НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра
спеціальності 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення розподілених систем»

на тему: «Мобільний модуль додатку «Автоматизоване робоче місце енергоменеджера»

Виконала:
студентка IV курсу, групи ТІ-71
Дробаха Маргарита Борисівна _____

Керівник:
доцент, к.т.н.,
Ковальчук Артем Михайлович _____

Рецензент:
головний енергоменеджер КПІ ім. Ігоря Сікорського, к.т.н.,
Шевченко Олена Миколаївна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

спеціальності 121 «Інженерія програмного забезпечення»

освітня програма «Інженерія програмного забезпечення розподілених систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2021р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Дробасі Маргариті Борисівні_____
(прізвище, ім’я, по батькові)

1. Тема роботи «Мобільний модуль додатку «Автоматизоване робоче місце енергоменеджера»
керівник роботи Ковальчук Артем Михайлович, к.т.н., доцент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від ”____” травня 2021р. № _____

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи форма реалізації — мобільний застосунок з використанням фреймворку Flutter на мові Dart. Середовище розробки — Android Studio.
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) розробити програмне забезпечення — мобільний застосунок для відображення, створення, редагування інформації по газо-, водо- та електропостачанню.
5. Орієнтовний перелік ілюстративного матеріалу
Діаграма класів, діаграма прецедентів, інтерфейс системи, віджети фреймворку, ієрархія віджетів, архітектури мобільних за стосунків для різних операційних систем, ілюстрації роботи інших систем.
6. Дата видачі завдання ”___”_____202___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	9.10.20	
2.	Вивчення та аналіз задачі	14.04.21	
3.	Розробка архітектури та загальної структури системи	16.04.21	
4.	Розробка структур окремих підсистем	18.04.21	
5.	Програмна реалізація системи	24.04.21	
6.	Оформлення пояснювальної записки	04.06.21	
7.	Захист програмного продукту	12.05.21	
8.	Передзахист	26.05.21	
9.	Захист	16.06.21	

Студент _____ _____

(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____

(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою роботи є розробка мобільного додатку, що надає можливість переглядати, змінювати та вносити дані по електро-, водо- та газолічильникам, переглядати актуальну статистику кожного лічильника, переглядати, завантажувати документацію та договори. В рамках даної дипломної роботи було розглянуто існуючі програмні рішення, оцінено їх сильні та слабкі сторони, а також специфіку вирішення проблем. Згідно зі специфікою роботи було розроблено алгоритми та підходи до відображання, внесення та зміни інформації. На основі заданого технічного завдання було створено архітектуру системи, розбито на модулі для коректної взаємодії системи та для можливості зміни та масштабування системи.

Дана дипломна робота обсягом у 59 сторінок, містить 3 додатки та 9 посилань. Також наведено 31 рисунок.

Ключові слова: енергоменеджмент, мобільний додаток, Android, iOS, Flutter, Dart, JSON, REST API, статистика.

ABSTRACT

The purpose of the work is to develop a mobile application that provides the ability to view, change and enter data on electricity, water and gas meters, view current statistics of each meter, view, download documentation and contracts. In the framework of this thesis, the existing software solutions were considered, their strengths and weaknesses were assessed, as well as the specifics of problem solving. According to the specifics of the work, algorithms and approaches to display, enter and change information were developed. Based on the given technical task, the system architecture was created, divided into modules for the correct interaction of the system and for the possibility of changing and scaling the system.

This thesis of 59 pages, contains 3 appendices and 9 references. There are also 31 figures.

Keywords: energy management, mobile application, Android, iOS, Flutter, Dart, JSON, REST API, statistics.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІТ — інформаційні технології.

ПЗ – програмне забезпечення.

REST (Representational state transfer) — підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів.

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

IDE (Integrated Development Environment) — інтегроване середовище розробки.

Фреймворк — інфраструктура програмних рішень, що полегшує розробку складних систем.

Мобільний додаток — програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях.

UML (Unified Modeling Language) — уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування.

JSON (JavaScript Object Notation) — текстовий формат обміну даними між комп'ютерами.

HTTP (HyperText Transfer Protocol) — протокол передачі даних, що використовується в комп'ютерних мережах.

ВСТУП

На сьогоднішній день існує проблема в ефективному використанні енергоресурсів та налагодженні їх автоматизованого обліку. Для моніторингу приміщень будівель НТУУ ‘‘КПІ ім. Ігоря Сікорського’’ на основі даних, які вносяться енергоменеджерами факультетів, може бути запроваджена веб система.

У сучасному світі постала проблема обмеженості енергоресурсів та водоресурсів для забезпечення всіх потреб людства. Енергозбереження та водозбереження є одними із шляхів вирішення цієї проблеми. Для раціонального збереження ресурсів та покращення енергоефективності необхідний постійний контроль за витраченими тепло-, електро-, водо-, газоресурсами.

Наразі така система буде доречною для будь-якого приміщення. Можливість відстежувати втрати енергоресурсів та аналізувати статистику дає змогу продумувати ефективний план заходів по енергозбереженню та, відповідно, економії.

Для зручності та автоматизації роботи з вхідними даними необхідне створення спеціального мобільного додатку, який буде допомагати вносити та актуалізувати дані, показувати необхідну статистику та кінцеві показники. З його допомогою внесення необхідних даних в загальну систему буде можливо будь-якою відповідальною особою, в якої є телефон з необхідним програмним забезпеченням. Зникає необхідність запам’ятовувати дані або їх записувати вручну. З мобільного додатку можливо переглянути внесені дані, статистику по енергоспоживанню. Також завантажити та переглянути документацію та договори по енергоспоживанню.

Для зручності користування будь-якою особою інтерфейс додатку повинен бути спрощеним, інтуїтивно зрозумілим та зручним у використанні. Мобільний додаток буде доповненням основної системи [7].

1. ЗАДАЧА СТВОРЕННЯ МОБІЛЬНОГО ДОДАТКУ ЕНЕРГОМЕНЕДЖЕРА

Основною метою роботи є розробка мобільного модулю додатку ‘‘Автоматизоване робоче місце енергоменеджера’’ для внесення, перегляду даних лічильників, документів та статистики по енерго- та водоспоживанню.

Основні задачі дипломної роботи:

1. Проаналізувати існуючі програмні рішення розв’язання задач енергоменеджмента;
2. Визначити переваги та недоліки застосованої архітектури та функціоналу в існуючих системах;
3. Розробити архітектуру програмного забезпечення мобільного додатку, яка буде враховувати проаналізовані переваги та недоліки архітектур існуючих систем;
4. Розробити основний інтерфейс системи мобільного додатку, схожий на інтерфейс веб-додатку;
5. Реалізувати зв’язок з існуючим сервером та мікросервісами, розробленими для цієї системи;
6. Протестувати розроблену систему, використовуючи різні вхідні та вихідні дані, на різних операційних системах та пристроях.

1.1 Аналіз існуючих програмних рішень

Необхідно проаналізувати існуючі програмні рішення задач енергоменеджмента. З’ясувати їх архітектури, створений функціонал, використані бібліотеки, мови програмування, інтерфейс систем. Знайти їх недоліки та переваги.

З врахуванням цієї інформації створити власну систему.

1.1 Створення архітектури системи

Найперше необхідно створити діаграму прецедентів згідно з наданим технічним завданням. Обробити різні варіанти роботи системи та взаємодії користувача з нею.

Надалі створити діаграму класів для розуміння того, які класи необхідно створити та яким чином вони будуть взаємодіяти між собою.

1.2 Створення інтерфейсу мобільного додатку

Інтерфейс системи повинен бути інтуїтивно зрозумілим, простим у використанні, не перенавантаженим. Також додаток повинен працювати на операційних системах Android та iOS, тобто бути кросплатформним. Для цього необхідно використовувати фреймворк Flutter та мову програмування Dart.

Для того, щоб інтерфейс мобільного додатку мінімально відрізнявся від інтерфейсу веб-додатку, необхідно використовувати вже створені дизайни веб-додатку, розміщені на платформі Figma.

1.3 Підключення серверної частини

Необхідно використовувати архітектуру RESTful API, яка забезпечує метод взаємодії між двома системами. Використовуючи документацію серверної частини, створену фреймворком Swagger, розробити функції, що надсилають запити GET, POST, DELETE та PATCH.

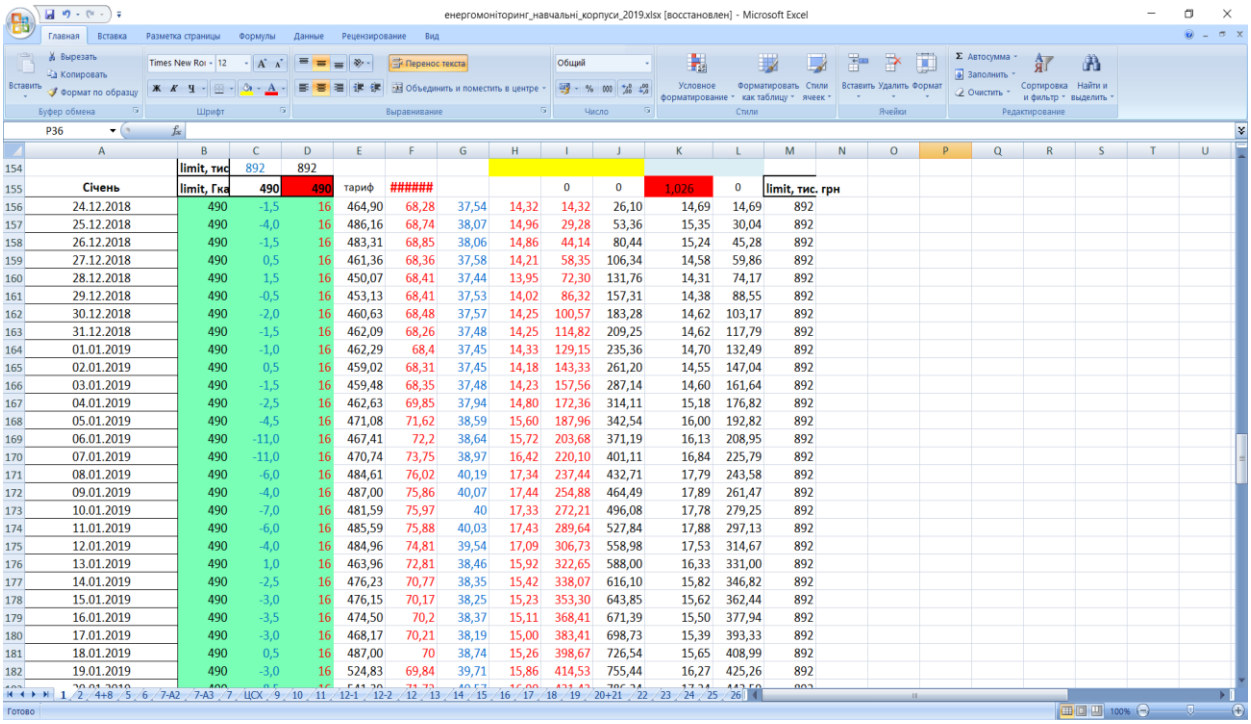
Для того, щоб система не була перенавантажена, для кожного модуля розробити сервіс, та один клас, що взаємодіє безпосередньо з сервером, та який надсилає на нього необхідні запити, та, відповідно, отримує інформацію. Такий спосіб розробки ПЗ забезпечує масштабування всієї системи, можливість змінювати окремі елементи, або навіть весь сервер.

2. ОПИС ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ПРИ РОЗВ'ЯЗАННІ ЗАДАЧ ЕНЕРГОМЕНЕДЖМЕНТУ

Для розв'язування існуючих задач енергоменеджменту використовуються електронні таблиці, програмне забезпечення для відображення статистичних даних, такі як: Microsoft Excel, 1C, Dashboard.

2.1 Електронні таблиці Microsoft Excel

Електронні таблиці Microsoft Excel — це інструмент, який дозволяє вводити кількісні дані в електронну таблицю (рисунок 2.1) для застосування одного чи багатьох математичних обчислень [8]. Ці обчислення в кінцевому рахунку перетворюють ці кількісні дані в інформацію.



	A	B	C	D	E	F	G	H	I	J	K	L	M
154		limit, тис	892	892									
155	Січень	limit, Гка	490	490	тариф	######			0	0	1,026	0	limit, тис. грн
156	24.12.2018	490	-1,5	16	464,90	68,28	37,54	14,32	14,32	26,10	14,69	14,69	892
157	25.12.2018	490	-4,0	16	486,16	68,74	38,07	14,96	29,28	53,36	15,35	30,04	892
158	26.12.2018	490	-1,5	16	483,31	68,85	38,06	14,86	44,14	80,44	15,24	45,28	892
159	27.12.2018	490	0,5	16	461,36	68,36	37,58	14,21	58,35	106,34	14,58	59,86	892
160	28.12.2018	490	1,5	16	450,07	68,41	37,44	13,95	72,30	131,76	14,31	74,17	892
161	29.12.2018	490	-0,5	16	453,13	68,41	37,53	14,02	86,32	157,31	14,38	88,55	892
162	30.12.2018	490	-2,0	16	460,63	68,48	37,57	14,25	100,57	183,28	14,62	103,17	892
163	31.12.2018	490	-1,5	16	462,09	68,26	37,48	14,25	114,82	209,25	14,62	117,79	892
164	01.01.2019	490	-1,0	16	462,29	68,4	37,45	14,33	129,15	235,36	14,70	132,49	892
165	02.01.2019	490	0,5	16	459,02	68,31	37,45	14,18	143,33	261,20	14,55	147,04	892
166	03.01.2019	490	-1,5	16	459,48	68,35	37,48	14,23	157,56	287,14	14,60	161,64	892
167	04.01.2019	490	-2,5	16	462,63	69,85	37,94	14,80	172,36	314,11	15,18	176,82	892
168	05.01.2019	490	-4,5	16	471,08	71,62	38,59	15,60	187,96	342,54	16,00	192,82	892
169	06.01.2019	490	-11,0	16	467,41	72,2	38,64	15,72	203,68	371,19	16,13	208,95	892
170	07.01.2019	490	-11,0	16	470,74	73,75	38,97	16,42	220,10	401,11	16,84	225,79	892
171	08.01.2019	490	-6,0	16	484,61	76,02	40,19	17,34	237,44	432,71	17,79	243,58	892
172	09.01.2019	490	-4,0	16	487,00	75,86	40,07	17,44	254,88	464,49	17,89	261,47	892
173	10.01.2019	490	-7,0	16	481,59	75,97	40	17,33	272,21	496,08	17,78	279,25	892
174	11.01.2019	490	-6,0	16	485,59	75,88	40,03	17,43	289,64	527,84	17,88	297,13	892
175	12.01.2019	490	-4,0	16	484,96	74,81	39,54	17,09	306,73	558,98	17,53	314,67	892
176	13.01.2019	490	1,0	16	463,96	72,81	38,46	15,92	322,65	588,00	16,33	331,00	892
177	14.01.2019	490	-2,5	16	476,23	70,77	38,35	15,42	338,07	616,10	15,82	346,82	892
178	15.01.2019	490	-3,0	16	476,15	70,17	38,25	15,23	353,30	643,85	15,62	362,44	892
179	16.01.2019	490	-3,5	16	474,50	70,2	38,37	15,11	368,41	671,39	15,50	377,94	892
180	17.01.2019	490	-3,0	16	468,17	70,21	38,19	15,00	383,41	698,73	15,39	393,33	892
181	18.01.2019	490	0,5	16	487,00	70	38,74	15,26	398,67	726,54	15,65	408,99	892
182	19.01.2019	490	-3,0	16	524,83	69,84	39,71	15,86	414,53	755,44	16,27	425,26	892

Рисунок 2.1 — Таблиця даних в Microsoft Excel

Інформація, оброблена в Excel, може бути використана для прийняття рішень як у професійному, так і в особистому контексті. Наприклад, співробітники можуть використовувати Excel, щоб визначити, скільки інвентарю купити для продавця одягу, скільки ліків вводити пацієнту чи скільки грошей витратити, щоб не виходити за рамки бюджету.

Ведення статистики в Excel не є найкращим рішенням, адже необхідно постійно обмінюватись одним файлом між різними відповідальними особами. Ці файли є достатньо важкими. Введення даних не є швидким, легко помилитись та не помітити цього. Тому Excel не вигідно використовувати для подібних цілей, це не є ефективним рішенням.

2.2 Програма 1С: Підприємство

Програма 1С: Підприємство — це універсальна програмна система, призначена для автоматизації обліку, планування та управління підприємством, а також для вирішення особистих завдань [2].

Робота поділяється на дві часові частини: налаштування (розробка конфігурації) та робота користувача (операції обліку або розрахунку).

На етапі розробки конфігурації додаток розробляється відповідно до особливостей підприємства. Сюди входить проектування структури об’єкта та способів перегляду об’єктів, дизайн інтерфейсу та дизайн доступних прав користувача.

Цей етап також включає адміністративні процедури: ведення списку користувачів, налаштування параметрів Infobase, налаштування журналу подій, оновлення конфігурації тощо.

Користувачі працюють з інформаційними базами, які працюють у режимі 1С: Підприємство. Це коли програмне забезпечення фактично працює для досягнення своїх цілей: введення даних, формування звітів, заплановані розрахунки тощо [1].

Користувач працює з даними, тоді як структура цих даних та алгоритми обробки даних визначаються на стадії розробки конфігурації.

Для того, щоб вести статистику та аналітику в 1С (рисунок 2.2) необхідна допомога програміста, який буде прописувати програмний код, оптимізувати, перевіряти систему. Без підтримки програміста користувачі не зможуть вести облік та аналізувати дані.

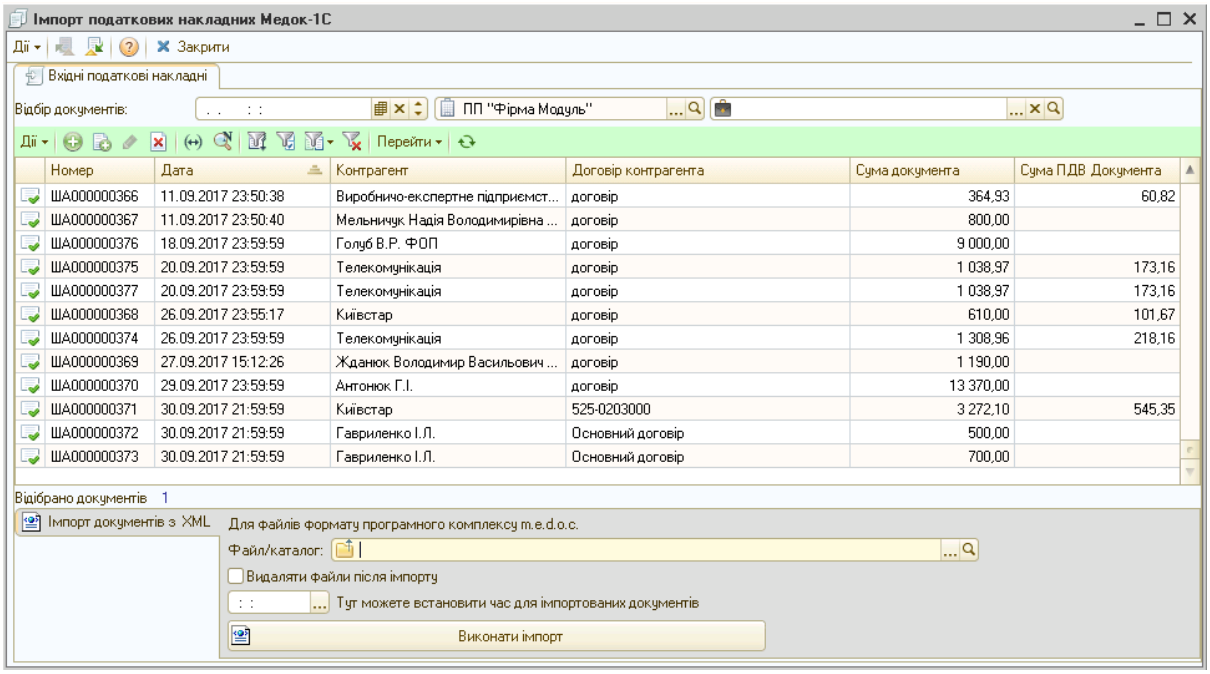


Рисунок 2.2 — Таблиця даних в 1С: Підприємство

Таким чином необхідне залучення додаткової робочої сили, що не є економним та ефективним рішенням для Університету.

2.3 Система Dashboard

Система Dashboard — це інструмент управління інформацією, який візуально відстежує, аналізує та відображає ключові показники ефективності (KPI), метрики та ключові точки даних для моніторингу стану бізнесу, підрозділу або конкретного процесу. Їх можна налаштувати відповідно до конкретних потреб відділу та компанії. За лаштунками інформаційна панель підключається до ваших

файлів, вкладень, служб та API, але на поверхні відображаються всі ці дані у вигляді таблиць, лінійних діаграм, стовпчастих діаграм та вимірювальних приладів. Інформаційна панель даних — це найефективніший спосіб відстеження декількох джерел даних, оскільки вона забезпечує центральне місце для бізнесу для моніторингу та аналізу ефективності.

Аналітичні інформаційні панелі, як правило, призначені для того, щоб допомогти особам, які приймають рішення та керівникам встановлювати цілі та розуміти, що і чому щось трапилось. Аналітична інформаційна панель робить це на основі аналізу даних, зібраних протягом періоду часу, визначеного користувачем (тобто минулого місяця, кварталу чи року).

Дані візуалізуються (рисунок 2.3) на інформаційній панелі як таблиці, лінійні діаграми, стовпчасті діаграми та вимірювальні прилади, щоб користувачі могли відстежувати стан свого бізнесу на основі тестів та цілей. Інформаційні панелі даних містять необхідні дані для розуміння, моніторингу та вдосконалення вашого бізнесу за допомогою візуальних зображень.

Рисунок 2.3 — Таблиця даних в Dashboard

Така система є достатньо ефективною, їх існує достатньо багато в інтернет просторі. Але вони мають значний недолік для їх ведення. Велика система буде дуже дорогою та складною для ведення. При будь-яких системних змінах необхідно звертатись до адміністраторів системи та чекати їх відповідь.

Найкращим засобом ведення задач енергоменеджменту є створення власних систем, які будуть повністю напрямлені на власну систему енергозбереження та моніторингу. Таку систему можна при необхідності масштабувати, покращити та змінити.

Мобільні додатки в таких системах є ефективним та зручним доповненням до основної системи, адже дозволяють в будь-який момент часу з будь-якого місця переглянути, внести або змінити дані.

2.4 Висновки до розділу 2

У розділі 2 – опис існуючих програмних рішень при розв’язанні задач енергоменеджменту було:

1. Проаналізовано існуючі задач енергоменеджменту, тобто електронні таблиці та програмне забезпечення для відображення статистичних даних, такі як: Microsoft Excel, 1C, Dashboard.
2. Виявлено архітектуру програмного забезпечення, застосований функціонал, зручність використання систем.
3. Визначено переваги та недоліки застосованої архітектури та функціоналу в існуючих системах.

3. ЗАСОБИ РОЗРОБКИ

Для створення програмного забезпечення використовувалось середовище розробки Android Studio, мова Dart, фреймворки Flutter та Swagger.

3.1 Розробка мобільних додатків

Розробка мобільних додатків - це сукупність процесів та процедур, пов’язаних із написанням програмного забезпечення для невеликих бездротових обчислювальних пристроїв, таких як смартфони та інші портативні пристрої.

Як і розробка веб-додатків, розробка мобільних додатків сягає корінням у більш традиційну розробку програмного забезпечення. Однак одна з критичних відмінностей полягає в тому, що мобільні програми часто пишуться спеціально для того, щоб скористатися унікальними можливостями конкретного мобільного пристрою. Наприклад, ігровий додаток може бути написаний, щоб скористатися акселерометром iPhone, або мобільний додаток для здоров'я, щоб скористатися датчиком температури смарт-годинника.

Сьогодні двома найвидатнішими мобільними платформами є iOS від Apple та Android від Google. Телефони та планшети від Apple постачаються з попередньо завантаженими основними програмами, включаючи повноцінний веб-браузер та Apple App Store.

Пристрої Android також мають попередньо завантажені подібні програми, і ви можете встановити більше за допомогою Google Play Store.

У перші роки використання мобільних додатків єдиним способом забезпечити оптимальну роботу додатка на будь-якому пристрої була його власна розробка. Це означало, що новий код потрібно було писати спеціально для конкретного процесора кожного пристрою. Сьогодні більшість розроблених мобільних додатків є пристроєм-агностиком.

Раніше, якщо додаток мав бути міжплатформеним і працювати на декількох операційних системах (ОС), було мало, якщо взагалі було, коду, який можна було б використати повторно з початкового проекту розробки. Процес розробки мобільного додатку зображено на рисунку 3.1.

Рисунок 3.1 — Процес розробки мобільного додатку

По суті, кожен пристрій вимагав власного проекту розробки мобільних додатків зі своєю базою коду. Сучасні крос-платформні інструменти використовують спільні мови, такі як C # та JavaScript, для обміну кодом між проектами; що більш важливо, вони добре інтегруються з інструментами управління життєвим циклом додатків, такими як Дженкінс. Це дозволяє розробникам використовувати єдину кодову базу для Apple iOS, Google Android та прогресивних веб-програм (PWA).

Перш ніж розробляти програму, потрібно визначити, який тип ви будете створювати. Ось розбивка декількох типів технологій розробки мобільних додатків з інформацією про кожну.

Рідні додатки — це програми, створені з використанням інтегрованих середовищ розробки (IDE) та мов для мобільних ОС, таких як Apple iOS або Google Android. Власні програми дозволяють налаштувати необхідні функції, але вони можуть коштувати дорожче, ніж інші технології.

Гібридні програми — це веб-програми, які діють як власні програми. Вони розроблені з використанням таких технологій, як HTML, JavaScript та каскадні таблиці стилів (CSS). Гібридні програми є більш економічно ефективними у розробці, ніж власні програми, і їх можна створювати швидше, але вони не такі багаті на функції, як рідні програми.

Інкапсульовані програми. Інкапсульована програма працює в межах програми-контейнера. Такі продукти, як інструмент створення програм перетягування та падіння Microsoft Power App, дозволяють менш досвідченим розробникам швидко створювати мобільні програми. Але відсутність ізоляції від базової ОС, блокування ОС та відносна новизна можуть створити проблеми.

Вартість розробки програми може коливатися від майже нічого до мільйонів доларів - все залежить від типу програми та її використання.

3.1 Операційна система Android

Android — це операційна система з відкритим кодом та операційною системою на базі Linux для мобільних пристроїв, таких як смартфони та планшетні комп'ютери. Android був розроблений Open Handset Alliance на чолі з Google та іншими компаніями.

Android пропонує уніфікований підхід до розробки додатків для мобільних пристроїв, що означає, що розробникам потрібно розробляти лише для Android, і

їх додатки повинні мати можливість працювати на різних пристроях на базі Android.

Вихідний код для Android доступний під вільними та відкритими ліцензіями програмного забезпечення. Google публікує більшу частину коду під ліцензією Apache версії 2.0, а решту, змінене ядро Linux, під загальною публічною ліцензією GNU версії 2.

Програми для Android, як правило, розробляються на мові Java за допомогою комплекту розробки програмного забезпечення Android. Архітектура стандартного мобільного додатку для Android зображена на рисунку 3.2.

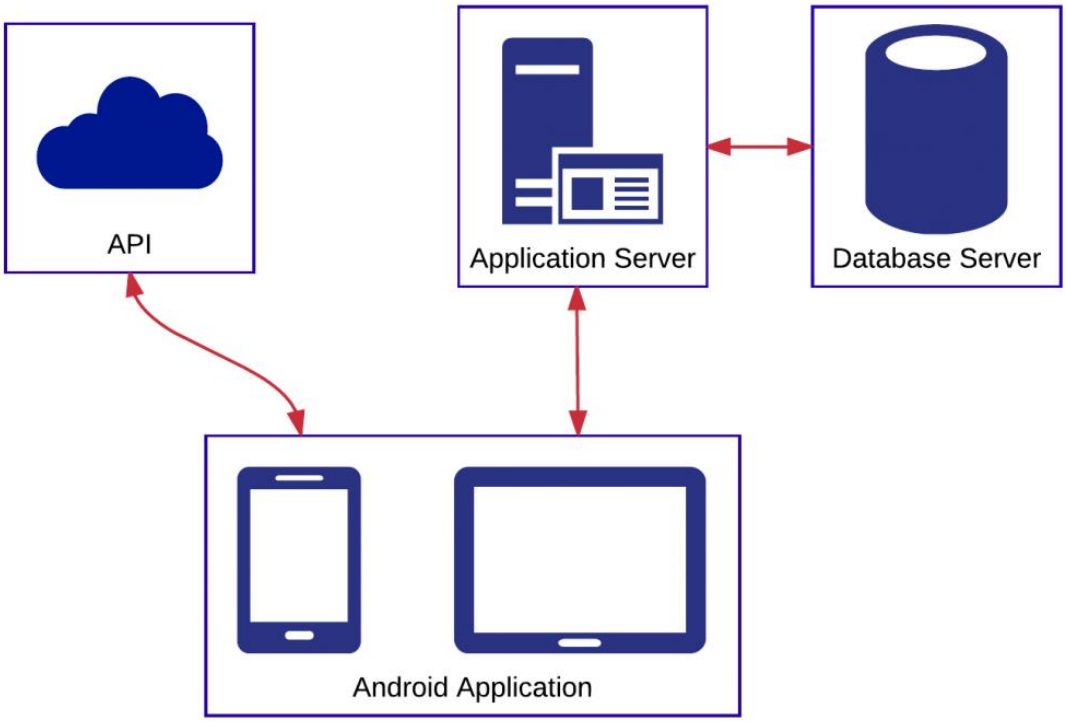


Рисунок 3.2 — Архітектура мобільного додатку для Android

Після розробки програми для Android можуть бути легко упаковані та розпродані через такий магазин, як Google Play, SlideME, Opera Mobile Store, Mobango, F-droid та Amazon Appstore.

Android забезпечує сотні мільйонів мобільних пристроїв у понад 190 країнах світу. Це найбільша встановлена база будь-якої мобільної платформи, яка

швидко зростає. Щодня у всьому світі активується понад 1 мільйон нових пристроїв Android.

3.2 Операційна система iOS

iOS, яку раніше називали iPhone OS, — це мобільна операційна система, розроблена компанією Apple Inc. Перший її випуск відбувся в 2007 році, до складу якої входили iPhone та iPod Touch. iPad (1-го покоління) вийшов у квітні 2010 року, а iPad Mini — у листопаді 2012 року.

Пристрої iOS розвиваються досить часто, і з досвіду ми виявляємо, що принаймні одна версія iPhone та iPad запускається щороку.

Потужність iOS можна відчути за допомогою деяких із наведених нижче функцій, що входять до складу пристрою.

- Карти
- Сірі
- Facebook та Twitter
- Multi-Touch
- Акселерометр
- GPS
- Високоякісний процесор
- Камера
- Сафарі
- Потужні API
- Ігровий центр
- Покупка в додатку
- Нагадування
- Широкий вибір жестів

Кількість користувачів iPhone / iPad значно зросла. Це створює можливість для розробників заробляти гроші, створюючи додатки для iPhone і iPad, Apple App Store.

Архітектура мобільного додатку iOS (рисунок 3.3) відрізняється від мобільного додатку Android. Додається сервіс ідентифікації, за допомогою якого дані будуть надійно збережені.

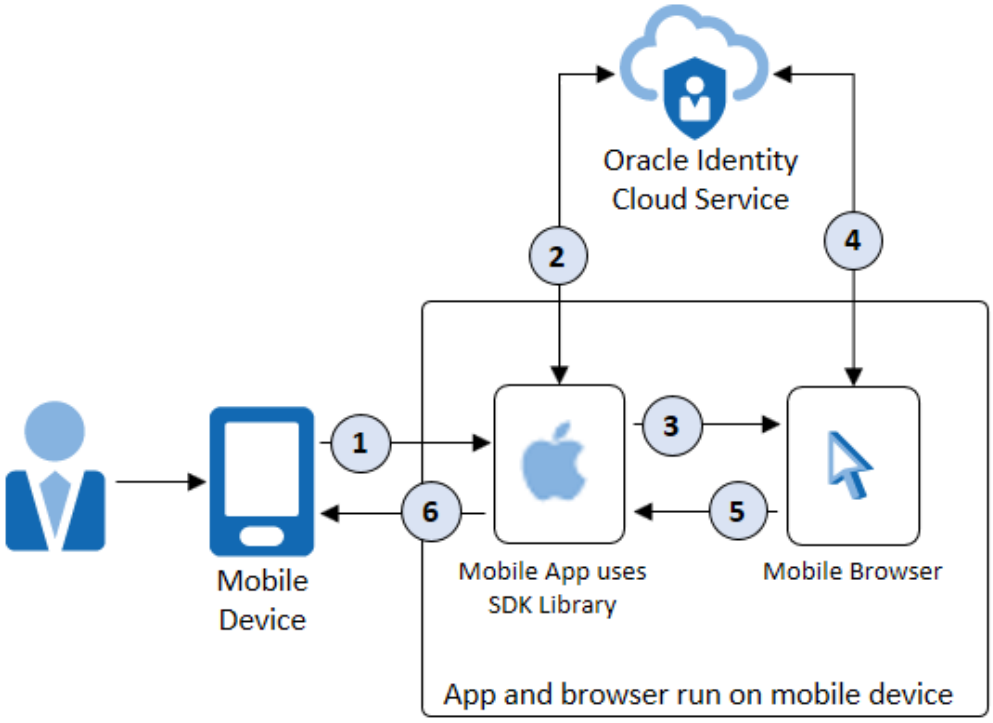


Рисунок 3.3 — Архітектура мобільного додатку iOS

Для нової для iOS Apple розробила магазин додатків, де користувач може купувати програми, розроблені для своїх пристроїв iOS. Розробник може створювати як безкоштовні, так і платні програми в App Store.

Щоб розробляти додатки та розповсюджувати їх у магазині, розробник вимагатиме реєстрації в програмі для розробників iOS, яка коштує $ 99 на рік, і Mac з Mountain Lion або вище для її розробки з найновішим Xcode.

3.1 Середовище розробки Android Studio

Програма Android Studio [3] — це офіційне інтегроване середовище розробки (IDE) для розробки додатків на Android, засноване на IntelliJ IDEA.

Окрім потужного редактора коду та інструментів розробника IntelliJ, Android Studio пропонує ще більше функцій, таких як:

1. Гнучка система побудови на основі Gradle;
2. Швидкий і багатофункціональний емулятор;
3. Єдине середовище, де ви можете розробляти на всі пристрої Android;
4. Застосовувати інструмент ‘‘hot-reload’’ для надсилання змін коду та ресурсів до запущеної програми без перезапуску програми;
5. Шаблони коду (рисунок 3.4) та інтеграція GitHub, щоб допомогти створити загальні функції програми та імпортувати код;

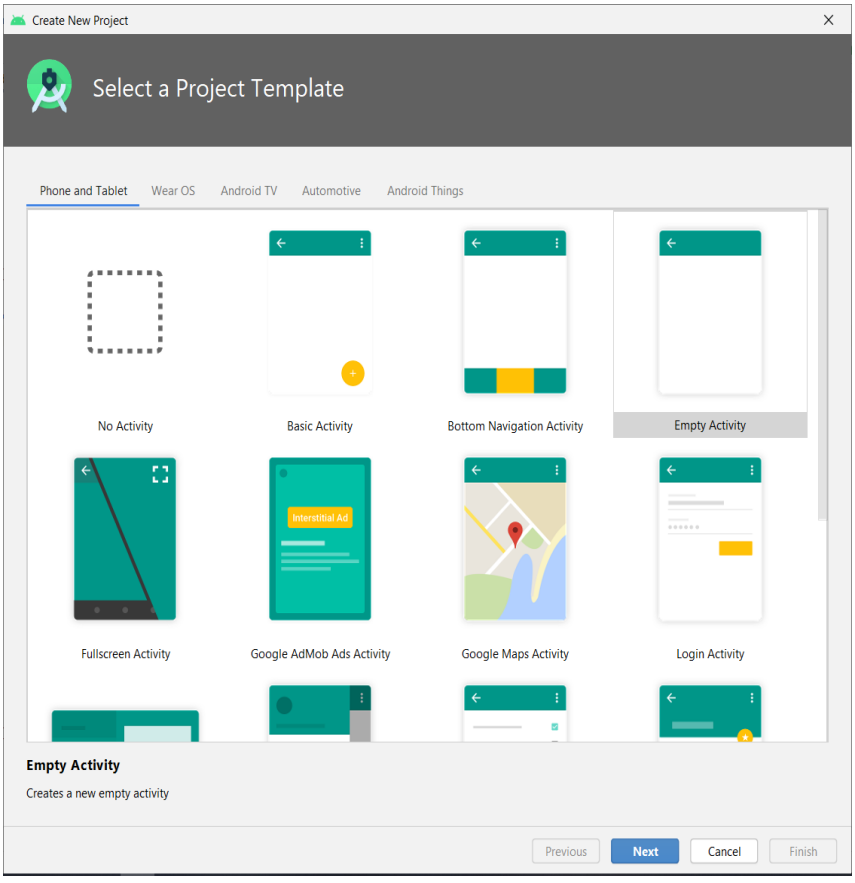


Рисунок 3.4 — Шаблони коду в Android Studio

6. Широкі інструменти та основи тестування;

7. Інструменти Lint для виявлення продуктивності, зручності використання, сумісності версій та інших проблем;

8. Підтримка C ++ та NDK;

9. Вбудована підтримка Google Cloud Platform, що спрощує інтеграцію Google Cloud Messaging та App Engine;

Емулятор Android Studio імітує пристрої Android на комп’ютері, так що можливо протестувати свою програму на різних пристроях та рівнях API Android, не маючи необхідності мати використовувати фізичний пристрій.

Емулятор забезпечує майже всі можливості справжнього пристрою Android. Можливо імітувати вхідні телефонні дзвінки та текстові повідомлення, вказувати місце розташування пристрою, моделювати різні швидкості мережі, імітувати обертання та інші апаратні датчики, отримувати доступ до Google Play Store та багато іншого.

Тестування програми на емуляторі є дещо швидшим та простішим, ніж на фізичному пристрої. Наприклад, швидше передаються дані на емулятор, ніж на пристрій, підключений через USB.

3.2. Мова Dart

Мова програмування Dart — це мова з відкритим вихідним кодом, розроблена в Google, з метою виклику розробників використовувати об'єктно-орієнтовану мову із статичним аналізом типів [4]. З часів першого випуску в 2011 році Dart сильно змінилась, як і сама мова, так і її основні цілі. У версіях 2.0 розробка додатків за допомогою Flutter стала основною метою мови.

Технологія компілятора Dart дозволяє запускати код по-різному (рисунок 3.5):

- Власна платформа: для додатків, орієнтованих на мобільні та настільні пристрої, Dart включає в себе як Dart VM із компіляцією "вчасно" (JIT), так і компілятор "AOT" для створення машинного коду.

- Веб-платформа: для програм, націлених на Інтернет, Dart включає як компілятор часу розробки (dartdevc), так і компілятор робочого часу (dart2js). Обидва компілятори перекладають Dart в JavaScript.

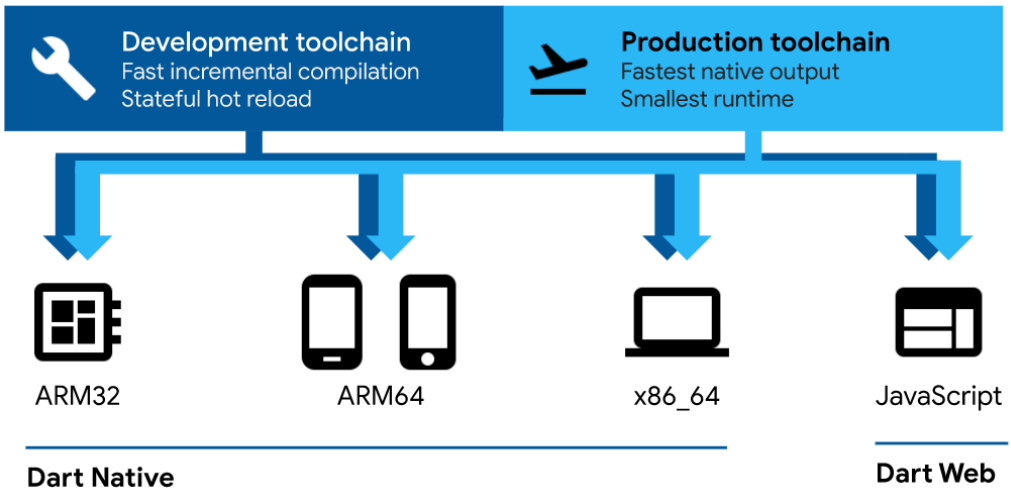


Рисунок 3.5 — Компіляція Dart

Dart дуже схожа на такі мови, як JavaScript, Java та C ++, тому написання коду на цій мові є зручним для тих, хто знайомий з вищенаведеними мовами.

3.3 Фреймворк Flutter

Фреймворк Flutter — це міжплатформенний набір інтерфейсів, призначений для повторного використання коду в таких операційних системах, як iOS та Android, а також дозволяє програмам взаємодіяти безпосередньо з базовими службами платформи [5]. Мета полягає в тому, щоб дозволити розробникам створювати високопродуктивні додатки, які працюють на різних платформах, враховуючи відмінності там, де вони існують, спільно використовуючи якомога більше коду.

Під час розробки програми на Flutter використовують віртуальну машину, яка дозволяє перезавантаження змін без повної перекомпіляції.

Для випуску програми Flutter компілюються безпосередньо до машинного коду, будь то інструкції Intel x64 чи ARM, або до JavaScript, якщо націлено на Інтернет. Фреймворк є відкритим кодом, має дозвільну ліцензію BSD і має процвітаючу екосистему сторонніх пакетів, які доповнюють основні функціональні можливості бібліотеки.

Flutter використовує віджети як одиниці композиції. Віджети є будівельними блоками користувацького інтерфейсу програми Flutter, і кожен віджет є незмінною декларацією частини користувальницького інтерфейсу.

Віджети утворюють ієрархію на основі композиції. Кожен віджет знаходиться всередині свого батьківського віджету і може отримувати дані від нього. Ця структура переноситься аж до кореневого віджета (контейнера, в якому розміщена програма Flutter, як правило, MaterialApp (рисунок 3.2) або CupertinoApp).

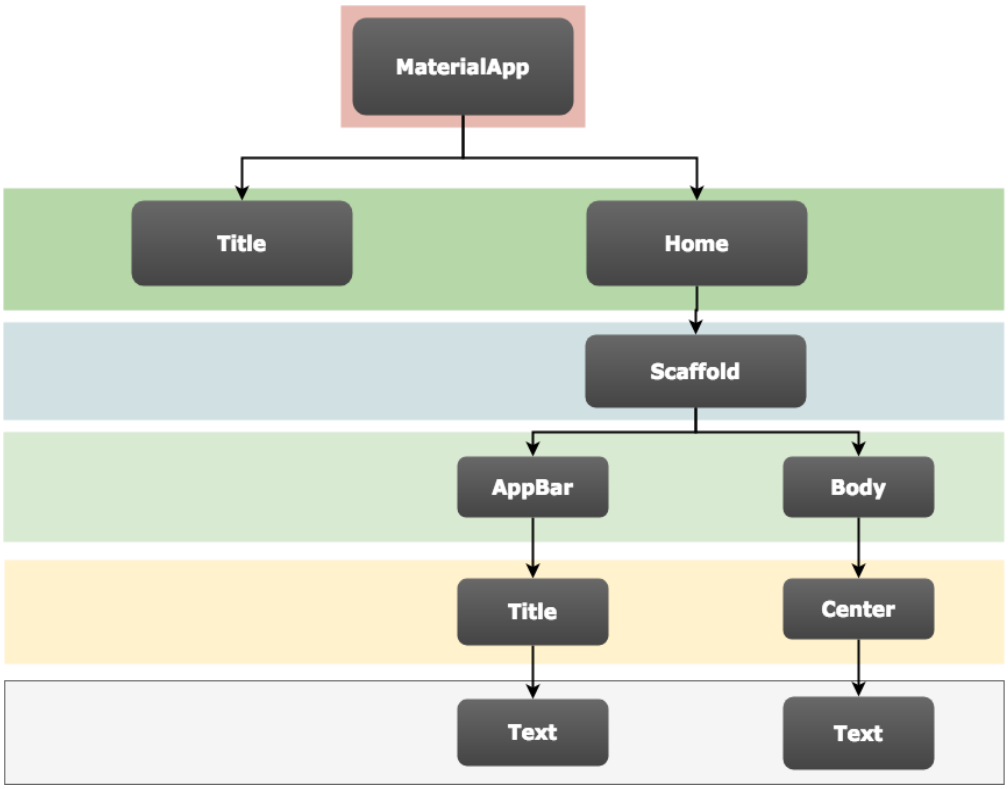


Рисунок 3.2 — Ієрархія віджета Flutter MaterialApp

Використання такого методу розробки є зручним та швидким. Не складно зрозуміти логіку написаного коду. Розробка програмного забезпечення за допомогою Flutter є вигідною, адже таку систему можна встановити на телефон з операційними системами iOS або Android, а доступні інструменти дозволяють розробити додаток зі зручним інтуїтивно зрозумілим інтерфейсом.

3.4 Фреймворк Swagger

Фреймворк Swagger — це набір правил (іншими словами, специфікація) для формату, що описує REST API [9]. Формат є як машиночитаним, так і зручним для читання користувачем.

Як результат, його можна використовувати для обміну документацією між менеджерами продуктів, тестувальниками та розробниками, але також може використовуватися різними інструментами для автоматизації процесів, пов'язаних з API.

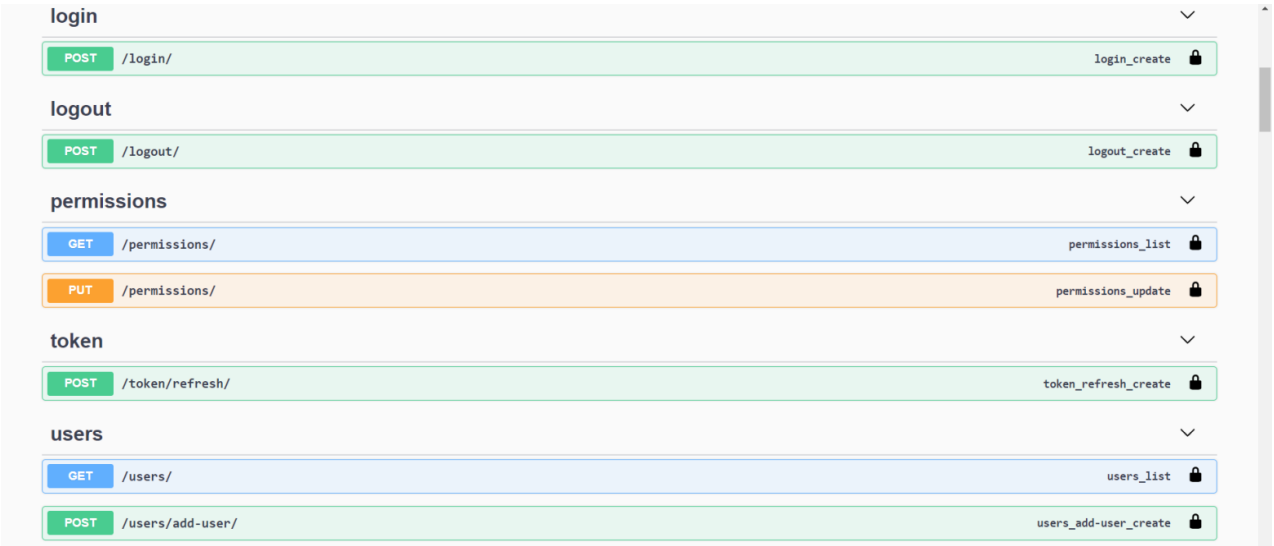


Рисунок 3.3 — Документація в Swagger

Swagger дозволяє описати структуру ваших API, щоб машини могли їх читати. Прочитавши структуру API, можна автоматично створювати інтерактивну документацію API (рисунок 3.3).

Ми також можемо автоматично створювати клієнтські бібліотеки для API багатьма мовами та досліджувати інші можливості, такі як автоматичне тестування.

3.5 Висновки до розділу 3

В розділі 3 – засоби розробки, було:

1. Обгрунтовано загальну структуру розробки мобільних додатків, особливості розробки, процес створення, тестування та налагодження роботи мобільного застосунку;
2. Проаналізовано операційну систему Android, особливості використання цієї ОС, розробки мобільних за стосунків для неї. Представлено загальну архітектуру мобільних застосунків для ОС Android;
3. Проаналізовано операційну систему iOS, особливості використання цієї ОС, розробки мобільних за стосунків для неї. Представлено загальну архітектуру мобільних застосунків для ОС iOS, порівняно з ОС Android;
4. Обгрунтовано рішення застосування середовища розробки Android Studio, особливості роботи з ним, його переваги та основний функціонал;
5. Пояснено рішення використання мови програмування Dart, її особливості та переваги;
6. Обгрунтовано використання фреймворку Flutter для розробки кросплатформних мобільних застосунків;

Описано використовуваний фреймворк Swagger для обміну документацією по розробленому серверу.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Програму було реалізовано, використовуючи архітектуру API RESTful та формат передачі даних запитів JSON. Також створено архітектуру програмного забезпечення та представлено у вигляді діаграми класів.

4.1 Архітектура API RESTful

Архітектура API (або Інтерфейс програмування програм) забезпечує метод взаємодії між двома системами.

API RESTful (або інтерфейс прикладних програм) використовує HTTP-запити GET, PUT, POST, та DELETE відповідно до стандартів REST. Це дозволяє двом програмним модулям взаємодіяти між собою.

По суті, REST API - це набір віддалених викликів із використанням стандартних методів повернення даних у певному форматі.

Системи, які взаємодіють таким чином, можуть бути дуже різними. Кожен додаток може використовувати унікальну мову програмування, операційну систему, базу даних тощо. Отже, за допомогою Rest API можна створити систему, яка може легко спілкуватися та розуміти інші програми.

Використовуючи RESTful API, ми повинні заздалегідь визначитись, які ресурси ми хочемо надавати зовнішньому світу. Зазвичай реалізується запит RESTful API, маючи на увазі наступні ідеї:

- Формат: Формат обміну даними не повинен мати обмежень;
- Реалізація: REST повністю заснований на HTTP;
- Визначення послуги: Оскільки REST дуже гнучкий, API можна змінити, щоб переконатися, що програма розуміє формат запиту / відповіді;
- API RESTful фокусується на ресурсах і на тому, як ефективно ви виконуєте з ним операції за допомогою HTTP.

Особливості стилю дизайну REST API:

- Кожна сутність повинна мати унікальний ідентифікатор;
- Для зчитування та модифікації даних слід використовувати стандартні методи;
- Він повинен забезпечувати підтримку різних видів ресурсів.

Щоб REST відповідав цій моделі, ми повинні дотримуватися наступних правил:

- Клієнт-серверна архітектура (рисунок 4.1): Інтерфейс відокремлений від серверного сховища даних. Це забезпечує гнучкість та розробку компонентів незалежно один від одного;

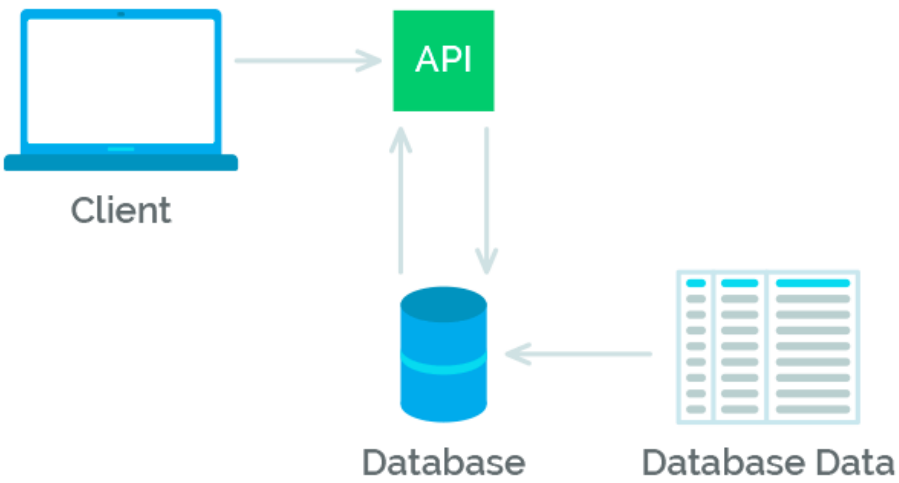


Рисунок 4.1 — Клієнт-серверна архітектура API RESTful

- Від'єднання: Клієнтські з'єднання не зберігаються на сервері між запитами;
- Кешування: Потрібно чітко вказати, чи може клієнт зберігати відповіді;
- Багаторівневість: API повинен працювати незалежно від того, взаємодіє він безпосередньо із сервером або через додатковий рівень.

Архітектура мобільного додатку з використанням REST API зображена на рисунку 4.2.

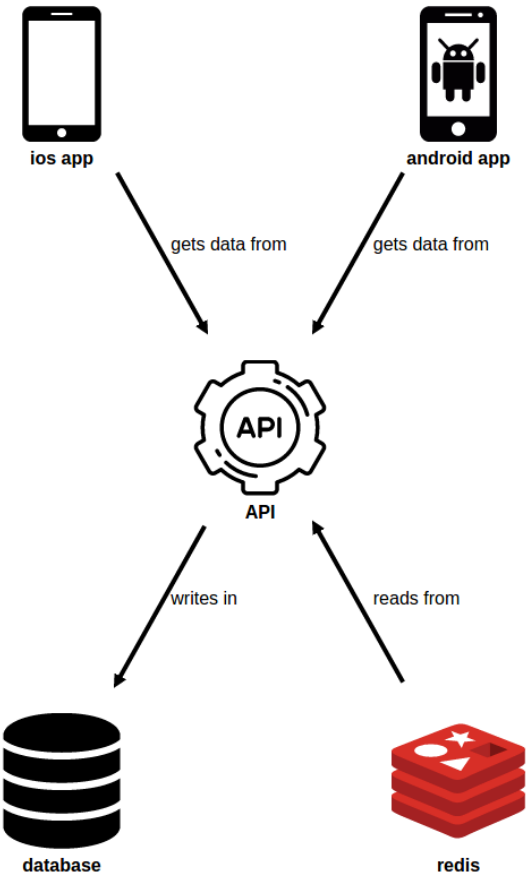


Рисунок 4.2 — Архітектура мобільного додатку

Така архітектура є дуже вигідною для будь-якого мобільного додатку, його створення, розширення та адміністрування.

4.2 Формат JSON

Формат JSON — скорочення від JavaScript Object Notation - це формат для обміну даними [6]. Як випливає з назви, JSON походить від мови програмування JavaScript, але він доступний для використання багатьма мовами, включаючи Python, Ruby, PHP та Java.

JSON використовує розширення .json. Коли його визначено в іншому форматі файлу (як .html), він може відображатися всередині лапок як рядок JSON або може бути об’єктом, присвоєним змінній. Цей формат легко передавати між

веб-сервером та клієнтом або браузером. Читабельний і легкий, JSON пропонує гарну альтернативу XML і вимагає набагато менше форматування.

JSON — це природний формат для використання в JavaScript і має безліч реалізацій, доступних для використання в багатьох популярних мовах програмування.

4.3. Реалізація комунікації з сервером

Запит POST використовується для:

- створення нового документу;
- створення нового договору;
- створення нового лічильника;
- внесення даних лічильників;
- авторизації користувача при вході в систему;
- вихід користувача з системи;
- отримання нового токена за допомогою refresh;

Функція POST має наступний вигляд:

```
Future<Map> apiPost(String url, Map data) async {
   var body = json.encode(data);
   var response = await http.post("$url",
      headers: {"Content-Type": "application/json"}, body: body);
   if (response.statusCode == 401) {
    await tokenRefresh();
    response = await http.get(url, headers: {
     "Authorization": "Bearer $token"});
   }
   if (response.statusCode == 200) {
    token = jsonDecode(response.body)['access'];
    refresh = jsonDecode(response.body)['refresh'];
```

```
      storage.write(key: "token", value: token);
      storage.write(key: "refresh", value: refresh);
      return jsonDecode(response.body);
    }
    return null;
  }
```

У випадку отримання коду 401 викликається функція tokenRefresh() для оновлення токену.

Запит GET використовується для отримання:

- списку користувачів;
- авторизованого користувача та його даних;
- списку лічильників;
- даних по одному лічильнику;
- показників лічильників;
- списку документів;
- списку договорів;
- даних одного документа.

Функція GET має наступний вигляд:

```
Future<Map> apiGet(String url) async {
    token = await storage.read(key: "token");
    var response = await http.get(url, headers: {
    "Authorization": "Bearer $token"});
    if (response.statusCode == 401) {
      await tokenRefresh();
      response = await http.get(url, headers: {
        "Authorization": "Bearer $token"});
    }
    return jsonDecode(utf8.decode(response.bodyBytes));
  }
```

Запит PATCH використовується для редагування:

- даних користувача;
- даних документа;
- даних договору;
- лічильника;
- показників лічильника;

Функція PATCH має наступний вигляд:

```
Future<Map> apiPatch(String url, Map data) async {
   var body = json.encode(data);
   var response = await http.patch("$url",
      headers: {"Content-Type": "application/json"}, body: body);
   if (response.statusCode == 401) {
    await tokenRefresh();
    response = await http.get(url, headers: {
      "Authorization": "Bearer $token"});
   }
   if (response.statusCode == 200) {
    return jsonDecode(response.body);
   }
   return null;
 }
```

Для отримання нового токену викликається функція tokenRefresh(). За її допомогою отримується новий токен для доступу користувача. Виключення, спіймане на сервері, ловиться та обробляється. Новий токен записується у внутрішнє сховище.

```
Future tokenRefresh() async {
   refresh = await storage.read(key: "refresh");
   await http.post("http://93.188.34.235:8001/token/refresh/",
      headers: {"Content-Type": "application/json"}, body:
jsonEncode({"refresh": refresh})).then((value) => {token =
jsonDecode(value.body)['access'],
```

```
        storage.write(key: "token", value: token)}).catchError((e) => {throw
Exception(e)});
  }
```

4.4 Токен доступу та його оновлення

Сучасні захищені програми часто використовують маркери доступу, щоб забезпечити користувачеві доступ до відповідних ресурсів, і ці маркери доступу зазвичай мають обмежений термін служби.

Це робиться з різних міркувань безпеки: з одного боку, обмеження терміну дії маркера доступу обмежує кількість часу, який зловмисник може використовувати викрадений маркер. Крім того, інформація, що міститься в маркері доступу або на яку посилається маркер доступу, може стати застарілою.

Коли маркери доступу закінчуються або стають недійсними, але програмі все ще потрібно отримати доступ до захищеного ресурсу, програма стикається з проблемою отримання нового маркера доступу, не змушуючи користувача ще раз надавати дозвіл.

Для вирішення цієї проблеми OAuth 2.0 представив артефакт, який називається маркером оновлення. Маркер оновлення дозволяє програмі отримати новий маркер доступу без запиту користувача (рисунок 4.3).

Програма може вимагати оновлення маркера як частину процесу отримання маркера доступу. Багато серверів авторизації реалізують механізм запиту маркера оновлення, визначений у специфікації OpenID Connect. У цьому випадку програма повинна включати область offline_access під час ініціювання запиту на код авторизації.

Після успішної автентифікації та надання згоди на доступ програми до захищеного ресурсу програма отримає код авторизації, який можна обміняти в кінцевій точці маркера як на маркер доступу, так і на оновлення.

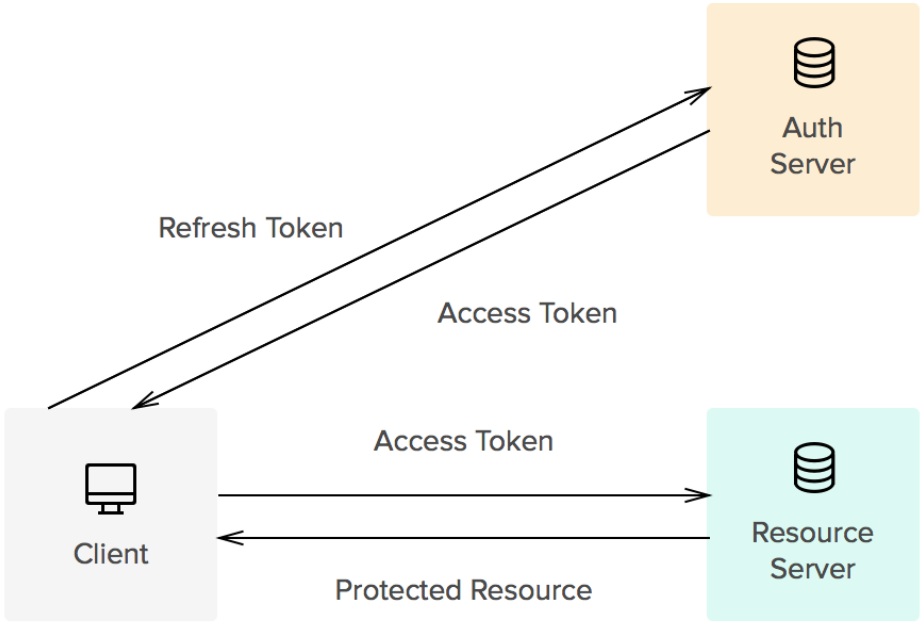


Рисунок 4.3 — Робота маркерів доступу

Коли потрібен новий маркер доступу, програма може зробити запит POST назад до кінцевої точки маркера, використовуючи тип дозволу refresh_token (веб-програми повинні містити секрет клієнта). Щоб використовувати маркер оновлення для отримання нового маркера ідентифікатора, сервер авторизації повинен підтримувати OpenID Connect, а область вихідного запиту повинна містити openid.

Хоча маркери оновлення часто довгоживуть, сервер авторизації може зробити їх недійсними. Деякі причини, через які маркер оновлення може бути недійсним, включають:

- сервер авторизації скасував маркер оновлення;
- користувач відкликав свою згоду на авторизацію;
- термін дії маркера оновлення закінчився;
- політика автентифікації для ресурсу змінилася (наприклад, спочатку в ресурсі використовувались лише імена користувачів та паролі, але зараз він вимагає MFA).

Оскільки маркери оновлення мають тривалий термін експлуатації, розробники повинні переконатися, що існують суворі вимоги до зберігання, щоб

уникнути їх витоку. Наприклад, у веб-програмах маркери оновлення повинні залишати серверну систему лише під час надсилання на сервер авторизації, а серверну систему слід захищати. Таємницю клієнта слід захищати подібним чином. Мобільні додатки не потребують клієнтського секрету, але вони все одно повинні зберігати маркери оновлення десь лише клієнтська програма може отримати доступ.

4.5 Синхронні та асинхронні операції

Підходи асинхронізації та очікування в Dart дуже схожі на інші мови (дивлячись на вас C #), що робить зручну тему для розуміння тим, хто раніше використовував цей шаблон. Асинхронні операції дозволяють програмі завершити роботу, чекаючи закінчення іншої операції, на відміну від синхронних (рисунок 4.3). Ось декілька поширених асинхронних операцій:

- отримання даних через мережу;
- запис у базу даних;
- зчитування даних з файлу.

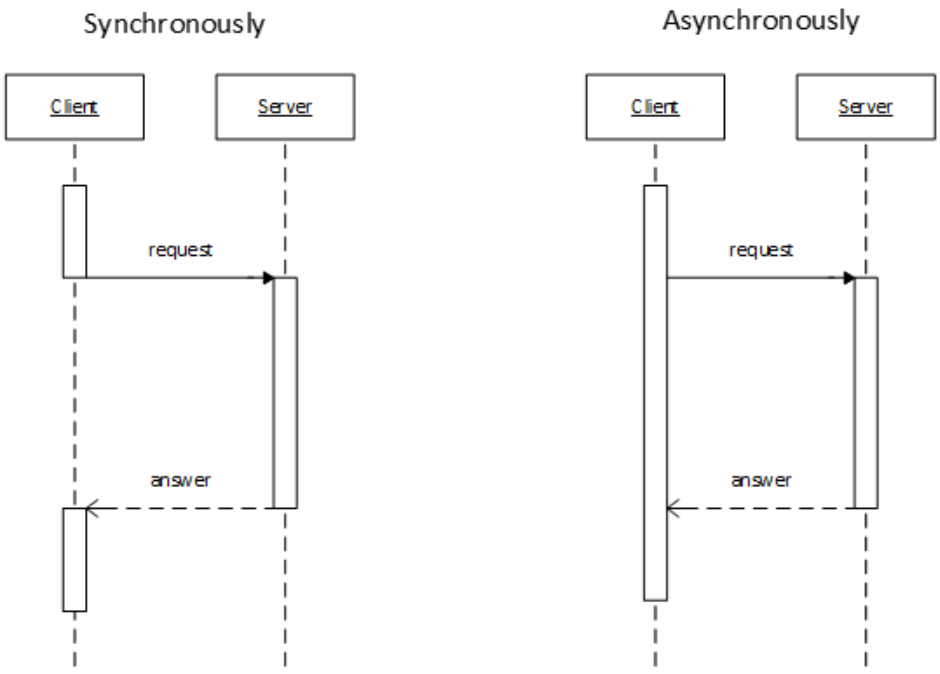


Рисунок 4.3 — Синхорнні та асинхронні операції

Для виконання асинхронних операцій у Dart можна використовувати клас Future та ключові слова async і await.

Для будь-яких функцій, які необхідно запустити асинхронно, потрібно додати до них модифікатор async. Також використовуються клас Future.

Вкоду всередині абонента призупиняється, поки виконується операція асинхронізації. Коли операція завершена, значення того, що чекали, міститься в об'єкті типу Future.

Віджет FutureBuilder використовується для створення віджетів на основі останнього знімка взаємодії з майбутнім. Потрібно, щоб Future було отримано раніше або через зміну стану, або через зміну залежностей. FutureBuilder - це віджет, який допоможе вам виконати якусь асинхронну функцію і на основі результату цієї функції ваш інтерфейс буде оновлений.

4.6. Віджети StatelessWidget та StatefulWidget

Віджет може бути або із станом, або без нього. Якщо віджет може змінюватися - наприклад, коли користувач взаємодіє з ним - він відображає стан. Такий відмет називається StatefulWidget.

StatelessWidget ніколи не змінюється. Icon, IconButton і Text - це приклади віджетів без стану (рисунок 4.4).

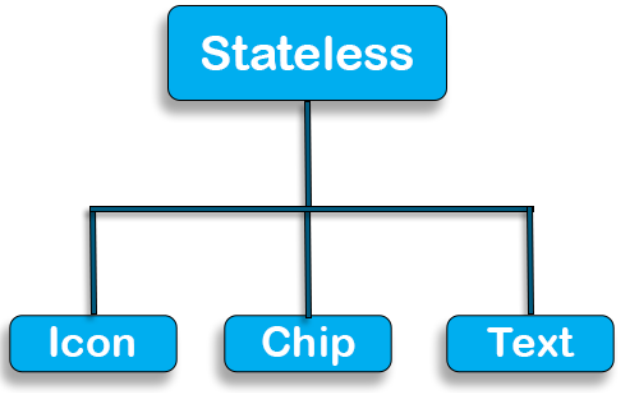


Рисунок 4.4 — Віджет Stateless

StatelessWidget є динамічним: наприклад, він може змінювати свій вигляд у відповідь на події, спричинені взаємодією користувача або коли він отримує дані. Віджети Checkbox, Radio, Slider, InkWell, Form та TextField - це приклади віджетів, що містять стан (рисунок 4.5).

Стан віджета зберігається в об'єкті стану, відокремлюючи стан віджета від його зовнішнього вигляду. Стан складається із значень, які можуть змінюватися, наприклад, поточного значення повзунка або встановленого прапорця. Коли стан віджета змінюється, об’єкт стану викликає setState(), повідомляючи фреймворку перекроювати віджет.

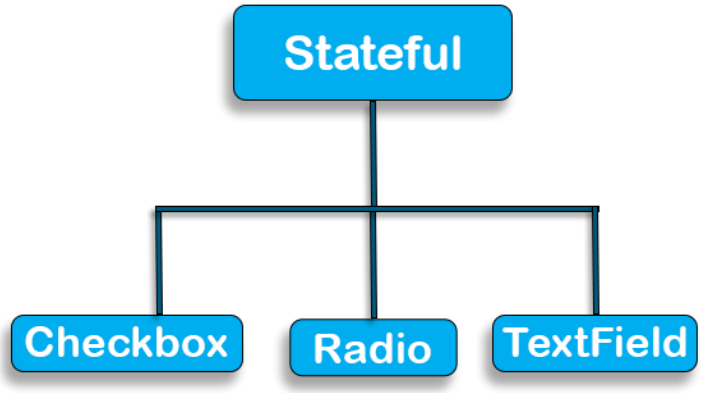


Рисунок 4.5 — Віджет Stateful

4.7 Архітектура програмного забезпечення

Для відображення архітектури програмного забезпечення було представлено діаграму класів, діаграму прецедентів, файлову систему та залежності, використані для створення програми.

4.7.1 Діаграма класів

Уніфікована мова моделювання (UML) може допомогти вам моделювати системи різними способами. Одним з найбільш популярних типів в UML є діаграма класів. Популярні серед інженерів програмного забезпечення для

документування архітектури програмного забезпечення, діаграми класів є типом структурної діаграми, оскільки вони описують, що повинно бути в системі, що моделюється. Незалежно від рівня обізнаності з UML або діаграмами класів, наше програмне забезпечення UML розроблено таким, щоб бути простим і простим у використанні.

UML був створений як стандартизована модель для опису об'єктно-орієнтованого підходу до програмування. Оскільки класи є будівельним блоком об'єктів, діаграми класів є будівельними блоками UML. Різні компоненти діаграми класів можуть представляти класи, які насправді будуть запрограмовані, основні об'єкти або взаємодії між класами та об'єктами.

Сама форма класу складається з прямокутника з трьома рядками. Верхній рядок містить ім'я класу, середній рядок містить атрибути класу, а нижній розділ відображає методи або операції, які клас може використовувати. Класи та підкласи згруповані разом, щоб показати статичний зв'язок між кожним об'єктом.

На рисунку 4.6 зображено архітектуру програмного модуля (діаграму класів) для входу користувача в систему, авторизацію та перехід на початкову сторінку.

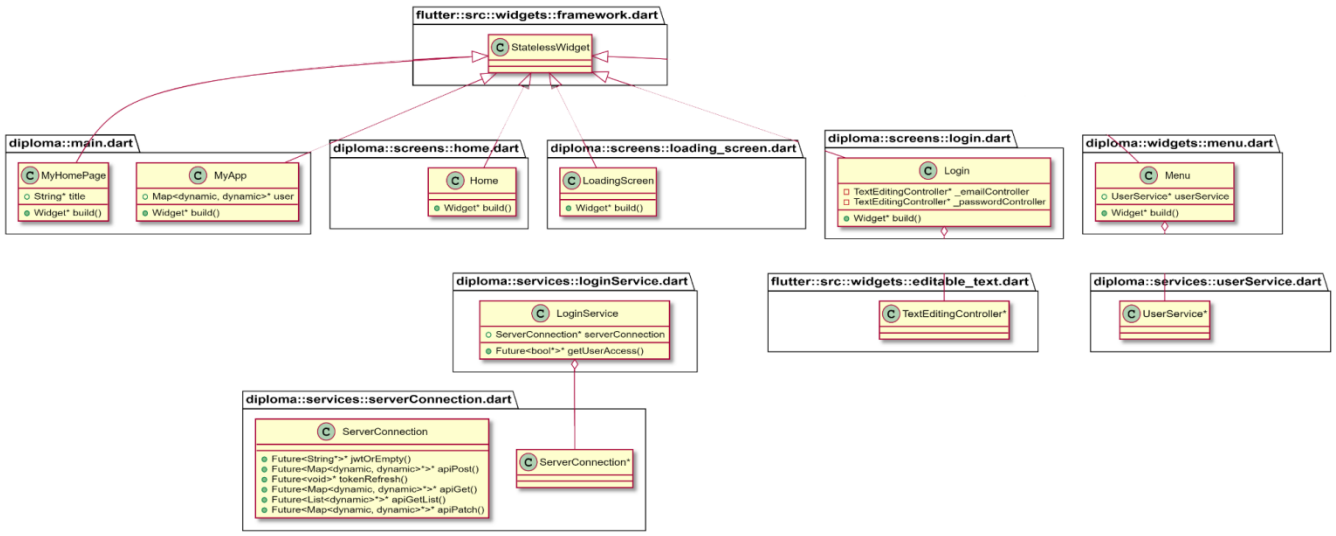


Рисунок 4.6 — Діаграма класів для авторизації користувача

Створено окремі класи для запитів на сервер. Також між сторінкою, що відображається користувачу та сервером створено окремий клас — сервіс, що передає та отримує дані класу сервера. Таким чином клас відображення ніяк не впливає на сервер та не передає, отримує дані з нього.

4.7.2 Діаграма прецедентів

Діаграма прецедентів UML - це основна форма системних / програмних вимог до нової слаборозвиненої програми. У випадках використання вказується очікувана поведінка (що), а не точний спосіб її здійснення (як). Описані випадки використання можуть бути позначені як текстовим, так і візуальним поданням (тобто діаграмою прецедентів). Діаграма прецедентів зображена на рисунку 4.7.

Ключова концепція моделювання діаграми прецедентів полягає в тому, що це допомагає розробити систему з точки зору кінцевого користувача. Це ефективний прийом для комунікації поведінки системи в термінах користувача шляхом вказівки всієї видимої зовні системи поведінки.

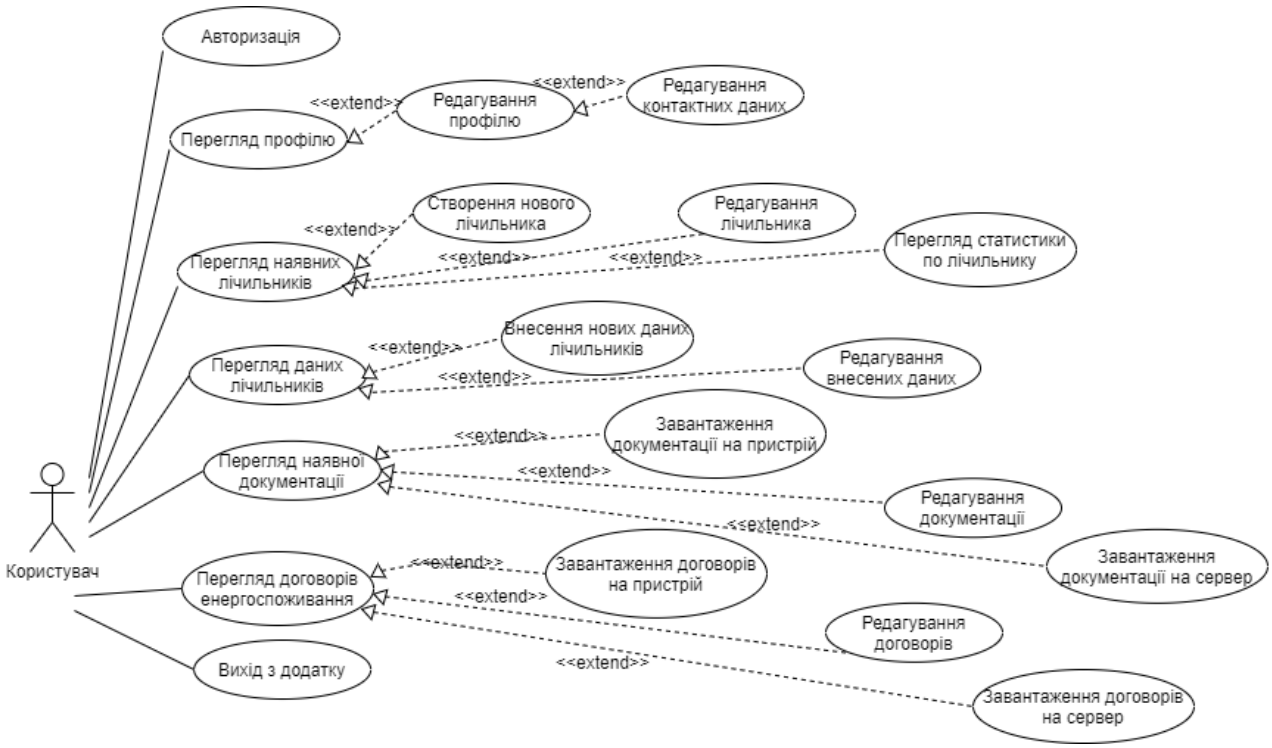


Рисунок 4.7 — Діаграма прецедентів

На діаграмі прецедентів зображено основні функції кінцевого користувача, доступні йому в мобільному додатку.

4.7.3 Файлова система

На рисунку 4.8 зображено файлову систему розробленої програми.

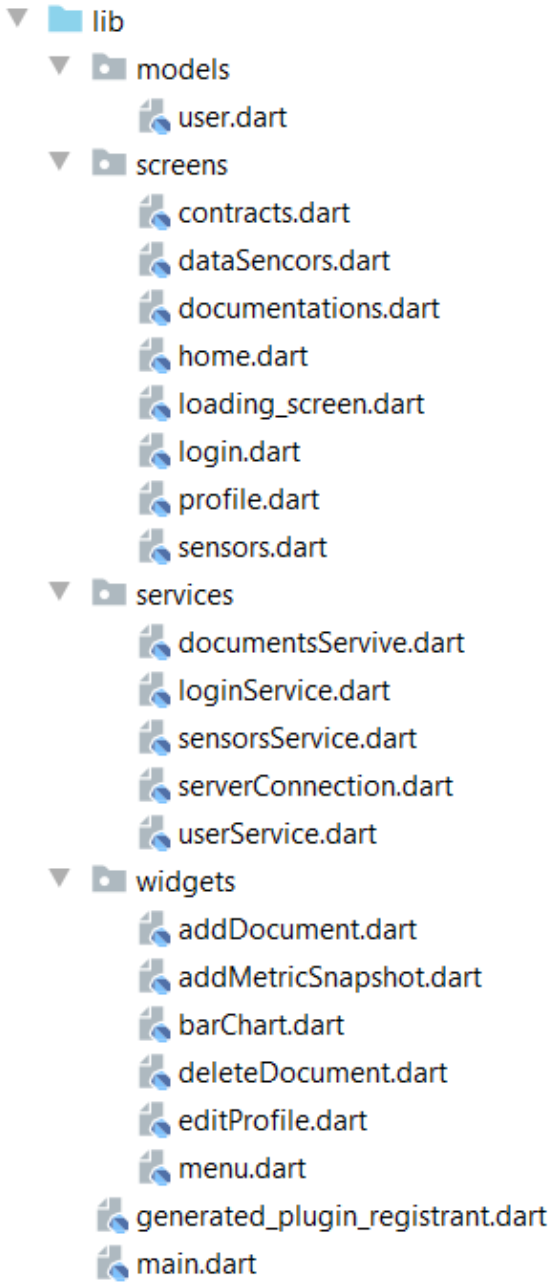


Рисунок 4.8 — Фйалова система

Основні папки – моделі, екрани, сервіси та віджети, які місять в собі відповідні файли. Головний файл, який запускає систему — main.dart. Файл, в якому йде безпосереднє підключення до серверу — serverConnection.dart.

4.7.4 Використані залежності

На рисунку 4.9 зображено усі використані залежності.

```
dependencies:
  flutter:
    sdk: flutter
  redux: ^3.0.0
  flutter_redux: ^0.5.2
  redux_logging: ^0.3.0
  redux_thunk: ^0.2.0
  redux_persist_flutter: ^0.6.0-rc.1
  redux_persist: ^0.7.0-rc.2
  flutter_bloc: ^4.0.0
  bloc: ^4.0.0
  rxdart: ^0.24.0
  equatable: ^1.1.1
  flutter_secure_storage: ^3.3.3
  dio: ^3.0.9
  introduction_screen: ^1.0.9
  flutter_svg: ^0.19.1
  eva_icons_flutter: ^2.0.0
  file_picker: ^3.0.1
  flutter_downloader: ^1.6.0
  dcdg: ^2.0.1
  charts_flutter: ^0.9.0
  charts_common: ^0.9.0
  flutter_mobile_carousel: ^1.0.3
```

Рисунок 4.9 — Використані залежності

Залежності є однією з основних концепцій менеджера пакунків pub. Залежність — це ще один пакет, який потрібен пакунку для роботи. Залежності вказані у файлі pubspec.yaml. Перелічуються лише безпосередні залежності — програмне забезпечення, яке пакет використовує безпосередньо. Pub обробляє для вас перехідні залежності.

4.8 Висновки до розділу 4

В розділі 4 – опис програмної реалізації, було:

1. Обгрунтовано використання архітектури API RESTful, що забезпечує метод взаємодії між двома системами;
2. Представлено формат для обміну даними між пристроєм та сервером JSON;
3. Показано алгоритми реалізації функцій для обміну інформацією між пристроєм та сервером. Пояснено запити POST, GET, PATCH, DELETE;
4. Обгрунтовано застосування токену доступу та його оновлення для авторизації користувача та отримання даних з серверу;
5. Показано принцип роботи віджетів зі станом (StatefulWidget) та без стану (StatelessWidget).
6. Продемонстровано архітектуру програмного забезпечення з використанням UML діаграми класів, діаграми прецедентів та файлової системи.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Інтерфейс користувача представлено у вигляді скриншотів, зображених на рисунках. Показано вхід та вихід користувача з системи, перегляд, редагування та завантаження документації. Внесення даних, редагування профілю.

5.1 Вхід користувача у систему та вихід з неї

На рисунку 5.1 зображено інтерфейс входу користувача у додаток. В першу чергу він вводить e-mail та пароль до свого облікового запису. Зареєструватись та увійти незареєстрованим особам неможливо. Якщо пароль буде неправильним або користувача з такою адресою немає, тоді система не приймає цей запит та повідомляє про помилку.

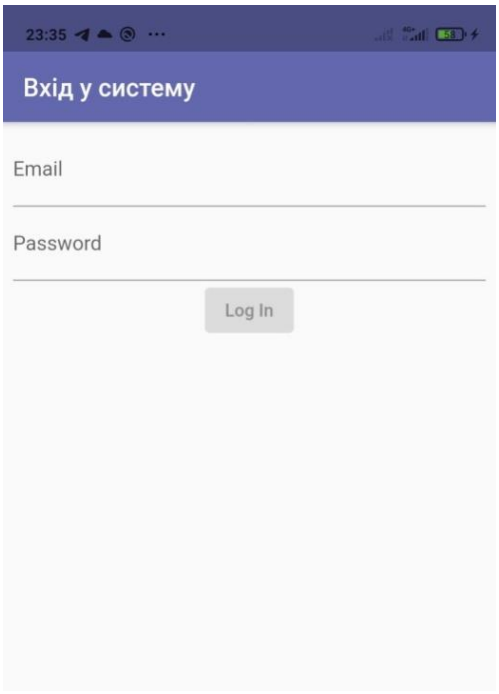


Рисунок 5.1 — Вхід у додаток

Якщо вхід було виконано успішно, користувач переходить на головну сторінку. Зліва створено меню, в якому користувач може обрати необхідну йому сторінку та перейти на неї. В мобільному додатку наявні такі сторінки: профіль користувача, сенсори, дані лічильників, документація та договори енергопостачання. Меню показано на рисунку 5.2.

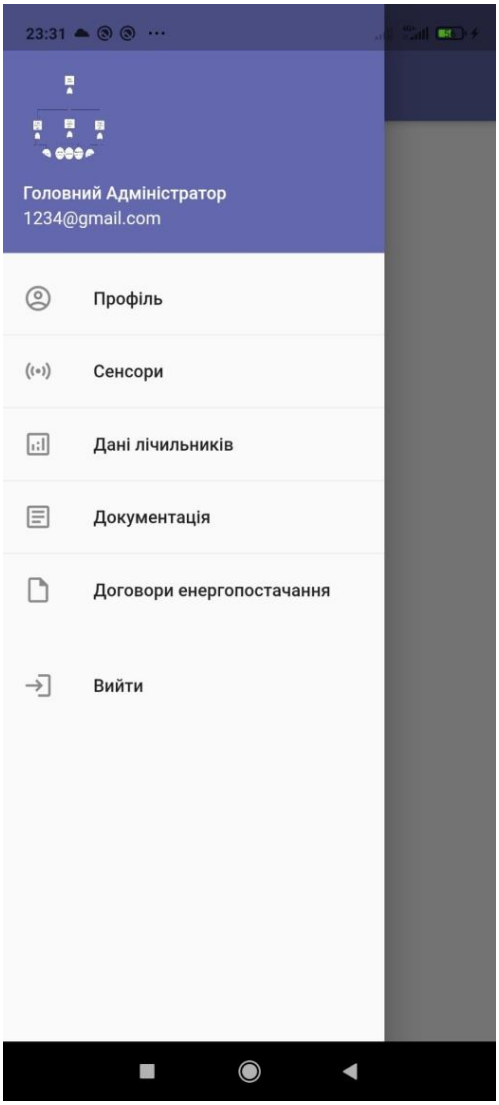


Рисунок 5.2 — Меню

При бажанні або необхідності користувач може вийти зі свого облікового запису на телефоні, скориставшись кнопкою ‘‘Вийти’’ внизу меню (рисунок 5.2). Після натиснення користувач одразу перейде на сторінку входу у систему.

5.2 Профіль користувача

На сторінці ‘‘Профіль’’ (рисунок 5.3) користувач може переглянути свій профіль: ім’я, прізвище, фотографію профілю та контактні дані.

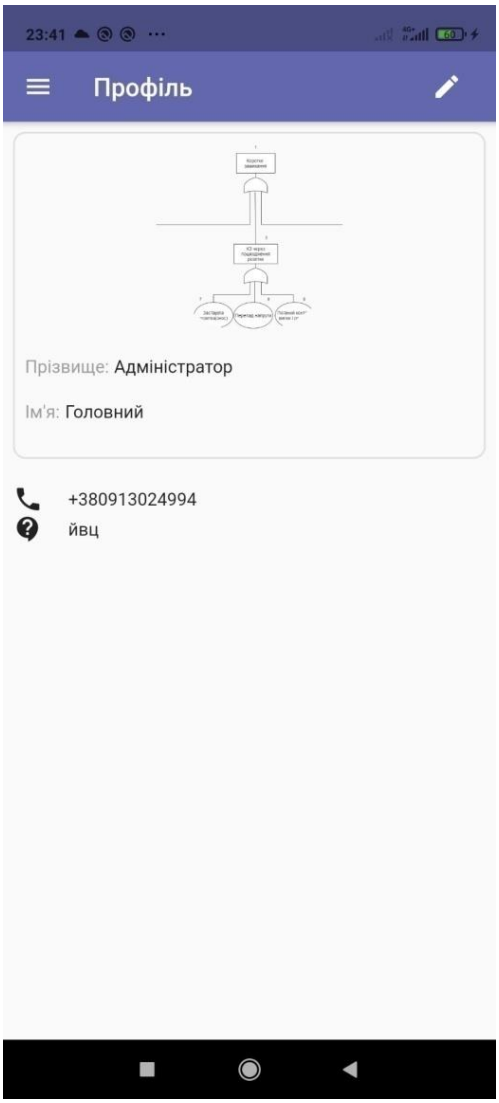


Рисунок 5.3 — Профіль користувача

При необхідності користувач може відредагувати свій профіль (рисунок 5.4). Для цього необхідно натиснути на іконку редагування в правому верхньому куті. При цьому поверх виприве вікно з можливістю редагування.

Можна змінити своє ім’я, прізвище, завантажити фотографію профілю з внутрішнього сховища телефона. Також додати контактні дані (телефон,

електронну адресу, месенджер або інше) або змінити (видалити) вже існуючі. Для цього користувач вводить значення (номер, електронну адресу, адресу месенджера або інше), примітки та вибирає тип даних (номер телефону, електронна адреса, месенджер або інше). При виборі типу контактних даних автоматично встановлюється іконка біля контакту в профілі.

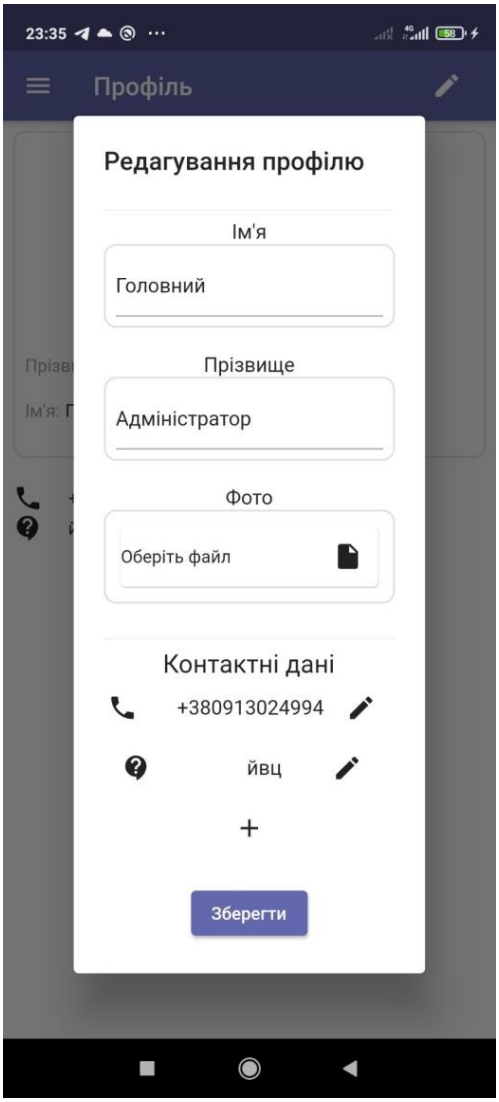


Рисунок 5.4 — Редагування профілю

Для редагування контактних даних необхідно натиснути на значок редагування біля контакту. Буде відображено новий блок (рисунок 5.5), який містить наявні контактні дані, які можна змінити — назва, тип (пошта, телефон, месенджер чи інше), значення та примітки. Після цього необхідно натиснути на

кнопку ‘‘Готово’’. Таким чином внесена інформація буде збережена.

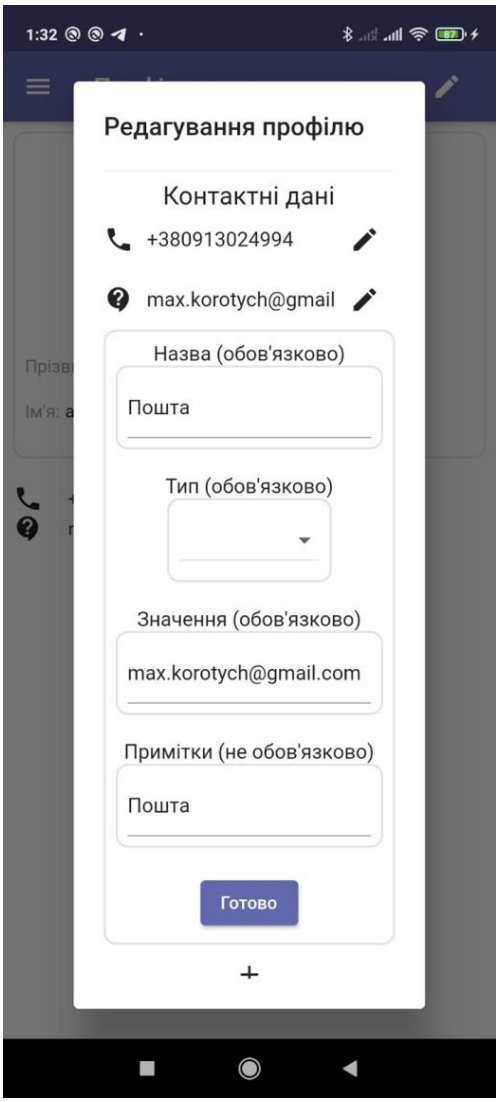


Рисунок 5.5 — Редагування контактних даних

Якщо дані було введено неправильно або не повністю, тоді система не приймає ці зміни. Для збереження змін необхідно натиснути кнопку внизу діалогового вікна ‘‘Зберегти’’.

5.3 Перегляд документів та договорів

Було створено можливість переглядати документи, що вже завантажені на сервер. Для цього необхідно перейти на сторінку ‘‘Документація’’ через меню

(рисунок 5.6). Буде надіслано запит на сервер та відображено усі наявні документи.

Можна переглянути дані про них (дату завантаження, редагування). Також відредагувати існуючий документ, натиснувши на іконку редагування справа від документа. Можна змінити назву документа та підвантажити новий документ замість наявного. Кожен документ можна зберегти на телефон у внутрішнє сховище, натиснувши на іконку завантаження справа від документа. Сторінку з документами зображено на рисунку.

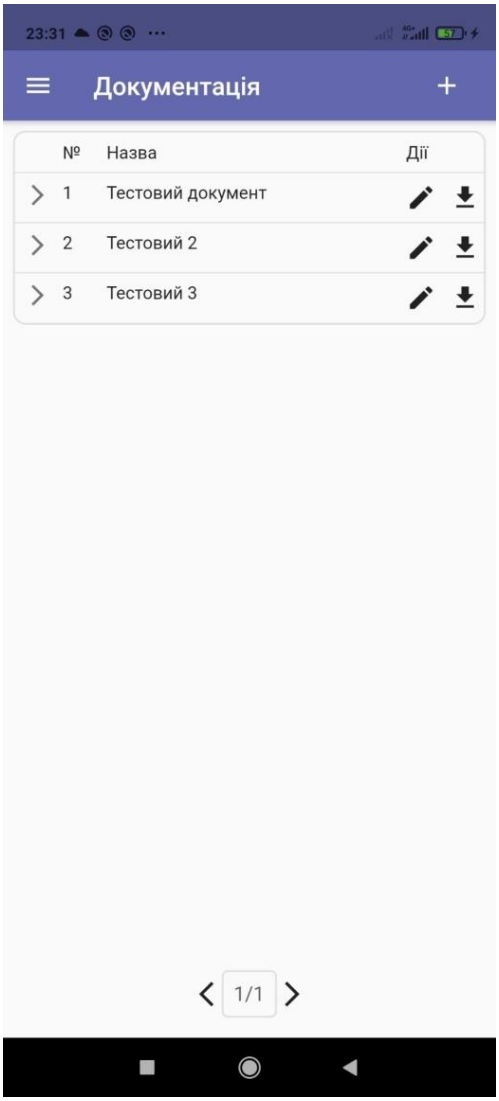


Рисунок 5.6 — Документація

Також на цій сторінці можна створити новий документ. Для цього необхідно натиснути на іконку додати в правому верхньому куті сторінки. Після натиснення буде відображено діалогове вікно, на якому є поля для назви документа та індексу (рисунок 5.7). Назва документа буде відображена у списку документів. Індекс необхідний для порядку відображення документів у списку, тобто індекс вказує пріоритетність. Внизу вікна є кнопка для завантаження документа з внутрішнього сховища.

Рисунок 5.7 — Завантаження документації

Для того, щоб додати документ, необхідно дати дозвіл на доступ системи до внутрішнього сховища. Це можливо зробити при першому натиску на кнопку

документа. При цьому висвічується діалогове вікно, в якому необхідно надати доступ до файлів внутрішнього сховища телефону.

Якщо якесь поле не було заповнене, або документ не було додано - система не приймає запит. Також є перевірка на те, чи є індекс, введений користувачем, натуральним числом.

5.4 Введення даних лічильників

Для того, що ввести дані з лічильників, необхідно перейти на сторінку ‘‘Дані лічильників’’. Після цього обрати необхідний лічильник (у кожного лічильника буде індивідуальний серійний номер). Після цього ввести отримані дані та зберегти їх.

На рисунку 5.8 зображено сторінку усіх лічильників. Кожен можна відредагувати (його назву, серійний номер).

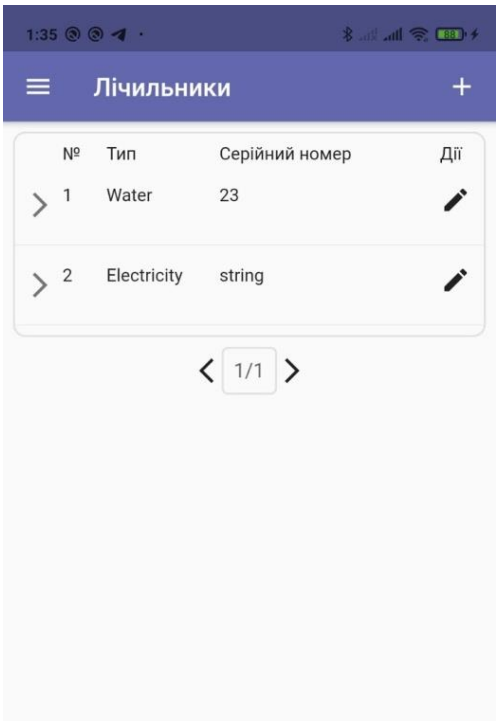


Рисунок 5.8 — Список сенсорів

У кожного корпуса будуть свої лічильники та відповідальні (енергоменеджери факультетів), які будуть мати до них доступ.

При натиску на іконку розгорнути зліва від номеру лічильника, то буде відображено статистику по цьому лічильнику (ступінчату діаграму) за весь час (рисунок 5.9).

При необхідності можна вибрати період часу, за який потрібно переглянути статистику. Для цього потрібно обрати з якої дати та до якої дати потрібно відобразити інформацію.

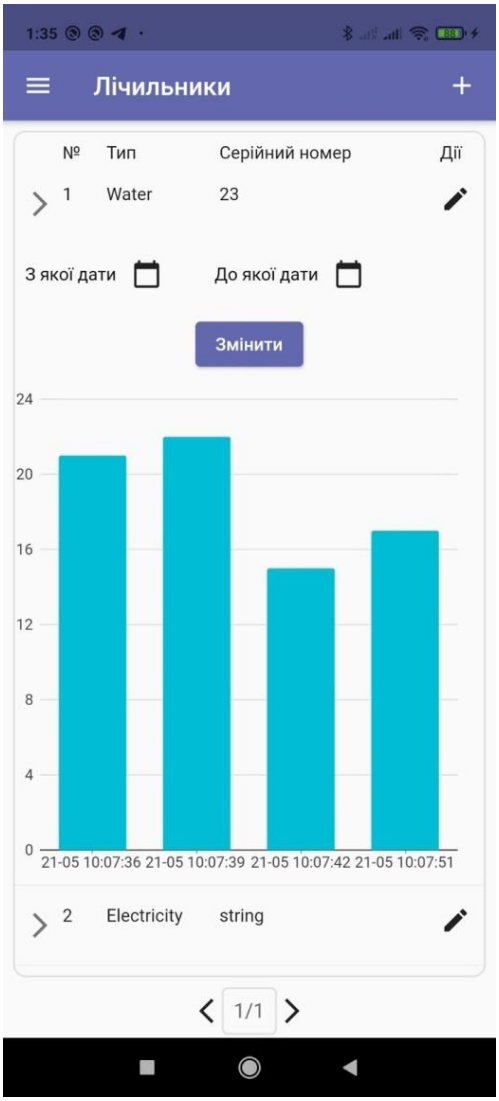


Рисунок 5.9 — Статистика по лічильнику

Для відображення усіх даних лічильників (внесених в систему), необхідно натиснути в меню на ‘‘Дані лічильників’’. Після цього користувач автоматично перейде на сторінку з усіма даними (рисунок 5.10).

Відображається номер запису, тип лічильника, значення на лічильнику та дата внесення цих даних.

При необхідності ці дані можна відредагувати, натиснувши на відповідну іконку редагування справа від дати в рядку.

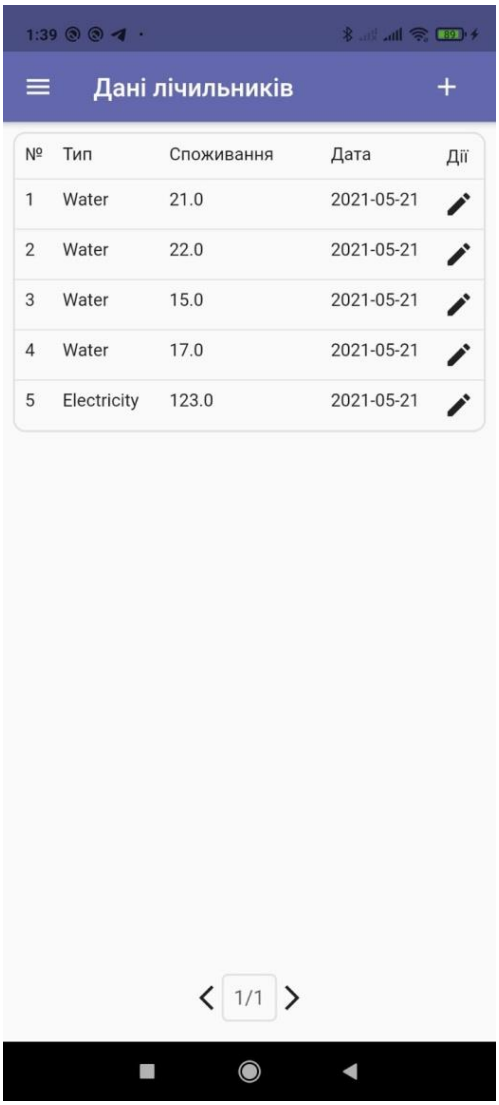


Рисунок 5.10 — Дані лічильників

За допомогою мобільного додатку можна додати нові дані лічильника (рисунок 5.11), натиснувши на іконку додати в правому верхньому кутку сторінки.

Для внесення даних необхідно вписати серійний номер лічильника, значення на лічильнику у вибраний момент часу та дату зняття (оскільки дата зняття може відрізнятись від дати внесення цих даних).

Після внесення даних потрібно натиснути кнопку ‘‘Зберегти’’ внизу діалогового вікна. Без цього внесені дані не будуть збережені.

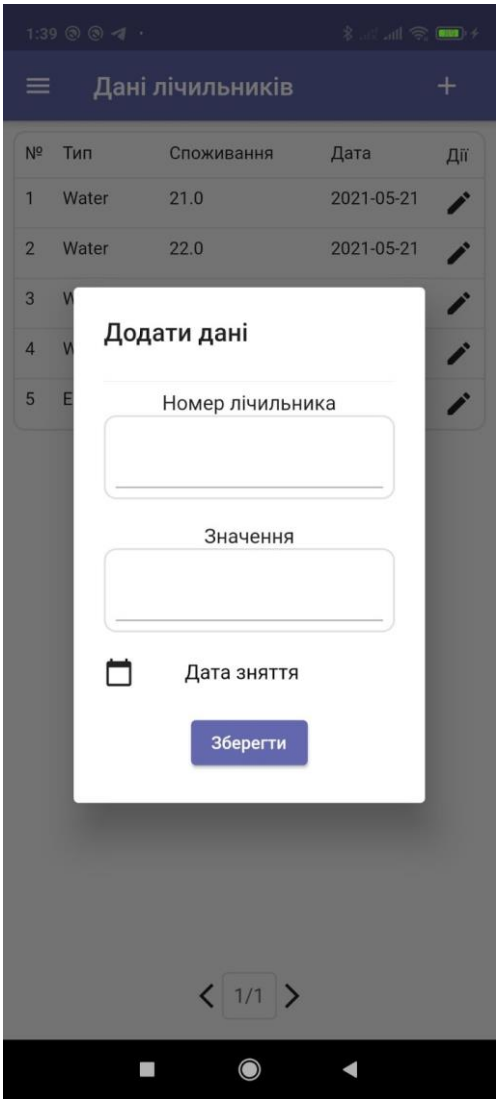


Рисунок 5.11 — Додавання нових даних

Якщо дану було внесено неправильно (текст замість значення) або неправильна дата, то система не приймає такий запис і відхиляє його.

Таким чином можна швидко і просто внести дані лічильника з мобільного телефону будь-якою відповідальною особою.

5.5 Висновки до розділу 5

В розділі 5 було продемонстровано розроблений інтерфейс програмного забезпечення, а саме:

1. Вхід користувача у систему та вихід з неї;
2. Профіль користувача, редагування профілю, контактні дані;
3. Перегляд документів та договорів, завантаження та вивантаження;
4. Введення даних лічильників.

Було представлено відповідні знімки екрану користувача та пояснено роботу окремих елементів інтерфейсу.

ВИСНОВКИ

Під час написання дипломної роботи було створено мобільний додаток за допомогою фреймворку Flutter та мови програмування Dart, який допомагає енергоменеджерам факультетів вносити нові дані та статистику лічильників, переглядати старі дані, завантажувати та переглядати необхідні документи та договори. В ході розробки дипломної роботи було:

1. Проаналізовано існуючі програмні рішення розв’язання задач енергоменеджмента та подібні системи. Сформовано задачі роботи для розробки необхідного програмного забезпечення та його запуску.

2. Визначено переваги та недоліки застосованої архітектури та функціоналу в існуючих системах;

3. Розроблено архітектуру програмного забезпечення мобільного додатку, яка буде враховувати проаналізовані переваги та недоліки архітектур існуючих систем. Для кращого уявлення архітектури системи було створено діаграму класів, діаграму прецедентів та відображено файлову систему та використані залежності.

4. Розроблено основний інтерфейс системи мобільного додатку, схожий на інтерфейс веб-додатку. Було представлено зручний інтерфейс користувача, який створено за допомогою фреймворку Flutter. Також створений мобільний додаток може працювати на різних платформах. Зручний інтуїтивно зрозумілий інтерфейс – запорука масштабності використання програми у наш час.

5. Реалізовано зв’язок з існуючим сервером та мікросервісами, розробленими для цієї системи. Для виведення даних та збереження їх додаток побудований у вигляді інтерфейсу API RESTful. Таким чином мобільний додаток має зв’язок із серверною частиною, яка приймає запити та відправляє на них необхідну інформацію у вигляді JSON за допомогою запитів GET, POST, DELETE, PATCH. Для того, щоб система не була перенавантажена, для кожного

модуля було розроблено сервіс, та один клас, що взаємодіє безпосередньо з сервером, та який надсилає на нього необхідні запити, та, відповідно, отримує інформацію.

6. Протестовано розроблену систему, використовуючи різні вхідні та вихідні дані, на різних операційних системах та пристроях.

7. Розроблено мобільний додаток для:

− Внесення даних про затрачені енергоресурси та водоресурси енергоменеджерами факультетів.

− Відображення інформації про місячне газо-, водо- та енергоспоживання. Інформація надається у вигляді стовпчастої діаграми та таблиці з даними.

− Відображення необхідної документації та договорів по енергопостачанню. Можливість завантаження на пристрій та з пристрою.

− Зручного і простого користування будь-якою відповідальною особою. Простий інтуїтивно зрозумілий інтерфейс.

− Доступу лише зареєстрованим (дозволеним) користувачам.

Використовувалось середовище розробки Android Studio, оскільки воно має всі необхідні інструменти для ефективного створення та тестування програмного забезпечення.

Додаток легко інтегрується з будь-яким сервером даних. За необхідності його можна масштабувати, додавши необхідний функціонал та розробивши інтерфейс.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 1С:Підприємство [Електронний ресурс] — Режим доступу: http://1c.ua/ua/
2. 1С:Предприятие 8.2. Практическое пособие разработчика. Примеры и типовые приемы / Е. Ю. Хрусталева, М. Г. Радченко // 1С-Паблишинг, 2009. — 876 с.
3. Android Studio [Електронний ресурс] — Режим доступу: https://developer.android.com/studio
4. Dart in Action / Chris Buckett [Електронний ресурс] // Manning Publications Co., 2013 — 436 p. — Режим доступу: https://www.programmer-books.com/wp-content/uploads/2018/12/Dart-in-Action.pdf
5. Flutter Framework [Електронний ресурс] — Режим доступу: https://flutter.dev
6. Introducing JSON [Електронний ресурс] — Режим доступу: https://www.json.org/json-en.html
7. Learning Android Application Development [Електронний ресурс] / Raimon Ràfols Montané, Laurence Dawson // Packt Publishing Ltd., 2016 —313 p. — Режим доступу: https://www.programmer-books.com/wp-content/uploads/2019/05/Learning-Android-Application-Development.pdf
8. Microsoft Excel [Електронний ресурс] — Режим доступу: https://www.microsoft.com/ua-ua/microsoft-365/excel
9. Swagger open source [Електронний ресурс] — Режим доступу: https://swagger.io/

ДОДАТОК А

Мобільний модуль додатку «Автоматизоване робоче місце енергоменеджера»

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ7105_21Б

Аркушів 2

Київ — 2021

Таблиця 2. Специфікація

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ7105_21Б	Записка_Дробаха.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ7105_21-1	Service	Компонент, що виконує з’єднання з сервером
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ7105_21-2	Widgets	Компонент, в якому створено користувацькі віджети
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ7105_21-3	Screens	Компонент, в якому створено екрани інтерфейсу

ДОДАТОК Б

Мобільний модуль додатку «Автоматизоване робоче місце енергоменеджера»

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ7105_21Б 12-1

Аркушів 10

Київ — 2021

home.dart

```
import 'package:diploma/screens/login.dart';
import 'package:diploma/services/userService.dart';
import 'package:diploma/widgets/menu.dart';
import 'package:flutter/material.dart';

class Home extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    UserService userService = UserService();
    return FutureBuilder(
        future: userService.getUserMap(),
        builder: (context, snapshot) {
          if (!snapshot.hasData) return CircularProgressIndicator();
          if (snapshot.data != null)
            return Scaffold(
              appBar: AppBar(
                title: Text("Домашня сторінка"),
              ),
              drawer: Menu(),
              body: Container(
                padding: const EdgeInsets.all(8.0),
                  child: Text(
                "Вітаємо, " + snapshot.data["first_name"] + "!",
                style: TextStyle(fontSize: 20), textAlign:
TextAlign.center,
              )),            );
          else {
            return Login();
          }
        });
  }}
```

loginScreen.dart

```
import 'package:diploma/screens/home.dart';
import 'package:diploma/services/loginService.dart';
import 'package:diploma/services/serverConnection.dart';
import 'package:flutter/material.dart';

class Login extends StatelessWidget {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController =
TextEditingController();
  @override
  Widget build(BuildContext context) {
    LoginService loginService = LoginService();
    return Scaffold(
      appBar: AppBar(title: Text("Вхід у систему"),
automaticallyImplyLeading: false,),
        body: Padding(
          padding: const EdgeInsets.all(8.0),
          child: Column(
            children: <Widget>[
              TextField(
                controller: _emailController,
                decoration: InputDecoration(
                    labelText: 'Email'
                ),
              ),
              TextField(
                controller: _passwordController,
                obscureText: true,
                decoration: InputDecoration(
                    labelText: 'Password'
                ),
              ),
```

```
              FlatButton(
                  onPressed: () async {
                    if (await
loginService.getUserAccess(_emailController.text,
_passwordController.text)) {
                      Navigator.push(context,
                        MaterialPageRoute(builder: (context) => Home()),
                      );
                    } else {
                      Navigator.push(context,
                        MaterialPageRoute(builder: (context) => Login()),
                      );
                    }
                    },
                  child: ElevatedButton(child: Text("Log In"))
              ),
            ],
          ),
        )
    );
  }
}
```

sereverConnection.dart

```
import 'dart:async';
import 'dart:convert';
import 'dart:core';

import 'package:flutter_secure_storage/flutter_secure_storage.dart';
import 'package:http/http.dart' as http;

final storage = FlutterSecureStorage();
String token;
```

```
String refresh;

class ServerConnection {
  Future<String> jwtOrEmpty() async {
    token = await storage.read(key: "token");
    await tokenRefresh().catchError((e) => {token = null});
    refresh = await storage.read(key: "refresh");
    if (refresh == null || token == null) {
      return "";
    }    return token;
  }

  Future<Map> apiPost(String url, Map data) async {
    var body = json.encode(data);
    var response = await http.post("$url",
        headers: {"Content-Type": "application/json"}, body: body);
    if (response.statusCode == 401) {
      await tokenRefresh();
      response = await http.get(url, headers: {
        "Authorization": "Bearer $token"});
    }
    if (response.statusCode == 200) {
      token = jsonDecode(response.body)['access'];
      refresh = jsonDecode(response.body)['refresh'];
      storage.write(key: "token", value: token);
      storage.write(key: "refresh", value: refresh);
      return jsonDecode(response.body);
    }    return null;
  }

  Future tokenRefresh() async {
    refresh = await storage.read(key: "refresh");
    await http.post("http://93.188.34.235:8001/token/refresh/",
```

```
        headers: {"Content-Type": "application/json"}, body:
jsonEncode({"refresh": refresh})).then((value) => {token =
jsonDecode(value.body)['access'],
        storage.write(key: "token", value: token)}).catchError((e) =>
{throw Exception(e)});
  }

  Future<Map> apiGet(String url) async {
    token = await storage.read(key: "token");
    var response = await http.get(url, headers: {
    "Authorization": "Bearer $token"});
    if (response.statusCode == 401) {
      await tokenRefresh();
      response = await http.get(url, headers: {
        "Authorization": "Bearer $token"});
    }    return jsonDecode(utf8.decode(response.bodyBytes));
  }

  Future<List> apiGetList(String url) async {
    token = await storage.read(key: "token");
    var response = await http.get(url, headers: {
      "Authorization": "Bearer $token"});
    if (response.statusCode == 401) {
      await tokenRefresh();
      response = await http.get(url, headers: {
        "Authorization": "Bearer $token"});
    }    return jsonDecode(utf8.decode(response.bodyBytes));
  }

  Future<Map> apiPatch(String url, Map data) async {
    var body = json.encode(data);
    var response = await http.patch("$url",
        headers: {"Content-Type": "application/json"}, body: body);
    if (response.statusCode == 401) {
```

```
      await tokenRefresh();
      response = await http.get(url, headers: {
        "Authorization": "Bearer $token"});
    }
    if (response.statusCode == 200) {
      return jsonDecode(response.body);
    }    return null;
  }}
```

loginService.dart

```
import 'package:diploma/services/serverConnection.dart';
const url = 'http://93.188.34.235:8001/';
class LoginService {
  ServerConnection serverConnection = ServerConnection();
  Future<bool> getUserAccess(String email, String pass) async {
    print("here");
    var res = await serverConnection.apiPost(url+"login/", {"email": email,
"password": pass});
    if (res != null) return true;
    else return false;
  }}
```

menu.dart

```
import 'package:diploma/screens/contracts.dart';
import 'package:diploma/screens/dataSencors.dart';
import 'package:diploma/screens/documentations.dart';
import 'package:diploma/screens/login.dart';
import 'package:diploma/screens/profile.dart';
import 'package:diploma/screens/sensors.dart';
import 'package:diploma/services/userService.dart';
import 'package:eva_icons_flutter/eva_icons_flutter.dart';
import 'package:flutter/cupertino.dart';
```

```
import 'package:flutter/material.dart';

Map<int, Color> color = {
  50: Color.fromRGBO(136, 14, 79, .1),
  100: Color.fromRGBO(136, 14, 79, .2),
  200: Color.fromRGBO(136, 14, 79, .3),
  300: Color.fromRGBO(136, 14, 79, .4),
  400: Color.fromRGBO(136, 14, 79, .5),
  500: Color.fromRGBO(136, 14, 79, .6),
  600: Color.fromRGBO(136, 14, 79, .7),
  700: Color.fromRGBO(136, 14, 79, .8),
  800: Color.fromRGBO(136, 14, 79, .9),
  900: Color.fromRGBO(136, 14, 79, 1),
};

class Menu extends StatelessWidget {
  UserService userService = UserService();
  @override
  Widget build(BuildContext context) {
    print(userService.getUserMap());
    return FutureBuilder(
        future: userService.getUserMap(),
        builder: (context, snapshot) {
          if (!snapshot.hasData) return CircularProgressIndicator();
          if (snapshot.data != null)
            return Drawer(
              elevation: 20.0,
              child: ListView(
                padding: EdgeInsets.zero,
                children: <Widget>[
                  UserAccountsDrawerHeader(
                    accountName: Text(
                        "${snapshot.data["first_name"]}
${snapshot.data["last_name"]}"),
```

```
  accountEmail: Text("${snapshot.data["email"]}"),
  currentAccountPicture: Container(
    decoration: BoxDecoration(
      shape: BoxShape.circle,
      image: DecorationImage(
          image: snapshot.data["photo_url"] != null
              ? NetworkImage(snapshot.data["photo_url"])
              : AssetImage('images/user.png'),
          fit: BoxFit.fill),
    ),
  ),
  decoration: BoxDecoration(
    color: MaterialColor(0xff6367AD, color),
  ),
),
ListTile(
  leading: Icon(Icons.account_circle_outlined),
  title: Text('Профіль'),
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => Profile()),
    );
  },
),
Divider(
  height: 2.0,
),
ListTile(
  leading: Icon(Icons.sensors),
  title: Text('Лічильники'),
  onTap: () {
    Navigator.push(
      context,
```

```
                    MaterialPageRoute(builder: (context) => Sensors()),
                );
              },
            ),
            Divider(
              height: 2.0,
            ),
            ListTile(
              leading: Icon(Icons.analytics_outlined),
              title: Text('Дані лічильників'),
              onTap: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) =>
DataSensors()),
                );
                },
            ),
            Divider(
              height: 2.0,
            ),
            ListTile(
              leading: Icon(Icons.article_outlined),
              title: Text('Документація'),
              onTap: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(
                      builder: (context) => Documentations()),
                );
              },
            ),
            Divider(
              height: 2.0,
```

```
                ),
                ListTile(
                  leading: Icon(Icons.insert_drive_file_outlined),
                  title: Text('Договори енергопостачання'),
                  onTap: () {
                    Navigator.push(
                      context,
                      MaterialPageRoute(builder: (context) =>
Contracts()),
                    );
                  },
                ),
                SizedBox(
                  height: 20,
                ),
                ListTile(
                  leading: Icon(Icons.login_outlined),
                  title: Text('Вийти'),
                  onTap: () {
                    Navigator.push(
                      context,
                      MaterialPageRoute(builder: (context) => Login()),
                    );
                  },
                ),
              ],              ),
            // Container(
            //   child: Text("Log Out"),
            // )
          );
        else
          return Login();
      });
  }}
```

ДОДАТОК В

Мобільний модуль додатку «Автоматизоване робоче місце енергоменеджера»

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ7105_21Б 14-1

Аркушів 10

Київ — 2021

АНОТАЦІЯ

Додаток містить опис мобільного додатку «Автоматизоване робоче місце енергоменеджера», що виконує деякі завдання, визначенні в розділі 1, а саме:

- Внесення даних про затрачені енергоресурси та водоресурси енергоменеджерами факультетів;
- Відображення інформації про місячне водо- та енергоспоживання;
- Перегляду даних лічильників;
- Відображення необхідної документації та договорів по енергопостачанню;
- Зручного і простого користування будь-якою відповідальною особою;
- Доступу лише зареєстрованим (дозволеним) користувачам.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У додатку міститься опис основних компонентів мобільного додатку «Автоматизоване робоче місце енергоменеджера», визначенні в розділі 1. Додаток Б містить програмний код цих компонентів.

Для роботи мобільного за стосунку енергоменеджера необхідно встановити його на мобільний телефон з операційною системою Android або iOS.

Необхідно мати підключення до Інтернету під час роботи з мобільним додатком.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений мобільний застосунок надає користувачам наступний функціонал:

- Внесення даних про затрачені енергоресурси та водоресурси енергоменеджерами факультетів;
- Відображення інформації про місячне газо-, водо- та енергоспоживання;
- Перегляду даних лічильників;
- Відображення необхідної документації та договорів по енергопостачанню;
- Зручного і простого користування будь-якою відповідальною особою;
- Доступу лише зареєстрованим (дозволеним) користувачам.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Основних структурних модулів у боті 5:

- Модуль взаємодії з Telegram-сервером
- Модуль обробки оновлень. В ньому ж буде прописана логіка боту.
- Модуль роботи з базою даних (ORM)
- Модуль для роботи з текстовими та чисельними константами, що використовуються у боті.
- Модуль для роботиAPI

Модуль взаємодії з Telegram отримує та відправляє запити на Telegram-сервер. Також він запускає потоки для обробки вхідних запитів модулем обробки оновлень.

Модуль обробки оновлень відповідає за логіку роботи бота. Також він зв’язує усі інші модулі в єдину систему. Даний модуль є головним та саме він є виконавчим при старті боту.

Модуль роботи з базою даних керує базою даних завдяки використанню бібліотеки peewee.

Модуль для роботи з текстовими та чисельними константами є технічним та створений для більш зручної організації коду проекту. Його суть полягає у тому, що при ініціалізації йому надається шлях до директорії з текстовими файлами, що використовуються у проекті. У самих файлах записані текстові константи з іменами, по яким до них можна доступитися. Тобто зручність у тому, що усі текстові константи записані в одному місці, що полегшує локалізацію та редагування інтерфейсу боту. Те ж саме, відповідно, з константами проекту. Як токен боту, id головного модератора тощо.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для роботи мобільного за стосунку енергоменеджера необхідно встановити його на мобільний телефон з операційною системою Android або iOS.

Необхідно мати підключення до Інтернету під час роботи з мобільним додатком.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Після того як мобільний додаток було встановлено на мобільний телефон за допомогою файлу app-release.apk необхідно його відкрити на телефоні. Після цього користувач автоматично перейде на сторінку авторизації, де потрібно ввести коректні електронну адресу та пароль. Якщо дані внесено правильно, тоді користувач автоматично перейде на головну сторінку.

Після цього необхідно натиснути на кнопку меню в лівому верхньому кутку та відкрити меню. Натиснувши на потрібну позицію меню, користувач перейде на обрану сторінку.

З боку адміністратора та розробника, для дебагу та тестування, необхідно відкрити програмний код в середовищі розробки Android Studio та запустити його через емулятор або на своєму, підключеному до пристрою мобільний телефон.

ВХІДНІ ДАНІ

Вхідна інформація для мобільного додатку:

- Дані користувача (контактні дані, ім’я, прізвище, фото);
- Документація;
- Договори енергопостачання;
- Дані лічильників;

ВИХІДНІ ДАНІ

Вихідна інформація для користувача:

- Документація, збережена на сервері (документи в різних форматах);
- Договори енергопостачання (документи в різних форматах;
- Статистика по лічильнику (стовпчаста діаграма);
- Дані лічильників (числові дані);
- Дані профілю користувача.