you want to see notifications?',
                                     size_hint = (.3, 1),
                                     pos_hint = {'x': .55},
                                     font_size = 22)

        notify_field.add_widget(check_box)
        notify_field.add_widget(time_chooser)
        notify_field.add_widget(notification_label)

        self.output_fields.append(field)

        self.lay_of_lay2.add_widget(X_label_button)
        self.lay_of_lay2.add_widget(field)
        self.lay_of_lay2.add_widget(notify_field)

        field.bind(minimum_height = field.setter('height'))
```

```
        self.lay_of_lay2.bind(minimum_height = self.lay_of_lay2.setter('height'))
        self.lay_of_lay2.bind(minimum_width = self.lay_of_lay2.setter('width'))

    def save_data(self, instance):
        data = {'labels': [label.text for label in self.output_fields]}

        with open('saved_data/saved_data.json', 'w') as file:
            json.dump(data, file)

    def load_data(self):
        try:
            with open('saved_data/saved_data.json', 'r') as file:
                data = json.load(file)

                for text in data['labels']:
                    self.add_label(text)
        except(FileNotFoundError, json.JSONDecodeError):
            pass


class MyApp(App):
    def build(self):
        self.sm = ScreenManager(transition = FadeTransition())

        self.main = MainScreen(name = 'Main menu')
        self.info = InfoScreen(name = 'Info menu')
        self.authorization = AuthorizationScreen(name = 'Authorization')

        self.sm.add_widget(self.authorization)
        self.sm.add_widget(self.main)
        self.sm.add_widget(self.info)


        self.bind(on_stop = self.main.save_data)

        self.main.load_data()

        return self.sm

    def switch_screen(self, screen_name):
        self.sm.current = screen_name


if __name__ == '__main__':
    MyApp().run()
```

ДОДАТОК Б

ВІДГУК

на кваліфікаційну роботу бакалавра

на тему:

«Розробка мобільного додатку для відстеження амбулаторного лікування пацієнта за допомогою мови програмування Python та фреймворку Kivy»

студента групи 122-19-2 Зюзіна Олега Сергійовича

Керівник економічного розділу
кваліфікаційної роботи
зав. каф. ПЕП та ПУ, проф. **Вагонова О.Г.**

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
Пояснювальна_записка_Зюзін_О_С.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Пояснювальна_записка_Зюзін_О_С.pdf	Пояснювальна записка до кваліфікаційної роботи у форматі PDF
Програма	
Diplom.rar	Архів, що містить код програми та необхідні структурні елементи
Презентація	
Презентація_Зюзін_О_С.ppt	Презентація кваліфікаційної роботи