ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ

ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

до бакалаврської роботи на

ступінь вищої освіти бакалавр

на тему: «**Розробка програмного забезпечення для автоматизації обліку сплати комунальних послуг громадян засобами мови C#**»

Виконав: студент 4 курсу, групи ПД– 41

спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Панібратов А.І.

(прізвище та ініціали)

Керівник Золотухіна О.А.

(прізвище та ініціали)

Рецензент_____

(прізвище та ініціали)

Нормоконтроль _____

(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ О.В. Негоденко

“

”_____” 2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Панібратову Андрію Івановичу
(прізвище, ім‘я, по батькові)

1. Тема роботи: «Розробка програмного забезпечення для автоматизації обліку сплати комунальних послуг громадян засобами мови C#»

Керівник роботи К.т.н., доц., доцент кафедри ІПЗ Золотухіна Оксана Анатоліївна
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “16” лютого 2022 року №. 22

2. Строк подання студентом роботи 06.06.2022.
3. Вихідні дані до роботи:
   - 3.1. Положення побудови програми для обліку;
   - 3.2. Методи побудови програми для обліку;
   - 3.3. Існуючі засоби для сплати комунальних послуг;
   - 3.4. Науково-технічна література
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   - 4.1.Огляд архітектури існуючих інструментів оплати комунальних послуг
   - 4.2.Розробка структури додатку на C#
   - 4.3. Програмна реалізація додатку
   - 4.4. Приклади використання та тестування системи
5. Перелік графічного матеріалу.
   - 5.1.Аналоги
   - 5.2. Мета, об’єкт та предмет дослідження

5.3. Технічне завдання
5.4. Програмні засоби реалізації
5.5. Діаграма варіантів використання
5.6. Блок-схеми головних функцій
5.7. ER-діаграма бази даних
5.8. Екранні форми
5.9. Апробація результатів дослідження
5.10. Висновки

6. Дата видачі завдання: 19.04.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	01.04-07.05	Виконано
2	Дослідження існуючих інструментів для автоматизації тестування ПЗ	01.05-07.05	Виконано
3	Проектування архітектури системи	01.05-07.05	Виконано
4	Розробка системи для автоматизації обліку комунальний платежів	07.05-14.05	Виконано
5	Висновки, оформлення роботи	14.05-21.05	Виконано
6	Розробка демонстраційних матеріалів	21.05-30.05	Виконано
7	Попередній захист роботи	01.06.2021	
8	Здача роботи	06.06.2021	

Студент _____ Панібратов А.І.
(підпис) (прізвище та ініціали)

Керівник роботи_____ Золотухіна О.А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 54 с., 38 рис., 1 таблиця, 16 джерел.

Об'єкт дослідження – процес обліку сплати комунальних послуг громадян.

Предмет дослідження – програмне забезпечення для автоматизації обліку сплати комунальних послуг громадян.

Мета роботи – підвищення зручності обліку сплати комунальних послуг громадян за рахунок автоматизації процесу із використанням програмного забезпечення

Методи дослідження – методи об'єктно-орієнтованого програмування, методи моделювання програмного забезпечення, методи проектування та розробки комп'ютерних додатків, методи тестування комп'ютерних додатків.

В роботі проведено аналіз особливостей обліку сплати комунальних платежів. Розглянуто основні етапи процесу та ключові характеристики кожного з етапів, а також існуючі програмні та інші засоби для ведення обліку комунальних платежів. Визначено функціональні вимоги до додатку. Виконано моделювання програми із використанням діаграм UML.

Спроектовано та реалізовано інтерфейс додатку для автоматизації обліку сплати комунальних послуг громадян. Розроблено програмне забезпечення мовою C# із використанням платформи .NET.

Галузь використання – користувачі, що зацікавлені у веденні автоматизованого обліку сплати комунальних платежів.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ОБЛІК, КОМУНАЛЬНІ ПОСЛУГИ, КОМП'ЮТЕРНИЙ ДОДАТОК, ТАБЛИЦІ, СТАТИСТИЧНА ІНФОРМАЦІЯ, C#

ЗМІСТ

ВСТУП

У сучасному світі усе більше й більше завдань автоматизуються. Комп'ютери, завдяки своїй швидкодії та надійності здатні виконувати завдання значно швидше за людину, допускаючи при цьому меншу кількість помилок. Тому з кожним днем нові й нові завдання автоматизуються, поліпшуючи якість роботи та збільшуючи швидкість її виконання. З розвитком технологій, швидкість та точність виконання завдань комп'ютерами буде лише рости, тому те, що ми маємо на сьогоднішній день – лише початок.

У ситуації, коли варто проводити облік якихось даних відсутність помилок стає особливо важливою. В залежності від терміновості та об'єму роботи, швидкість також може мати важливу роль. Навіть якщо не брати до уваги цілком очевидне покращення якості роботи, такі рутинні завдання є досить нецікавими для виконання. А тому цілком логічно передати такі типи роботи комп'ютеру на опрацювання.

Головна мета - підвищення зручності обліку сплати комунальних послуг громадян за рахунок автоматизації процесу із використанням програмного забезпечення. Під час виконання дослідження об'єктом став процес обліку сплати комунальних послуг громадян. Предметом дослідження стало програмне забезпечення для автоматизації обліку сплати комунальних послуг громадян.

В роботі розглянуто процес створення додатку для автоматизації обліку комунальних послуг та розглянуті аналоги вже існуючих додатків. Визначено головні функції додатку, та розроблено технічні вимоги до нього. Створено додаток, що здатний автоматизувати процес обліку комунальних платежів для користувача та відповідає поставленим технічним вимогам. Додаток розроблено з використанням мови програмування C# та функціоналу Windows Forms.

1. ОГЛЯД АРХІТЕКТУРИ ІСНУЮЧИХ ІНСТРУМЕНТІВ АВТОМАТИЗАЦІЇ ОБЛІКУ СПЛАТИ КОМУНАЛЬНИХ ПЛАТЕЖІВ

1.1 Електронна оплата комунальних платежів

Комунальні платежі є рутиною для кожного громадянина майже будь-якої країни. У минулому вони, здебільшого, проводились у банках, на пошті або у спеціалізованих відділеннях створених спеціально для проведення подібних операцій. Платіжки доставлялися листоношею і, доволі часто, банально не доходили до громадян, створюючи проблеми з веденням обліку оплат.

На наше щастя ці часи давно пройшли. У наші дні платежі можна спокійно проводити не виходячи з дому. Більше того, навіть стан своїх рахунків, історію минулих платежів та заборгованостей можна подивитись там же. Усе це стало можливим завдяки мережі Інтернет. Використовуючи усі можливості цієї мережі ми можемо за лічені хвилини пройти весь процес, включаючи не тільки оплату послуг, а й безліч інших варіантів, виконання яких стало можливим завдяки використанню обчислювальних можливостей сучасних комп’ютерів.

Надаючи перевагу електронним платежам над застарілим методам оплати ми досягаємо одразу декілька цілей. Перша з них – значна економія часу користувача. Завдяки тому, що користувачі мають можливість працювати з нашими платіжками одразу у їхньому персональному комп’ютері або смартфоні вони значно заощаджуємо час який в іншому випадку вони би витратили на отримання платіжок в паперовому варіанті, а потім на подорож до точки сплати.

Друга досягнута ціль – це зручність. Користувачу не потрібно змінювати його рутину на зайві дії. Достатньо лише зайти у необхідний додаток або перейти на потрібний сайт, зробити декілька кліків – і він отримує бажаний результат.

Третя перевага – можливість зберігати минулі оплати/показання лічильників, що дозволяє нам автоматизувати та систематизувати роботу з платіжками. Це особливо важливо у тому випадку, коли користувач хоче перевірити свої минулі платіжки або у випадку необхідності передачі показань іншим людям. Користувачу більше не потрібно зберігати купу паперів – уся інформація буде зберігатися на девайсі, завжди доступна для використання у зручному форматі.

Четверта перевага – значне полегшення роботи з платіжками при наявності одночасно декількох об’єктів. При використанні не-електронних методів проведення та зберігання платежів/передачі показань лічильників для декількох об’єктів людина може заплутатися у наявній інформації через велику кількість необхідних паперів. Платежі можуть бути занесені не для тих об’єктів або невірно обраховані. У електронних додатках більшість дій виконуються автоматично, а ті що потребують дій зі сторони користувача мають зручну організацію, що зводить можливість помилки до нуля.

Остання перевага на яку потрібно звернути увагу – можливість значно полегшити розрахунок витрат у минулому. Завдяки доступності тарифів та показань лічильників у минулих місяцях (або сумах витрат, якщо ці суми є незмінними платежами) користувач може легко розрахувати свої витрати у поточному чи наступному місяці, сворити прогноз річного бюджету, тощо.

Що особливо приємно – єдиний наявний недолік це необхідність наявності персонального комп’ютеру (чи будь-якого іншого девайсу у залежності від потребуємої платформи) та додатку, що має функціонал потрібний для виконання бажаних дій по роботі з комунальними платежами. При наявності обох елементів, електронний метод оплати є набагато кращим за оплату звичайними способами.

Так як аналіз вказує на перевагу такого методу, буде сконцентровано увагу на наступному важливому кроці при його використанні – виявленню необхідного функціоналу додатку.

1.2 Типи програмного забезпечення по оплаті комунальних платежів

Кожен додаток по роботі з комунальними платежами надає по своєму унікальні послуги для користувача. Це може бути виражене у платформі використання, особливостях інтерфейсу та подання інформації або видах доступного для користувача функціоналу додатку, проте зазвичай усі застосунки можна поділити на різноманітні підтипи. З них можна виділити 3 основних, які зустрічаються найчастіше.

Перший з підтипів це Web-додаток. Головна особливість такого додатку полягає у його розміщенні в мережі інтернет. Це дозволяє користуватися таким додатком на будь-якому пристрої та будь якій системі, за умови що така платформа підтримує можливість використання інтернету та має інтернет-браузер. Варто зауважити, що Web-додатки розробляються під певні браузери та можуть некоректно працювати при використанні несумісного браузеру. Важливим недоліком такого виду додатків, як не складно догадатися, є необхідність підключення до мережі інтернет, адже без наявності зв'язку такий застосунок просто не буде працювати.

В Україні державні додатки зазвичай належать саме до цього типу. Чудовим прикладом є Центр Комунальних Послуг, GIOC, такі комунальні підприємства як КиївВодоКанал та КиївТеплоЕнерго, Yasno. Можна впевнено сказати, що для цих компаній легка доступність для користувача є головною причиною обрання саме цього типу застосунку.

Web-додатки для роботи з комунальними платежами, здебільшого, пишуться на мові HTML, рідше на мовах XHTML або WML. Для покращення графічного вигляду сайту та додання додаткового функціоналу також використовуються CSS, PHP, Ruby, Java Script, VB Script, тощо [1].

Другим типом є доволі стандартний додаток для використання на персональному комп'ютері. Такий тип застосунку вимагає завантаження на робочий апарат користувача. Це може бути не так зручно, проте дозволяє реалізувати більш складний функціонал та надає можливість користувачу використовувати додаток вне залежності від підключення до мережі інтернет. Такі додатки зазвичай прив'язані до певної оперативної систему, за виключенням мультиплатформенних додатків.

Додатки такого типу можуть бути написані на різноманітних мовах програмування. На даний момент доволі популярним є використання мови програмування C# з платформою .NET.

Третім підтипом є мобільні додатки, створені для використання на таких мобільних пристроях як смартфони та планшети. Завдяки зручності користування такими пристроями та їх розповсюдженості, додатки орієнтовані на роботу на даних платформах набирають чималу популярність. Головною проблемою таких додатків є їх сумісність з девайсом користувача (адже тут потрібно брати на увагу не лише операційну систему , а й технічні характеристики) та технічні обмеження функціоналу що можна реалізувати.

Ці додатки також мають чимало прикладів. Один з них – додаток "Дах", що дозволяє користувачу керувати своєю нерухомістю прямо з екрану мобільного пристрою. Пишуться мобільні додатки на різноманітних мовах програмування. Серед найбільш поширених – Java, C#. [2]

1.3 Аналіз загальних вимог до програмного забезпечення

Головна мета цієї роботи – створення програмного забезпечення, здатного автоматизувати облік комунальних платежів, та виконувати різноманітні дії з даними, що користувач заносить у систему. У переліку необхідних дій, до яких має

мати доступ користувач можна виділити необхідний для реалізації у додатку функціонал:

а)надати користувачу посилання для швидкого переходу на сайти оплати;

б)надавати можливість користувачу вносити минулі/поточні показники у базу даних для подальшого аналізу;

в)створити функціонал для занесення щомісячних платежів у систему для подальшого зберігання;

г)створити систему, що надасть можливість передивлятися актуальні тарифи;

г)створити функціонал для ведення звітності по минулим оплатам за послуги;

д)створити функціонал для ведення звітності по минулим показанням лічильників.

Додання цього функціоналу дозволить створити більш централізований додаток, що значно пришвидшить роботу користувача у цій сфері, зменшить вірогідність допуску помилки та надасть можливість зберігати дані по минулим виконанням цих робіт у доступній для швидшого аналізу користувачем формі. Це має збільшити привабливість додатку для користувача.

Створене програмне забезпечення орієнтується на те, що їм буде користуватися специфічне коло людей, а отже при розробці програми в першу чергу будуть братися потреби, що виникають у локальних зонах. Під цим розуміється те, що створені посилання будуть вести на сайти місцевих компаній, при розрахунках вартості будуть використовуватися тарифи, що актуальні для певного міста або регіону. При цьому додаток має зберігати деяку гнучкість у механізмі своєї роботи, адже умови галузі, а саме роботи з комунальними платіжками, доволі часто змінюються. Це значить, що користувач обов’язково має мати можливість змінювати тарифи або суми регулярних платежів за виникненням потреби.

При роботі над цією програмою фокус буде робитися на дві головні мети: функціонал, що має поєднувати різноманітні функції по опрацюванню та роботі з комунальними платіжками, включаючи їх внесення, обробку, зберігання, обчислення, тощо, а також на створенні зручного інтерфейсу, що дозволить користувачу з легкістю використовувати програмне забезпечення.

Створення зручного інтерфейсу має особливо важливе значення, адже саме інтерфейс програми визначає наскільки легко користувач зможе використовувати дане програмне забезпечення. Інтерфейс – це перше, що користувач побачить у програмі і, доволі часто, користувач орієнтується його зручністю та інтуїтивністю обираючи між двома засобами для вирішення тої чи іншої проблеми. Головні складові успішного інтерфейсу можна описати у декількох пунктах [3]:

а)зручність — це якість, що визначає, наскільки простим у використанні є інтерфейс;

б)легкість опанування (Learnability) — тобто те, наскільки легко ваші користувачі можуть виконувати базові завдання під час першої взаємодії з продуктом;

в)ефективність (Efficiency) — тобто те, наскільки швидко користувач виконує в інтерфейсі задачі, опанувавши його;

г)запам’ятовуваність (Memorability) — це те, наскільки просто користувачеві буде згадати, як працювати з інтерфейсом, після тривалої перерви;

г)робота з помилками (Errors) — як часто користувачі припускаються помилок, наскільки серйозними є ці помилки, і наскільки легко користувачі можуть з ними впоратись;

д)задоволеність (Satisfaction) — наскільки приємним для користувача є дизайн продукту.

Поєднавши наші дві мети (функціонал додатку та його інтерфейс) отримаємо утилітарність додатку – можливості, які доступні користувачу.

1.4 Аналіз існуючого програмного забезпечення для автоматизації обліку оплати комунальних послуг

Оскільки комунальні платежі це те, з чим має справу майже кожний громадянин держави, нескладно здогадатися, що на ринку вже існують доступні аналоги. Зазвичай вони виражені як веб-додатки, що пов’язані з комунальним підприємством або як комерційні застосунки. Останні здебільшого можна знайти саме на мобільній платформі. Під час аналізу буде розглянуто 5 додатків, що надають можливість працювати з обліком комунальних платежів.

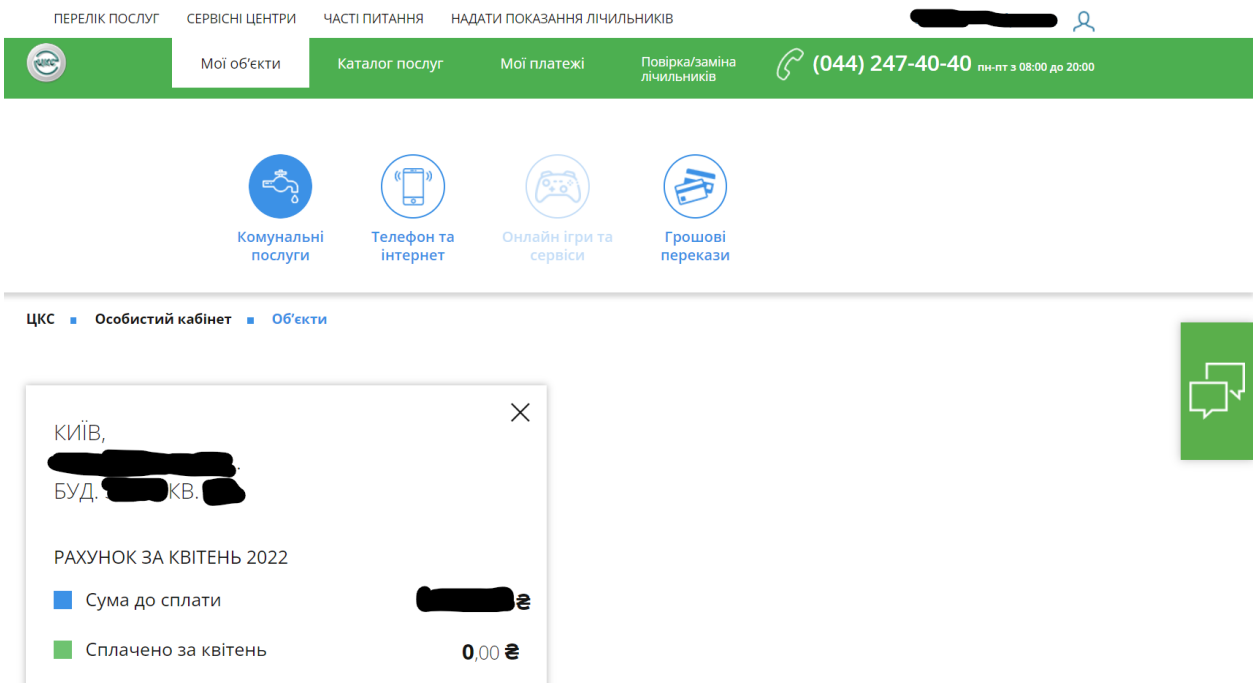


Рисунок 1.1 – Головне меню сайту ЦКС

Один з найвідоміших прикладів застосунку для роботи з комунальними платіжками є сайт Центру Комунальних Послуг (ЦКС)[4]. Його можна віднести до першого типу – web-додатків. Хоча загалом цей сайт виконує і інші ролі (наприклад на цьому сайті можна знайти новини), ЦКС має велику кількість інструментів для роботи з комунальними послугами та платіжками, що стають доступними для користувача за умови створення особистого кабінету. До них відносяться:

а)можливість оплати комунальних послуг;

б)передача показань лічильників;

в)можливість переглянути стан свого рахунку на наявність заборгованостей;

г)доступ до минулих платежів;

ґ)можливість оплати не лише комунальних платежів, а й інших, пов'язаних послуг (наприклад телебачення, телефону або інтернету);

д)замовити послуги з обслуговування.

Сайт надає користувачу можливість виконувати декілька дій одночасно, а зручний та дружній для користувача інтерфейс допомагає легко орієнтуватися на сайті. ЦКС дозволяє працювати з платіжками по великій кількості напрямків не переходячи на сайти окремих компаній. До таких дій відносяться не лише можливість оплати послуг, а й передача показників лічильників. Такий підхід до проведення оплат дозволяє користувачу економити багато часу, а також не сплачувати зайву комісію, а отже є гарним рішенням для користувача.

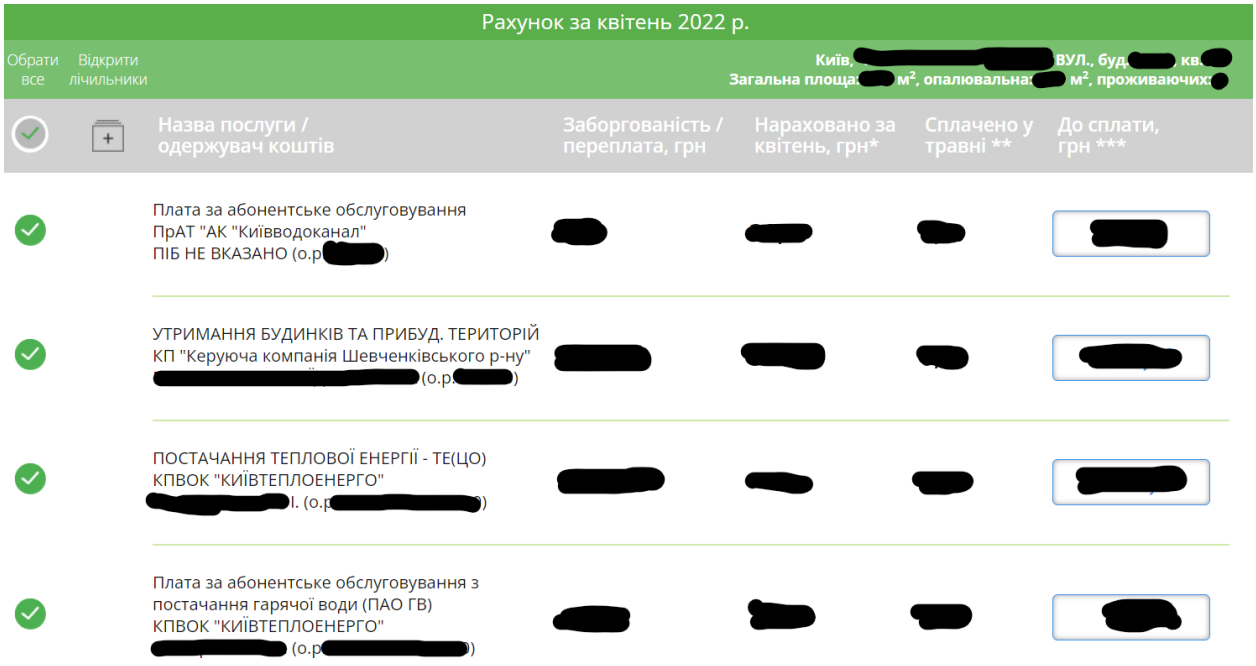


Рисунок 1.2 – Меню оплати

Цікавим рішенням можна назвати поєднання меню оплати комунальних платежів та передачі лічильників. Хоча це і є зручним рішенням при проведенні оплати, у ситуаціях коли користувач хоче лише передати показання це стає незручним недоліком через велику кількість зайвих даних. Велика кількість даних

на екрані може викликати дезоріентацію у користувача, знизити загальну уважність та, відповідно, приведе до збільшення кількості помилок

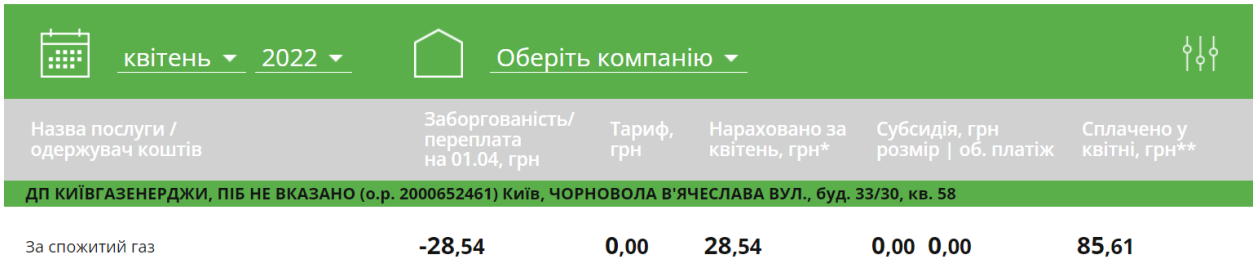


Рисунок 1.3 – Попередні оплати

Сайт також надає доступ до детальної інформації по попереднім оплатам, що дає користувачу можливість проводити аналіз по своїм минулим платіжкам за необхідності. Користувач може дивитися як історію загалом, так і історію по окремим категоріям оплати.

Загалом переваги та недоліки цього сайту досить очевидні. З переваг можна виділити:

а)зручний та зрозумілий інтерфейс;

б)доступ до детальної інформації по історії платежів;

в)автоматичне обрахування сум до сплати;

г)комбінація великої кількості.

З недоліків можна зазначити:

а)пов’язаність оплати та передачі показників;

б)відсутність можливості ручного обрахування суми оплати.

У той час як ЦКС надає можливість працювати з комунальними платіжками у широкому спектрі, існують і інші, більш спеціалізовані варіанти. Один із них – веб-сайт енергетичної компанії Yasno[5]. На відміну від різностороннього ЦКС, у цьому сайті весь функціонал спрямований на роботу з платіжками по електроенергії. Цей веб-додаток спрямований виключно на роботу з комунальними платіжками, а тому, на відміну від ЦКС, не має функціоналу що не пов'язаний з комунальними послугами. Вузька спеціалізація є одночасно як перевагою, так і

недоліком цього програмного забезпечення . З одного боку фокусування лише на комунальних платежах по електроенергії дозволяє надавати користувачу саме ту інформацію, що він потребує у момент сплати комунальних послуг по електроенергії. З іншого боку, така вузька спеціалізація створює необхідність використання інших додатків, що є доволі незручним фактором для користувача.

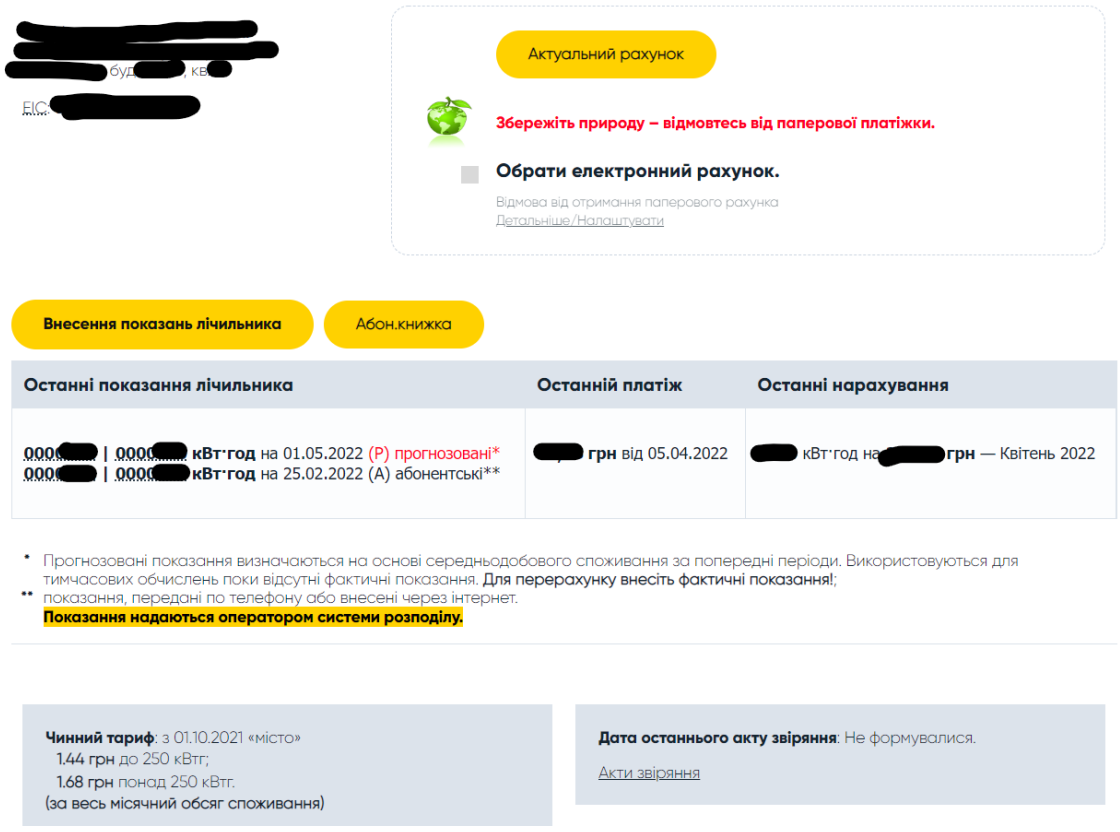


Рисунок 1.4 – Головне меню Yasno

Як можна побачити, на цьому сайті більшість елементів знаходяться прямо на головному меню. Це досить зручно для користувача у ситуаціях, коли детальний огляд ситуації не потрібен. Такий інтерфейс швидко та зрозуміло доносить усю необхідну інформацію користувачеві, у той час як додаткова інформація, така як показники з минулих місяців чи років, знаходиться у додаткових розділах. Чудовим прикладом є Абонентська Книжка, де якраз і зберігається така інформація у формі таблиці. Зберігання даних у вигляді таблиці допомагає користувачу отримати та опрацювати інформацію у малий проміжок часу. Комбінація таких шляхів

показання інформації показує орієнтацію інтерфейсу на максимальну ефективність та дружність до користувача.

Додаток Коммуналочка є мобільним додатком для ведення обліку комунальних платежів[6]. На відміну від складних веб-додатків, що мають багато доступних функцій, Коммуналочка фокусується на веденні простої звітності по наявним об’єктам. Користувач може вносити показники лічильників, вручну вводити тарифи, передивлятися оплати та вираховувати розміри поточної платіжки. Історія платежів доступна у помісячному форматі.

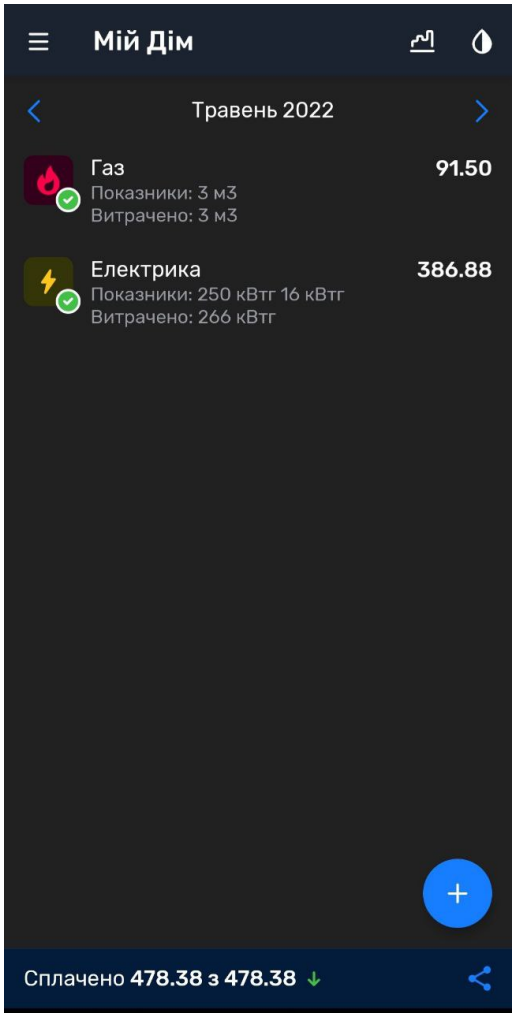


Рисунок 1.5 – Меню оплати у додатку “Коммуналочка”

Завдяки спеціалізованості цієї програми, користувач може сконцентруватися на потрібних йому функціях, не відволікаючись на непотрібні

показники. З іншої сторони це є й недоліком програми, адже така обмеженість по функціоналу може вимусити користувача використовувати декілька додатків.

Варто відмітити, що у програмі майже кожна дія має свою окрему сторінку, що дозволяє збільшити інформативність інтерфейсу та полегшує сприйняття даних користувачем.

Дах – це мобільний додаток, що поєднує у собі одразу багато функцій [7]. В нього входять контроль класності, можливість спілкуватися з сусідами, власне робота з платежами та можливість створювати заявки. Як можна побачити, не всі з можливостей відносяться до комунальних послуг, що може стати проблему для користувача, який не зацікавлений у всіх з них. Щоб уникнути цього, Дах використовує графічний інтерфейс, де кожна функція виділена у окреме меню та показана у вигляді зображення. Такий метод надання доступу до інформації є достатньо зручним до користувача, адже він дозволяє не лише легко отримати доступ до необхідної інформації, але й надає можливість зосередитись на необхідному функціоналі, не відволікаючись на зайві деталі.

Варто зауважити, що додаток функціонує доволі швидко у порівнянні з веб-застосунками. Так, наприклад, для завантаження інформації у ЦКС потрібно біля 5 секунд, хоча час може збільшуватись залежно від швидкості інтернету. У додатку Дах завантаження показників оплат чи лічильник проходить майже моментально, менше однієї секунди. Схожа ситуація і з іншими функціями програмного забезпечення. Це показує, що мобільні додатки є швидшими за веб-додатки, хоча і програють останнім в універсальності.

Ще одна проблема Даху, що відсутня у Коммуналочці – вимоги до авторизації. Користувач має надати доволі багато інформації та підтвердити своє місце проживання, у той час як у Коммуналочці створення об’єктів та робота з ними вимагає від користувача лише ввести адресу.

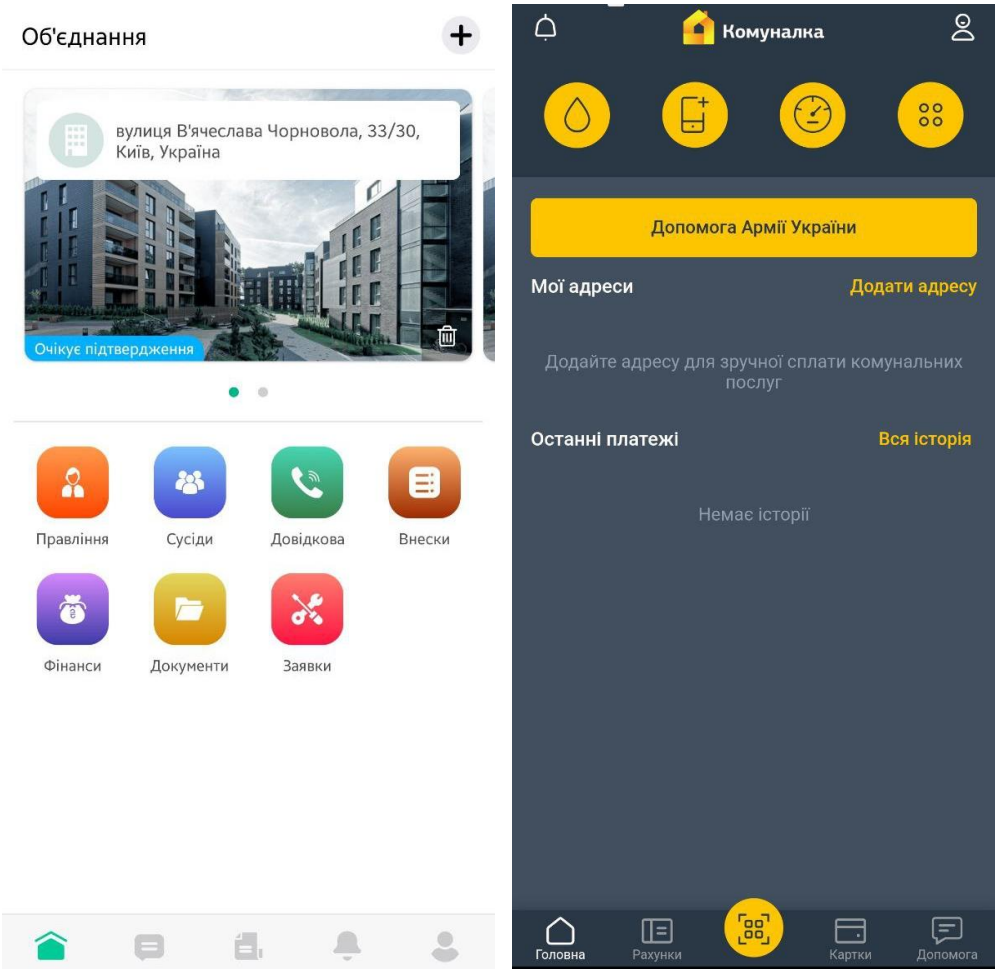


Рисунок 1.6 – Головне меню мобільного застосунків “Дах” та “Комуналка”

Комуналка

Останній додаток, що буде розглянуто – Комуналка. Це державний мобільний застосунок, що повністю орієнтовано на роботу з комунальними платежами, проведення дій спрямованих на облік та оплату платежів. Цей додаток можна назвати найбільш підходящим для ведення обліку, адже в ньому відсутні непотрібні для цієї мети послуги, присутній широкий функціонал для проведення дій спрямованих на облік комунальних платежів та присутній графічний інтерфейс, мінімалістичність якого спрямована на поліпшення ефективності користування додатком [8].

З недоліків можна виділити складний процес авторизації, відсутність пояснювальний підписів під елементами інтерфейсу та необхідність постійно переходити у нові меню для тих-чи інших дій.

Після аналізу вже існуючого програмного забезпечення можна створити таку таблицю, що показує загальні переваги та недоліки розглянутих програм. Можна дійти висновку, що не дивлячись на те, що загалом додатки пропонують схожий функціонал для проведення обліку комунальних платежів, додатки відрізняються за своїми інтерфейсами, робочими платформами та додатковими можливостями.

Аналіз недоліків та використання переваг інших додатків допоможуть у створенні більш зручного застосунку для проведення обліку комунальних платіжок. Створений додаток має бути кращим за вже існуючі аналоги у своїй сфері.

З переваг, що можна використати при створенні додатку орієнтованого саме на автоматизацію обліку комунальних платежів, можна виділити можливість автоматичного розрахування платіжок, швидке налаштування тарифів, детальну історію платежів а також можливість швидкого перемикання між об’єктами користувача.

Таблиця 1.1 – Порівняння аналогів програмного забезпечення для обліку комунальних платежів

Застосунок		Переваги	Недоліки
	ЦКС	- Великий набір даних - Автоматичний розрахунок платежів	- Відсутність можливості налаштування тарифів при розрахунку - Перевантаження інформацією
	Yasno	- Сконцентрованість на одному типу комунальних платежів - Простий та доступний інтерфейс	- Недостатньо детальна інформація по минулих платежах - Лише один об’єкт на користувача
	Коммуналочка	- Швідке налаштування тарифів та проведення розрахунків - Детальна історія платежів	- Заплутаний інтерфейс - Лімітована кількість об’єктів на користувача
	Дах	- Зручне перемикання між об’єктами - Можливість створення коментарів	- Перевантаження функціоналом, що не відноситься до роботи з комунальними послугами
	Комуналка	- Мінімалістичний інтерфейс - Можливість створення комплексних тарифів	- Інформація занадто децентралізована - Незрозумілість інтерфейсу

1.5 Інструменти побудови програмного забезпечення

1.5.1 Мова C#

C# - сучасна об'єктно-орієнтована мова програмування. C# дозволяє розробникам створювати різні типи безпечних та надійних програм, що виконуються в .NET [9]. Об'єктно-орієнтована мова програмування орієнтується саме на компоненти мова програмування. C# надає мовні конструкції для безпосередньої підтримки такої концепції роботи. Завдяки цьому C# підходить для створення та застосування програмних компонентів. .Особливості мови дозволяють обирати типи компонентів та їх поведінку [10].

Мова C# має свої особливості, що роблять її надзвичайно зручною для використання. Наприклад:

а) складання сміття автоматично звільняє пам'ять, зайняту недосяжними об'єктами, що не використовуються.

б) типи, що допускають значення null, забезпечують захист від змінних, які посилаються на виділені об'єкти.

в) обробка винятків надає структурований та розширюваний підхід до виявлення помилок та відновлення після них.

г) лямбда-вираження підтримують прийоми функціонального програмування.

ґ) синтаксис LINQ створює спільний шаблон для роботи з даними будь-якого джерела.

д) підтримка мов для асинхронних операцій забезпечує синтаксис для створення розподілених систем.

е) C# є Єдина система типів. Всі типи C#, включаючи типи-примітиви, такі як int та double, успадковують від одного кореневого типу object. Всі типи використовують загальний набір операцій, а значення будь-якого типу можна зберігати, передавати та обробляти таким чином. Більше того, C# підтримує як визначені користувачами типи посилань, так і типи значень.

э) C# дозволяє динамічно виділяти об'єкти та зберігати спрощені структури у стеку.

ж) C# підтримує універсальні методи та типи, що забезпечують підвищену безпеку типів та продуктивність.

з) C# надає ітератори, які дозволяють розробникам класів колекцій визначати варіанти поведінки для клієнтського коду.

1.5.2 Windows Forms

Windows Forms – це інтерфейс для створення додатків. Він відповідає в першу чергу за графічний інтерфейс користувача, дозволяючи розробнику значно пришвидшити та полегшити процес створення програмного забезпечення[11].

Цей інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за рахунок створення обгортки для існуючого Win32 API в керованому коді. Причому керований код - класи, що реалізують API для Windows Forms, не залежать від мови розробки. Тобто програміст однаково може використовувати Windows Forms як із написанні ПЗ на C#, З++, і VB.Net, J# та інших.

Використання Windows Forms дозволить поєднати успішні інтерфейсні рішення у єдине ціле, створюючи інтерфейс, що буде одночасно інформативним, зручним у використанні, простим для розуміння та швидким у створенні.

Windows Forms є подієво-орієнтованою програмою, що підтримується Microsoft .NET Framework. На відміну від пакетних програм, у Windows Forms більшість часу витрачається на очікування від користувача будь-яких дій, як, наприклад, введення тексту в текстове поле або кліка мишкою по кнопці.

1.5.3 База даних

Додаток, головна мета якого зводиться до обліку будь-якої інформації потребує місця , де ця інформація буде зберігатися. Для виконання цієї мети буде використана реляційна база даних – база даних з чіткою структурою.

По своїй суті реляційна база даних – це набір даних із зумовленими зв'язками з-поміж них. Ці дані організовані як набору таблиць, що , у свою чергу, складаються з стовпців і рядків. У таблицях зберігається інформація про об'єкти, які є у базі даних. У кожному стовпці таблиці зберігається певний тип даних, у кожному осередку – значення атрибута. Кожна строка таблиці є набором пов'язаних значень, які стосуються одного об'єкта чи сутності. Кожен рядок у таблиці може бути позначений унікальним ідентифікатором, що називається первинним ключем, а рядки з двох чи більше таблиць можуть бути пов'язані за допомогою зовнішніх ключів. До цих даних можна отримати доступ багатьма способами і при цьому реорганізовувати таблиці БД не потрібно [12].

Так як потреба у складній базі даних відсутня, у розробці програмного забезпечення буде використано MySQL. Така база даних є чудовим рішенням для додатків малого та середнього розмірів.

Для керування базою даних буде використано MAMP – застосунок для керування базами даних. Завдяки використанню цього програмного забезпечення, буде отримана можливість створити необхідну базу даних, провести налаштування таблиць, полів, значень. MAMP також надасть можливість перевіряти результати роботи розробленого програмного забезпечення під час розробки та тестування додатку.

1.6 Постановка завдань дослідження

Під час розробки концепції системи було визначено, що перш ніж проектувати систему, та вибирати конкретні інструменти та засоби реалізації, потрібно дослідити сферу автоматизації обліку комунальних платежів та вивчити системи, які мають подібний функціонал, до нашої майбутньої. Така підготовка є надзвичайно важливою, адже якісне планування, виконане заздалегідь, дозволить значно підвищити якість кінцевого програмного забезпечення, а також зменшить час самої розробки.

Мета роботи – підвищення зручності обліку сплати комунальних послуг громадян за рахунок автоматизації процесу із використанням програмного забезпечення.

Об’єкт дослідження – процес обліку сплати комунальних послуг громадян.

Предмет дослідження – програмне забезпечення для автоматизації обліку сплати комунальних послуг громадян.

Для ефективності побудови додатку, були поставлені завдання на дослідження певних технологій та інструментів, що дозволять реалізувати розроблену концепцію, а саме:

а) Дослідити інструменти побудови десктопних додатків за допомогою мови програмування C#;

б) Дослідити інтерфейс створення додатків Windows Forms;

в) Дослідити структуру даних майбутньої системи, та підібрати БД, яка задовольнятиме архітектуру та вимоги до системи.

г) Дослідити переваги та недоліки існуючого програмного забезпечення для ведення обліку комунальних платежів

ґ) Дослідити вимоги створення зручного та ефективного інтерфейсу для ведення обліку комунальних платежів.

Додаток, створений у результаті цієї роботи, має допомогти користувачам:

а) Збільшити зручність роботи з комунальними платежами, а саме внесення та облік комунальних платежів.

б) Підвищити швидкість, з якою користувач може працювати з платежами та обробляти інформацію, що стосується них.

в) Знизити кількість дій, що користувач має виконувати під час роботи з комунальними платіжками, і, як наслідок, зменшити кількість можливих помилок під час роботи.

Додаток, що може виконувати наведені вище функції стане достатньо корисним для користувача і стане центром, з якого користувач зможе керувати своїми житловими об’єктами без необхідності ведення паперової звітності.

2. РОЗРОБКА СТРУКТУРИ ДОДАТКУ ДЛЯ АВТОМАТИЗАЦІЇ ОБЛІКУ КОМУНАЛЬНИХ ПЛАТЕЖІВ ЗАСОБАМИ МОВИ C#

2.1. Моделювання об’єкту проектування

Будь-яке програмне забезпечення створюється для виконання певної мети чи задачі. Навіть найпростіші програми мають перед собою якесь завдання які вони мають виконувати, що вже говорити про більш складні та комплексні додатки, що можуть поєднувати у собі одночасно декілька завдань.

Від того, чи може застосунок виконувати поставлену заздалегіть мету, залежить успіх чи провал проекту, адже при розробці програмного забезпечення головним є досягнутий під кінець результат. Навіть найкращий та найзручніший додаток буде вважатися провалом, якщо він не здатний виконувати поставлену розробником чи, частіше, замовником мету.

Як можна побачити, знати завдання програмного забезпечення – надзвичайно важливо. Для того, щоб не збиватися зі шляху, надзвичайно корисним буде використовувати різноманітні схеми та діаграми, що легко візуалізують необхідну для розробника інформацію. Існує безліч діаграм, що можуть справлятися з цією метою, проте навіть використання лише найголовніших може значно полегшити процес розробки програмного забезпечення.

2.1.1. Діаграма варіантів використання

Одна з фундаментальних діаграм, що використовується під час розробки програмного забезпечення – це діаграма Use-case-ів, також відома як діаграма прецедентів. Використання такої діаграми проводиться повсемістно у всьому світі – та не лише при розробці програмного забезпечення.

Головною ідеєю цієї діаграми є графічне зображення можливих дій, що користувач може виконувати при роботі з програмним забезпеченням. Варто зауважити, що така діаграмам включає у себе усіх можливих користувачів, а не

лише користувача у звичайному розумінні цього слова. Так, наприклад, якщо в певній програмі є системний адміністратор, він також буде рахуватися користувачем.

Діаграма прецедентів, що відноситься до створеного програмного забезпечення включає у себе не тільки звичайні дії, а і можливі розширення, які активуються за певних умов.

При побудування діаграми варіантів використання необхідно дотримуватися певного порядку дій. Не дивлячись на те, що це порядок може відрізнятися у залежності від умов та ситуації, зазвичай він виглядає так:

а) аналіз створюваного продукту;

б) визначення дійовий осіб (акторів);

в) визначення варіантів використання;

г) складання опису до варіантів використання;

д) опис моделі прецедентів в цілому.

Так як аналіз створюваного продукту уже був проведений, цей крок уже виконаний, а отже варто перейти до визначення акторів. У конкретно цьому випадку, усі користувачі мають однаковий доступ до програмного забезпечення, а отже їх можна показати на діаграмі як одного актора.

Визначення варіантів використання проводиться відповідно до вимог, що поставлено при розробці програмного забезпечення. Загалом воно має виконувати наступні функції:

а) внесення показників лічильників та сум сплат

б) автоматичне розрахування сум сплати

в) виведення таблиці з минули показниками чи сумами сплат

г)вказування тарифів

ґ) надання посилань на сайти оплат

д) окремі об’єкти для кожного користувача

е) робота з декількома об’єктами у одному профілі

Усі функції описують самі себе, за винятком окремих об’єктів для кожного користувача. Цього можна досягнути через створення меню авторизації. Таким кожен користувач отримає необхідну приватність при роботі з програмним забезпеченням.

Також варто зауважити, що при виконання функції вказування тарифів, користувач має можливість використовувати базові тарифи, що також варто вказати на діаграмі.

Автоматичне розрахування сум сплати є розширенням до внесення сум сплати, проте є незалежним від самої дії, а отже має бути вказано за допомогою зв’язку extend.

Функції роботи з окремими об’єктами та можливість працювати з декількома об’єктами у одному профілі також вимагають можливість створення нових профілів користувача та об’єктів для кожного облікового запису окремо. Це можна показати через зв’язок include.

Опис необхідної діаграми варіантів використання завершено, а отже можна перейти до створення самої діаграми. У цій роботі для створення діаграм було використано онлайн сервіс LucidChart [13].

Як можна побачити з діаграми, більшість дій, що користувач може виконати у програмному забезпеченні не залежать одна від одної. Орієнтуючись на проаналізовані раніше аналоги, можна сказати, що у такому випадку найбільшо оптимальним рішенням буде розділити ци дії у окремі підменю, що будуть пов’язані одне з одним через центральне меню, наприклад головний екран.

Такий розподіл допоможе сконцентрувати увагу користувача на поточній потрібній йому функції, при цьому не створюючи занадто складної системи, що буде незручною для використання.

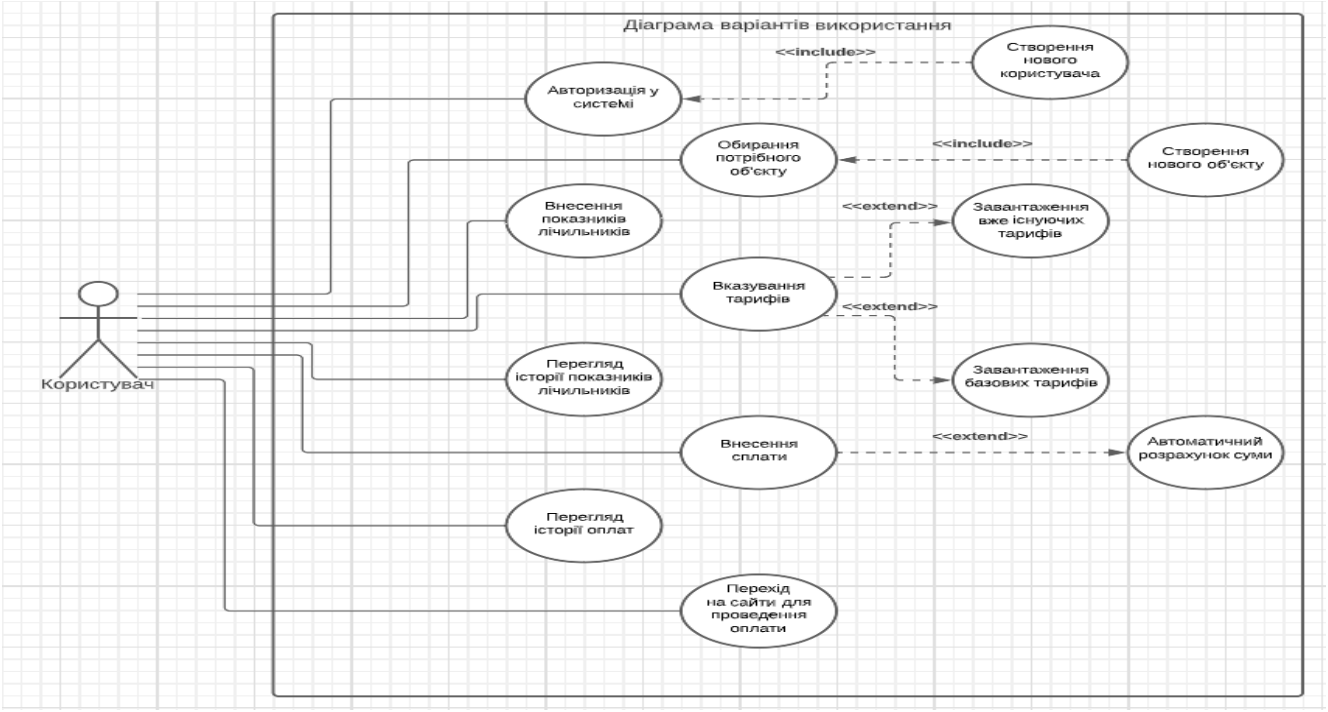


Рисунок 2.1 – Діаграма варіантів використання

Пов’язані між собою елемнти варто розташувати у одному підменю для підвищення зручності користування додатком.

2.1.2. Діаграма класів

Діаграма класів визначає типи класів системи та різного роду статичні зв'язки, які існують між ними. На діаграмах класів зображуються також атрибути класів, операції класів та обмеження, що накладаються на зв'язки між класами. Вигляд та інтерпретація діаграми класів істотно залежить від точки зору (рівня абстракції): класи можуть представляти сутності предметної галузі (у процесі аналізу) або елементи програмної системи (у процесах проектування та реалізації).

Основними елементами є класи та зв'язки між ними. Класи характеризуються за допомогою атрибутів та операцій.

Атрибути описують характеристики об'єктів класу. Більшість об'єктів у класі отримують свою індивідуальність через відмінності в їх атрибутах та взаємозв'язку з іншими об'єктами. Однак, можливі об'єкти з ідентичними значеннями атрибутів та взаємозв'язків. Тобто. індивідуальність об'єктів

визначається самим фактом їх існування, а чи не відмінностями у тому властивостях. Ім'я атрибута має бути унікальним у межах класу. Назва атрибута може містити його тип і значення за промовчанням.

Зв’язки між класами вказують на відношення між класами а також можливість перезоду від одного класу до іншого. Загалом можна виокремити такі типи зв’язків:

а) асоціація;

б) успадкування;

в) агрегація.

Використання діаграми класів дозволить полегшення розуміння структури створюваного програмного забезпечення, полегшить його розробку та, у подальшому, тестування та використання. Варто зауважити, що через пов’язаність з базами даних, що не вказується на такій діаграмі, у цьому конкретному випадку можна очікувати низьку кількість атрибутів у кожному окремому класі розробленого програмного забезпечення.

Створена діаграма класів демонструє зв’язки між класами програмного забезпечення. У той час як більшість класів є незалежними один від одного, обидва класа історій залежать від відповідного класу передачу інформації.

Клас реєстрації повністю залежить від класу логіну у систему і посзначений типом зв’язку “Успадкування”.

Такі класи як LinksTab та CalendarForm не мають у собі ніяких атрибутів, адже є виключно інформативними і не надають користувачу можливість змінювати себе у тому чи іншому вигляді.

Отримана у результаті програма значно спростила роботу з програмним забезпеченням завдяки візуалізації, а також покращила розуміння структури додатку.

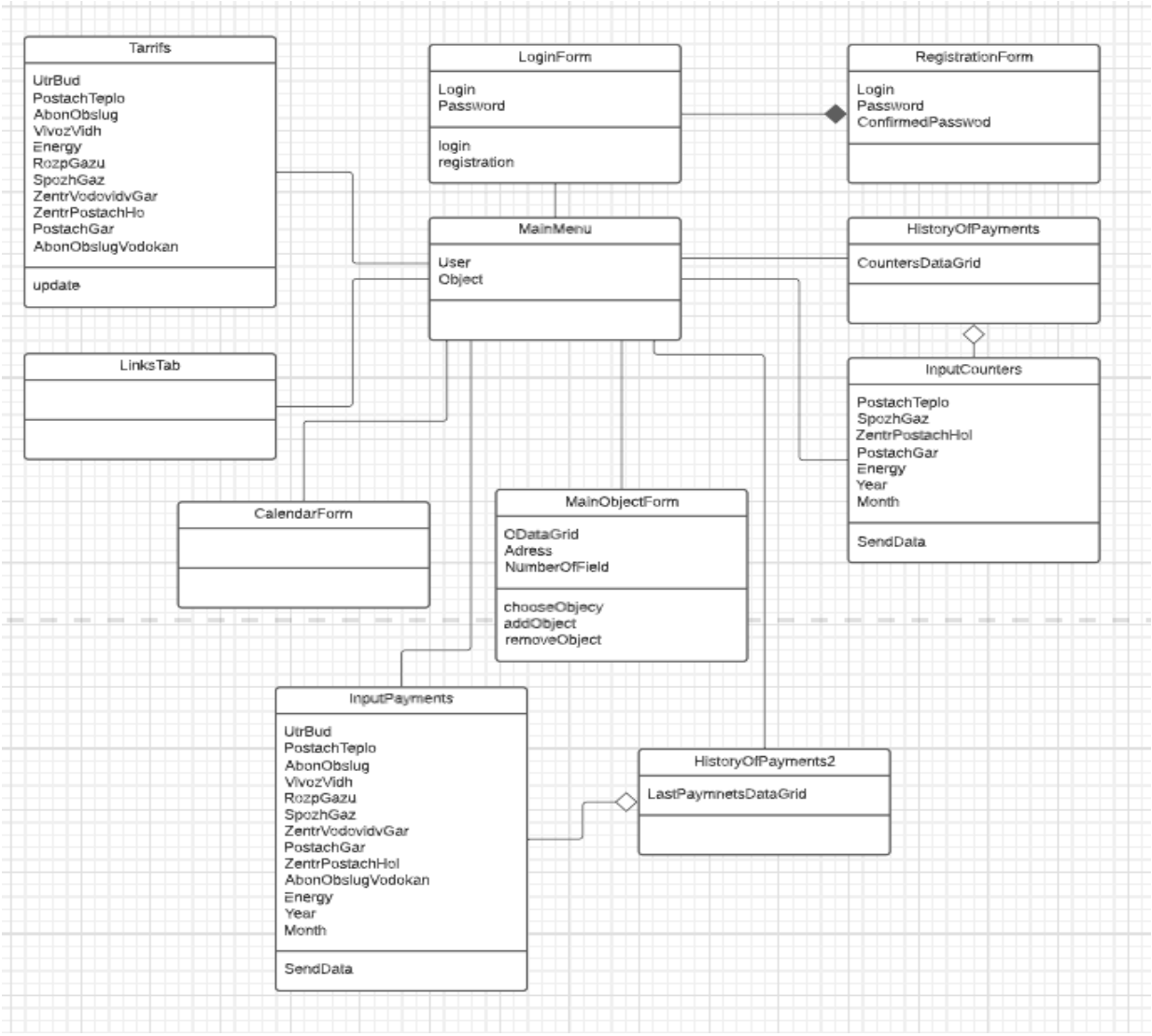


Рисунок 2.2 – Діаграма класів

2.1.3. ER-діаграма бази даних

Через велику кількість даних, що проходить через програму, використання бази даних є обов’язковим для створення додатку. Проте перед розробкою бази даних потрібно визначитися з її структурою. Для виконання цього завдання буде побудована ER-діаграма, також відома для користувачів під назвою модель сутність-зв’язок.

Діаграми Entity Relations - це блок-схеми, які ілюструють, як «сутності» (люди, об'єкти або концепції) ставляться один до одного в системі. ER-діаграма - це та модель, яка найчастіше використовуються для розробки або налагодження

реляційних баз даних в областях ПО, бізнес-інформаційних систем і досліджень. Вона використовує набір геометричних символів, таких як прямокутник, ромб, овал і лінії, для відображення взаємозв'язку об'єктів, відносин і їх атрибутів [14].

ER-діаграми складаються з сутностей, відносин і атрибутів. Вони також відображають потужність, яка визначає відносини в термінах чисел. Сутність - визначається об'єкт, такий як людина, концепція або подія. Може містити дані, що зберігаються в ньому. Сутності діляться на сильні, слабкі або асоціативні. Сильний об'єкт визначається тільки за своїми ознаками, а слабка сутність цього не може. Асоціативний вид пов'язує об'єкти або елементи. Ключі сутностей вказують на атрибут, який визначає об'єкт в наборі.

ER діаграми надають дуже корисну основу для створення і управління базами даних. По-перше, діаграму ER легко зрозуміти. Це означає, що, наприклад, дизайнери можуть використовувати діаграми ER для простого спілкування з розробниками, клієнтами та кінцевими користувачами, незалежно від їх професіоналізму в області IT. По-друге вони легко переводяться в реляційні таблиці, які можна використовувати для швидкої збірки баз даних. Крім того, діаграми ER можуть безпосередньо використовуватися розробниками БД як план для впровадження даних в конкретні програмні додатки.

Таким чином ER-діаграма є чудовими варіантом для опису бази даних і буде побудована для покращення розуміння структури створюваного програмного забезпечення.

Створена діаграма демонструє структуру базу даних, показуюючи її таблиці, поля, коди, а також зв’язки. Це надає можливість краще зрозуміти принцип роботи бази даних, а також те, як передані показники зберігаються всередині.

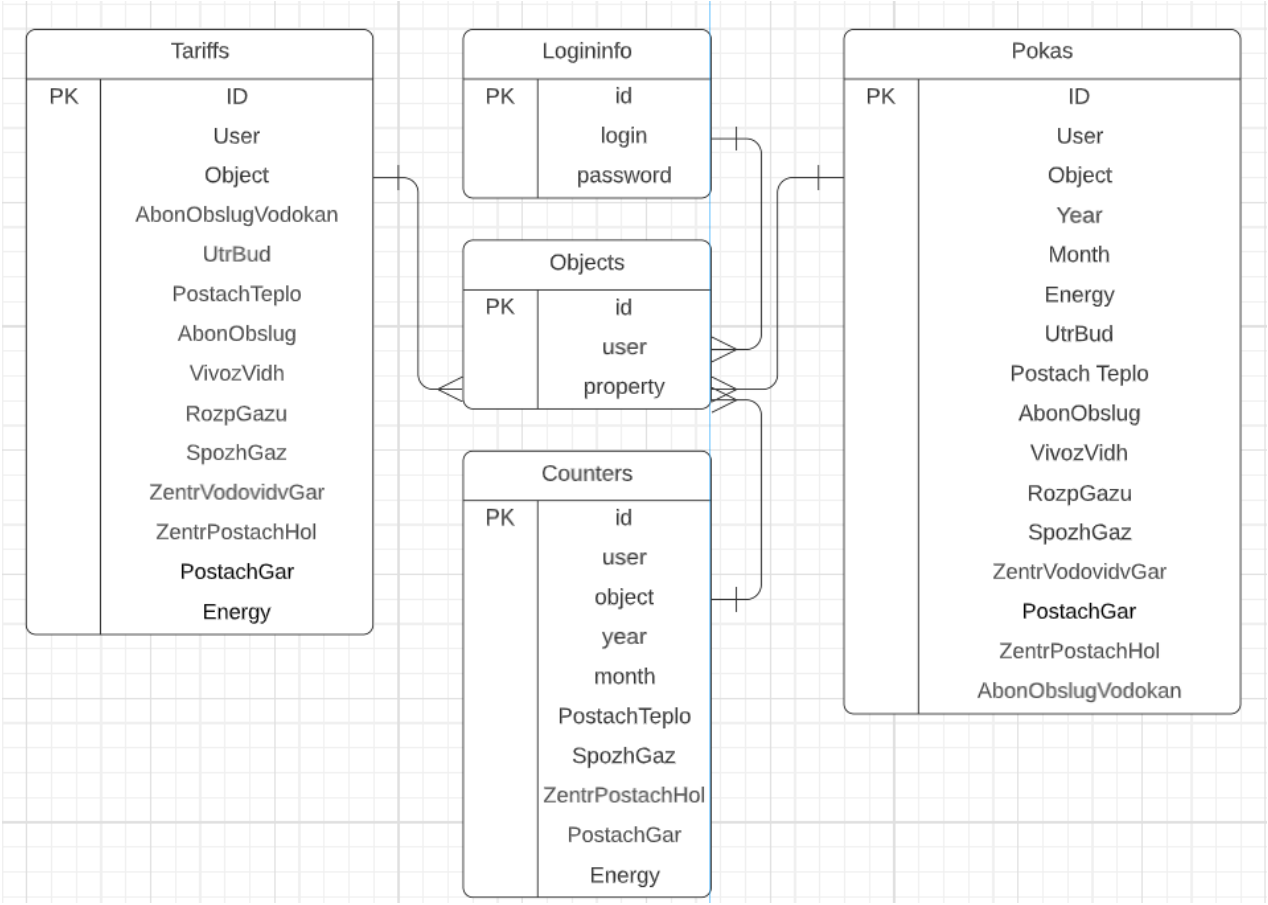


Рисунок 2.3 – ER-діаграма бази даних

З діаграми видно, що таблиці Tarrifs, Counters так Pokas зберігають більшу частину даних у собі. Вони зберігають інформацію:

а) Tarrifs про тарифи;

б) Counters про передані користувачем показники лічильників;

в) Pokas про суми оплат.

Інформація зберігається для кожного об’єкту окремо і при обиранні даних для відображення у додатку фільтрується по полю object, що має відповідати поточному значенню property користувача. Поля Year та Month допомагають фільтрувати дані по даті.

2.2. Використане програмне забезпечення

При розробці додатку було використано різноманітне програмне забезпечення як для написання коду, так і для створення елементів графічного інтерфейсу.

Для написання коду була застосована IDE Visual Studio, програмне забезпечення створене компанією Microsoft. Visual Studio містить у соба інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Це програмне забезпечення включає у себе редактор вихідного коду та має можливість рефакторінгу. Він дає змогу розробляти як консольні програми, так і програми з графічним інтерфейсом.

IDE Visual Studio чудово пристосована до написання додатків на мові програмування C#, а отже є логічним вибором. Використання Visual Studio також надає можливість використання Windows Forms, що є важливою частиною створюваного продукту.

Для створення графічних елементів, що у подальшому стануть частиною інтерфейсу розроблюємого додатку, буде використано графічний редактор Photoshop, продукт компанії Adobe. Цей додаток має широкий спектр застосування і використовується для створення зображень, веб-дизайну, поліграфі і т.д. Завдяки широким вбудованим можливостям цього редактору, якість графічних елементів інтерфейсу буде поліпшено, а процес їх створення – пришвидшено. Не дивлячись на те, що основний формат файлів Photoshop – PSD, цей додаток може бути використано і для створення зображень у форматі, що підтримується при створенні додатків, наприклад PNG чи JPG. Зображення будуть створені у форматі PNG, адже використання JPG не має сенсу через невеликий розмір проекту.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Планування розробки проекту

Після формування вимог та завершення планування, варто приступати до програмної реалізації застосунку. Для зручності у роботі було створено Excel документ, в якому був розташований check-list зі списком мінімальних вимог необхідних для того, щоб програмне забезпечення відповідало потребам користувача. Такий підхід називається MVP (Minimum Viable Product), і він дозволяє отримати робочу версію програмного забезпечення у мінімальний проміжок часу.

Вимога	Статус
Авторизація/реєстрація	Імплементовано
Робота з декількома об'єктами	Імплементовано
Внесення показників лічильників	Імплементовано
Перегляд минулих показників лічильників	Імплементовано
Автоматичне установлення базового тарифу	Імплементовано
Редагування тарифів	Імплементовано
Внесення сум оплати	Імплементовано
Автоматичне розрахування місячної суми	Імплементовано
Перегляд минулих оплат	Імплементовано
Посилання на сервіси оплат	Імплементовано

Рисунок 3.1 – MVP вимоги до продукту

За словами Еріка Ріса, автора підходу, MVP — це версія нового продукту, яка дозволяє команді зібрати максимальну кількість перевірених знань клієнтів із найменшими зусиллями. Деякі визначають MVP як «першу версію продукту», інші як «урізану версію продукту», а дехто і взагалі заперечують ідею MVP і розробляють «повномасштабний, але простий продукт» [15].

Така різниця у розумінні відбувається через збільшення очікувань клієнтів, що ґрунтуються на поширенні та адаптації складних технологій та технологічних продуктів.

Така ситуація призвела до створення модифікованих понять MVP. Найбільш відомими з них є:

- Minimum Lovable Product , що передбачає створення достатньої функціональності, яка надасть клієнтам можливість обожнювати продукт відразу після запуску, а не просто терпіти його.
- Minimum Marketable Product, що орієнтується на створення продукту з мінімальною кількістю реалізованих функцій, що дозволить протестувати ринок для подальшої розробки.

Кожен з цих методів має свої переваги та недоліки, проте у цій роботі буде використовуватись саме базовий метод (MVP), адже він є найбільш гнучким і повністю відповідає моєму баченню продукту.

Після виконання вимог встановлених MVP були реалізовані й інші, менш значущі вимоги, без яких продукт міг би функціонувати, але які підвищують потенціальне задоволення користувача від використання розробленого програмного забезпечення.

Вимога	Статус
Графічний інтерфейс кнопок	Імплементовано
Календар подій	Імплементовано
Вивід інформації про користувача на головне меню	Частково
Можливість переходу між пов'язаними меню	Імплементовано

Рисунок 3.2 – Таблиця додаткових вимог

Через обмеження у часі, деякі з додаткових вимог не були імплементовані, або були імплементовані лише частково. Проте, так як усі головні вимоги до продукту вже були реалізовані, це не є проблемою для функціонування програмного забезпечення.

Створення бази даних проводилось разом з розробкою основного застосунку, адже через пов’язаність бази даних та додатку це дозволяло оперативно реагувати на проблеми, які виникали чи, потенціально, могли виникнути у процесі розробки.

База даних мала свій власний список вимог, які обов’язково мали бути реалізовані у ній. Через низьку кількість вимог та важливість кожної з них, усі вони отримали статус важливих для функціонування застосунку.

Вимога	Статус
База користувачів	Імплементовано
База об'єктів	Імплементовано
Зберігання показників лічильників	Імплементовано
Зберігання минулих оплат	Імплементовано
Базовий тариф та користувацькі тарифи	Імплементовано

Рисунок 3.3 – Таблиця вимог до бази даних

При такій низькій кількості вимог було прийнято рішення створити окремі таблиці для кожної з них, які у подальшому будуть використовуватися за потреби. Таке рішення значно спростить роботу з базою даних напряму, якщо така потреба виникне.

3.2 Методологія розробки

Обирання методології розробки є надзвичайно важливим кроком для створення програмного забезпечення. Вибір робиться між стандартними моделями розробки, такими як каскадна, водоспадна, спіральна та ітеративна моделі та гнучкими моделями. У цій роботі буде надано перевагу саме останній методології.

3.2.1 Agile

Гнучка методологія Agile - це спосіб керувати проектом, розбиваючи його на кілька етапів. Це передбачає постійну співпрацю із зацікавленими сторонами та

постійне вдосконалення на кожному етапі. Як тільки робота починається, команди проходять цикл через процес планування, виконання та оцінки. Постійна співпраця є життєво важливою як з членами команди, так і з зацікавленими сторонами проекту.

Гнучкі методології проекту є доволі популярними у наш час, і це не просто так. Завдяки можливості підлаштовуватись під вимоги, що стоять перед проектом, така методологія розробки дозволяє виконати проект у найбільш ефективний спосіб.

3.2.2 Kanban

Не дивлячись на те, що гнучкі методології, загалом, будуються на одних і тих же принципах, Agile є лише узагальненим поняттям для таких методологій, а самі вони розділяються на підвиди.

Так як програмне забезпечення розробляється невеликою командою, було вирішено звернути уваги на методологію Kanban – один із підтипів Agile. Kanban використовує візуальний підхід до управління проектами, коли команди створюють фізичні репрезентації своїх завдань, часто використовуючи наліпки на дошках (або онлайн-додатках). Завдання переміщуються через заздалегідь визначені етапи, щоб відстежувати прогрес і виявляти загальні перешкоди. Такий підхід дозволяє візуально оглядати прогрес у розробці додатку, чітко бачити перешкоди та тимчасово залишати завдання у незавершеному стані, щоб вирішити проблему, яка постала перед цією задачею.

Використання Kanban неможливе без Kanban-дошки, що є надзвичайно важливим елементом цієї методології розробки. Дошка Kanban - це інструмент управління Agile-проектами, який допомагає наочно уявити завдання, обмежити обсяг незавершеної роботи та досягти максимальної ефективності (або швидкості). Вона може допомогти командам Agile та DevOps упорядкувати повсякденну роботу. За допомогою карток та стовпців на дошці Kanban команди з технічних питань та сервісні команди можуть зрозуміти, який обсяг роботи слід взяти на себе,

та виконати цей обсяг, дотримуючись принципів безперервного вдосконалення [16].

Девід Андерсон, чоловік що визначив принципи цієї методології, виділяє п'ять складових дошок Kanban: видимі сигнали, стовпці, ліміти незавершеної роботи, точка прийняття зобов'язань та точка постачання продукту. Розгляд цих складових дає зрозуміти, чому ця методологія така ефективна.

а) Видимі сигнали. Першими на дошці Kanban впадають у вічі картки (стікери, листки та ін.). Kanban-команди виносять записи про всі проекти та робочі завдання на картки; одна картка, як правило, відповідає одному проекту або робочому завданню. Для Agile-команд кожна картка позначає одну користувальницьку історію. Побачивши ці сигнали на дошці, учасники команди та зацікавлені сторони зможуть легко зрозуміти, над чим працює команда.

б) Стовпці. Ще однією відмітною ознакою дошки Kanban є стовпці. Вони символізують конкретні дії, що у сукупності становлять «робочий процес». Картки переміщуються робочим процесом до стадії завершення. Робочі процеси можуть бути простими і складатися лише зі стовпців «Має бути», «У процесі» та «Завершено», а можуть бути набагато складнішими.

в) Обмеження незавершеної роботи (WIP). Обмеження WIP – це максимальна кількість карток, яка може бути в одному стовпці одночасно. Якщо для стовпця вибрано обмеження WIP, що дорівнює 3, то в ньому не може бути більше трьох карток. Коли кількість карток у стовпці досягає максимуму, команда має зосередити зусилля цих картках і передати їх далі, щоб у цю стадію робочого процесу могли надійти нові картки. Обмеження WIP потрібні, щоб виявляти проблемні місця в робочому процесі та досягати максимальної швидкості роботи. Обмеження WIP допомагають на ранніх етапах зрозуміти, чи команда взяла на себе занадто багато завдань.

г) Точка прийняття зобов'язань. На дошці у Kanban-команд часто є беклог. Клієнти та учасники команди вносять у нього ідеї щодо проектів, до яких команда

може звернутися, коли буде готова. У точці прийняття зобов'язань команда вибирає ту чи іншу ідею, після чого розпочинається робота над проектом.

ґ) Точка постачання продукту. Точка постачання продукту означає завершення робочого процесу команди Kanban. Багато команд приймають за точку постачання товару момент, коли продукт або сервіс передаються у розпорядження клієнта. Мета команди - якнайшвидше перенести картки з точки прийняття зобов'язань до точки поставки продукту. Час, протягом якого картка проходить з однієї точки в іншу, називається часом виконання. Kanban-команди постійно удосконалюються, прагнучи звести час виконання до мінімуму.

Дотримання цих принципів у роботі з методологією є надзвичайно важливою вимогою успіху. Методологія Kanban має свої недоліки яких варто уникати. Головний з них – можливість виникнення великої кількості незавершених завдань при порушенні третього пункту (WIP). Це може призвести до втрати контролю над проектом та деорганізації, команді знадобиться багато часу на виправлення цієї проблеми.

Для створення Kanban-дошки буде використано онлайн сервіс WorkSection. Завдяки безкоштовному пробному періоду, цей сервіс дозволяє отримати повний доступ до можливостей Kanban без необхідності в оплаті. Двох тижнів цілком вистачить для завершення проекту.

Створення дошок у цьому сервісі просте на інтуїтивно зрозуміле. Завдяки можливості назначення пріоритетів та сортуванню задач по них, можна легко слідкувати за процесом розробки проекту.

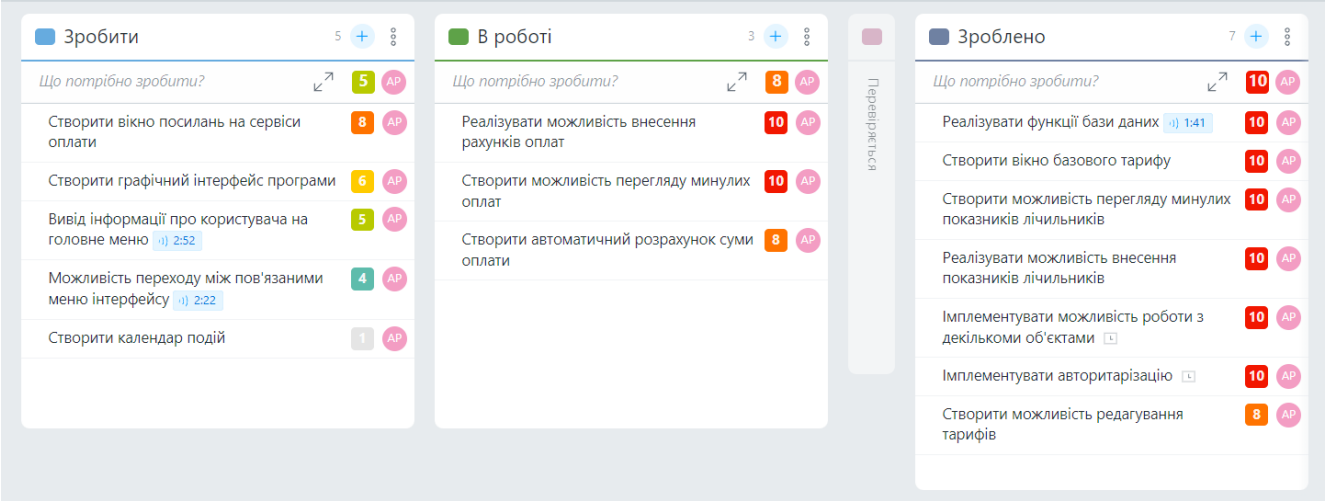


Рисунок 3.4 – Kanban-дошка проекту

Сервіс WorkSection також надає можливість передивлятися статистику проекту, для кращого розуміння поточної ситуації. Така можливість є надзвичайно зручною у довгих проектах для визначення орієнтовного часу, необхідного для виконання тої чи іншої задачі поставленої перед працівниками компанії.

Завдяки усім мірам, що були прийняті для контролю над ходом проекту, можна очікувати плавний процес розробки, що завершиться успіхом.

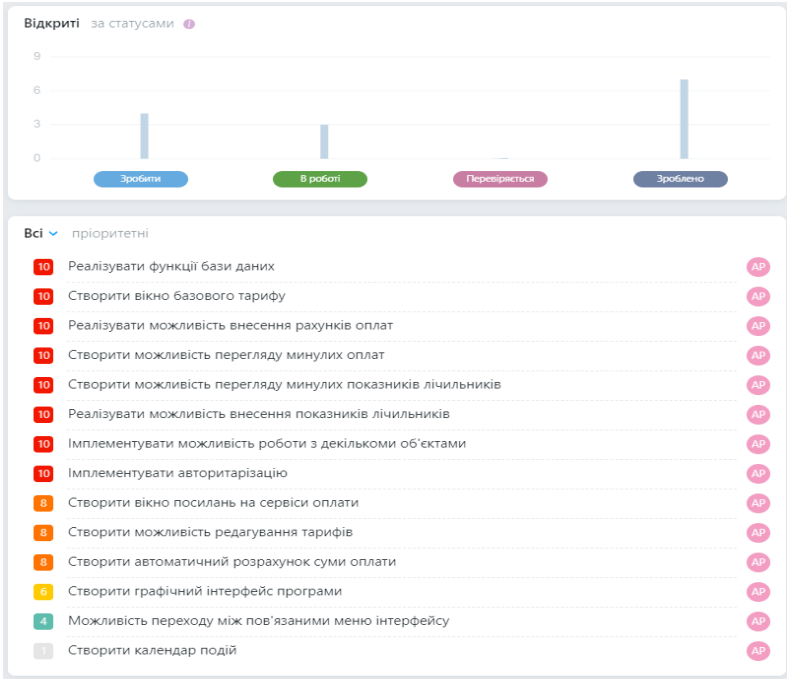


Рисунок 3.5 – статистика проекту

3.3 Опис функціонування додатку

Розроблений додаток здатний виконувати завдання по обліку комунальних послуг та даних, що відносяться до них. У цьому розділі буде розглянуто декілька головних функцій програмного забезпечення, що найбільш вірогідно будуть використані користувачами. Вони включають в себе методи, з яких складається вся програма, а оте розглянувши їх можна отримати гарне розуміння додатку у цілому.

Для покращення розуміння, кожна функція буде супроводжена блок-схемою, що буде показувати алгоритм її діяльності. Поєднання коду та блок схем має значно покращити сприйняття інформації.

3.3.1 Внесення показань лічильників

Це одна з ключових функцій додатку. Не дивлячись на її простоту у використанні, вона включає в себе декілька кроків, які має виконати користувач для успішного виконання (рис.3.6).

Рисунок 3.6 – Блок-схема функції внесення показань лічильників

Перший з них – введення дати, по якій користувач планує передавати показники. В користувача є можливість обрати будь-який рік та місяць, не обмежуючи його лише поточною датою. У випадку, коли користувач обрав дату, по якій вже були передані показники, програма завантажить ці показники для редагування. Метод перевірки показників можна побачити на рисунку 3.7

```
DB db = new DB();
DataTable checktable = new DataTable();

MySqlDataAdapter checkadapter = new MySqlDataAdapter();

MySqlCommand cmdLogin = new MySqlCommand("SELECT * FROM `counters` WHERE `User` = @ul AND `Month` = @mn AND `Year` = @yr  AND `Object` LIKE @ob", db.checkConnection());
cmdLogin.Parameters.Add("@ul", MySqlDbType.VarChar).Value = DataBank.UserNameGlb;
cmdLogin.Parameters.Add("@ob", MySqlDbType.VarChar).Value = DataBank.ObjectGlb;
cmdLogin.Parameters.Add("@mn", MySqlDbType.VarChar).Value = MonthField.Text;
cmdLogin.Parameters.Add("@yr", MySqlDbType.VarChar).Value = YearField.Text;

checkadapter.SelectCommand = cmdLogin;
checkadapter.Fill(checktable);
```

Рисунок 3.7 – перевірка наявності показників

У випадку, коли показання не знайдено, користувач не помітить ніяких змін, поля залишаться у тому стані, в якому знаходились. Проте якщо дані по цій даті є, буде виконано код , що завантажить показники для редагування.

```
if (checktable.Rows.Count > 0)
{
    db.establishConnection();
    PostachTeplo.Text = checktable.Rows[0].Field<float>("PostachTeplo").ToString();
    SpozhGaz.Text = checktable.Rows[0].Field<float>("SpozhGaz").ToString();
    ZentrPostachHol.Text = checktable.Rows[0].Field<float>("ZentrPostachHol").ToString();
    PostachGar.Text = checktable.Rows[0].Field<float>("PostachGar").ToString();
    Energy.Text = checktable.Rows[0].Field<float>("Energy").ToString();
    db.cutConnection();
}
```

Рисунок 3.8 – перевірка наявності показників

Механізм цієї перевірки доволі простий – при зміні місяця, року, або відкритті цієї форми, програма намагається завантажити з бази даних у табличку показники по обраній даті. Якщо таблиця залишилась чи стала пустою – перевірка з рисунку 3.8 не виконується. Таким чином виконується майже непомітна для користувача робота, що спрямована на підвищення зручності його роботи.

При натисканні кнопки відправити виконується доволі простий запит до бази даних, що вносить показники у таблицю. Ця дія також є майже миттєвою та непомітною для користувача. На випадок виникнення помилки у цьому процесі,

існує додаткова умова. Якщо дані не вдалось передати (наприклад через відсутність зв'язку з базою даних) користувач отримає повідомлення про помилку.

```
DB db = new DB();
MySqlCommand command = new MySqlCommand("INSERT INTO `counters` (`User`,`Object`, `Year`,`Month`, `PostachTeplo`,`SpozhGaz`, `ZentrPostachHol`,`PostachGar`,

command.Parameters.Add("@aq", MySqlDbType.VarChar).Value = DataBank.UserNameGlb;
command.Parameters.Add("@ob", MySqlDbType.VarChar).Value = DataBank.ObjectGlb;
command.Parameters.Add("@aw", MySqlDbType.VarChar).Value = YearField.Text;
command.Parameters.Add("@ae", MySqlDbType.VarChar).Value = MonthField.Text;
command.Parameters.Add("@ar", MySqlDbType.VarChar).Value = PostachTeplo.Text;
command.Parameters.Add("@at", MySqlDbType.VarChar).Value = SpozhGaz.Text;
command.Parameters.Add("@ay", MySqlDbType.VarChar).Value = ZentrPostachHol.Text;
command.Parameters.Add("@au", MySqlDbType.VarChar).Value = PostachGar.Text;
command.Parameters.Add("@ai", MySqlDbType.VarChar).Value = Energy.Text;

db.establishConnection();

if (command.ExecuteNonQuery() == 1)
{
    MessageBox.Show("Data sent");
}
else
{
    MessageBox.Show("An error has ocurred during uploading of data");
}

db.cutConnection();
```

Рисунок 3.9 – Метод передачі показників до бази даних

Для підвищення безпеки конфіденційності користувача, під час передачі даних у базу даних та під час надсиланні запросу на перевірку, назви полів введено в якості "заглушок" – змінних, що не дають потенційному зловмиснику ніякої інформації дані що передаються. Хоча цей метод і не пропонує повного захисту даних, він зробить процес взлому бази даних набагато складнішим.

```
VALUES (@aq ,@ob ,@aw ,@ae ,@ar ,@at ,@ay,@au, @ai);", db.checkConnection());
```

Рисунок 3.10 – Значення у запиті замінено на "заглушки"

Також на рисунку 3.9 можна помітити, деякі зі значень беруться з класу DataBank. Цей клас включає в себе лише два значення – ім'я поточного користувача та обраний об'єкт. Ці значення зберігаються не в базі даних, а у самі програмі. Це зроблено для зменшення кількості необхідних запитів для бази даних та захисту цих полів від помилок, якщо зв'язок з базою даних перерветься.

Значення цих полів визначається у інших частинах програми. UserNameGlb встановлюється під час авторизації, а ObjectGlb – під час обирання об'єкту. Якщо

користувач ще не обрав об’єкт (або не користується можливістю обирання об’єктів взагалі) це поле може бути пустим.

3.3.2 Перегляд та редагування тарифів

У цьому додатку тарифи були зроблені доволі гнучкими. При першому заході у акаунт користувач не буде мати налаштованих тарифів – замість цього його тарифи будуть дорівнювати базовому значенню з бази даних. Це значення дорівнює найбільш вірогідним показникам і може буде змінено у базі даних, або через користувача “root”. Проте через те, що тарифи доволі часто відрізняються для різних об’єктів, користувач може вільно змінювати їх за своїми вподобаннями. При ввімкненні вікна тарифів (або проведенні обрахунків) додаток перевіряє, чи існують користувацькі тарифи для цього акаунту. Якщо так – саме їм надається пріорітет.

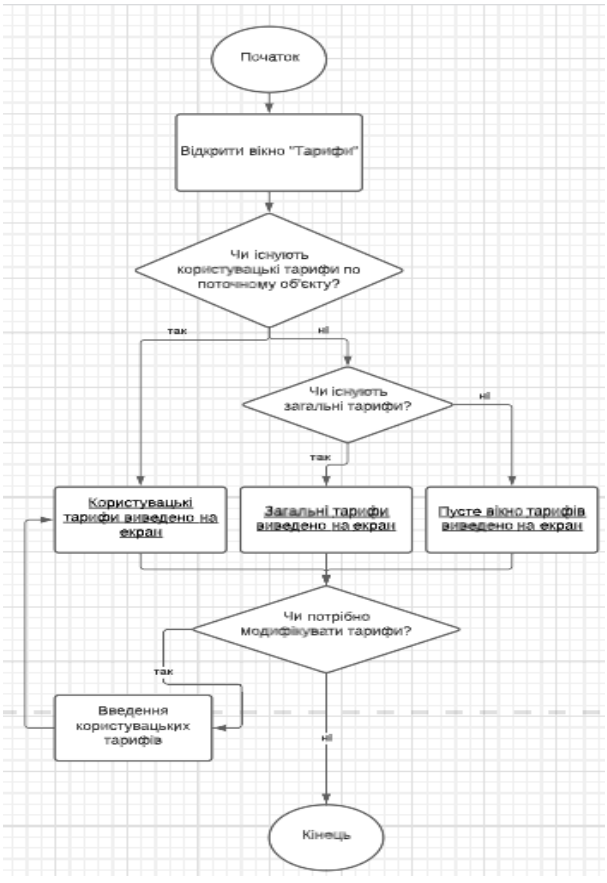


Рисунок 3.11 – Блок-схема функції перегляду та внесення тарифів

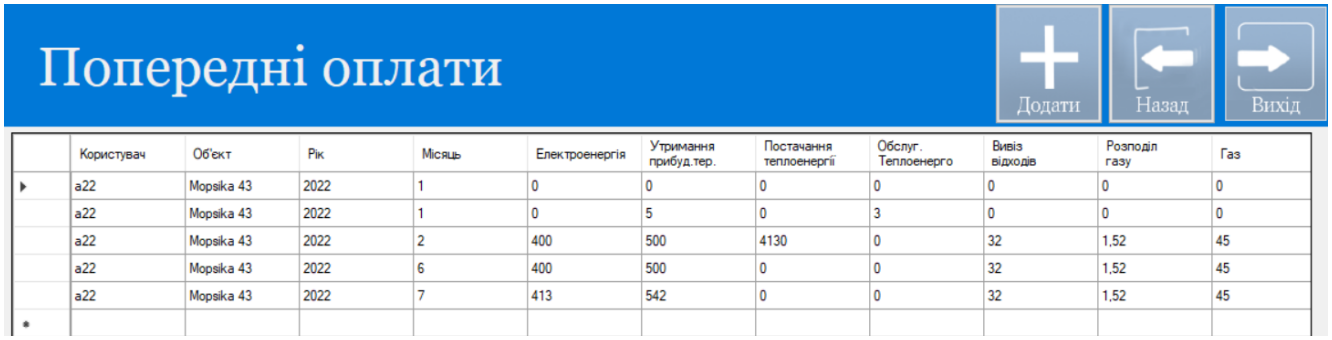
Загалом у роботі цієї функції використовуються ті ж самі методи, що і в минулій. Зміни є лише у самих даних, що передаються чи приймаються під час роботи програми.

ID	User	Object	AbonObslugVodokan	UtrBud	PostachTeplo	AbonObslug	VivozVidh	RozpGazu	SpozhGaz	ZentrVodovidvGar	ZentrPostachHol
1	root		39.19	559.09	1341.53	31.07	72.23	1.56	7.8	14.22	16.164
12	a22		1	1	1	1	1	1	1	1	1
13	a22	Mopsika 43	1	1	1	1	1	1	1	1	1
14	a22	Hops 3	13	23	23	32	32	13	42	52	35

Рисунок 3.12 – Тарифи у базі даних

На рисунку 3.12 можна побачити дані, що знаходяться у таблиці Tarrifs. Там знаходяться показники по замовчуванню, збережені у користувачі root, та дані для користувача a22 – його двух об'єктів та ситуації коли об’єкт не обрано.

3.3.3 Вивід даних у таблицю



	Користувач	Об'єкт	Рік	Місяць	Електроенергія	Утримання прибуд.тер.	Постачання теплоенергії	Обслуг. Теплоенерго	Вивіз відходів	Розподіл газу	Газ
▸	a22	Mopsika 43	2022	1	0	0	0	0	0	0	0
	a22	Mopsika 43	2022	1	0	5	0	3	0	0	0
	a22	Mopsika 43	2022	2	400	500	4130	0	32	1,52	45
	a22	Mopsika 43	2022	6	400	500	0	0	32	1,52	45
	a22	Mopsika 43	2022	7	413	542	0	0	32	1,52	45
*											

Рисунок 3.13 – Таблиця попередніх оплат

Внесені у базу даних як показники, так і оплати, можуть бути переглянуті у меню попередних оплат чи показників. У цій формі дані будуть показані у форматі таблиці. Корисутвач може сортувати дані натискаючи на назви стовпців.

Дані, презентовані у таблиці, набагато легше зрозуміти та обробити, ніж дані розкидані по платіжках користувача. Можливість сортування також є важливим елементом, спрямованим на підвищення зручності користування додатком.

Кожна таблиця має кнопку, що дозволяє перейти до форми додання нових показників і навпаки, кожна форма передачі показників має посилання на пов’язану з нею таблицю. Так як таблиця оплат має велику кількість значень для відображення, користувач має можливість пролистувати її завдяки слайд-бару внизу сторінки.

3.3.4 Передача сум оплат та автоматичне розрахування

Функція внесення оплат має як схожі характеристики з функцією внесення показників лічильників, так і додаткові елементи. Головний з низ – можливість автоматичного обрахування показників по обраній даті. Для цього потрібно виконати декілька умов – встановити тарифи, або мати тарифи за замовченням, та передати показники лічильників по бажаній даті.

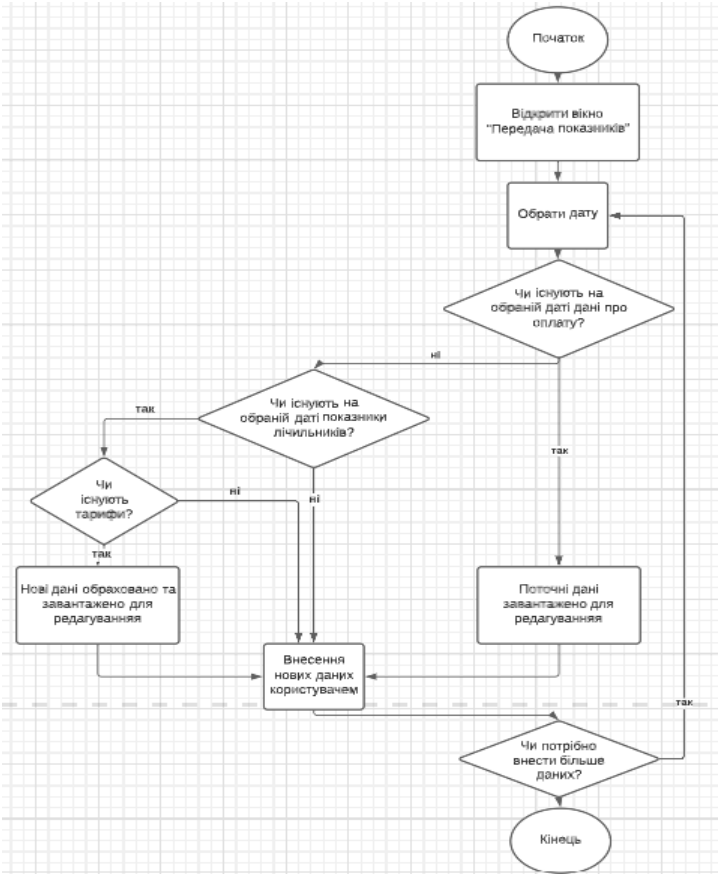


Рисунок 3.14 – блок-схема функції внесення оплат

Якщо умови виконано, при обиранні бажаної дати програма сама розрахує суму до сплати. Таким чином користувач зекономить свій час та зможе уникнути риску помилки. За бажанням користувач може редагувати ці дані перед дим як надсилати їх до бази даних.

Суми обраховуються за логікою зображеною на рисунку 3.15. Варто зауважити, що не обов'язково передавати усі значення – навіть наявність лише одного переданого показника лічильника дозволить провести розрахунок по тій позиції.

```
DataTable Ttable = new DataTable();
MySqlDataAdapter Tadapter = new MySqlDataAdapter();

MySqlCommand cmdT = new MySqlCommand("SELECT * FROM `tarrifs` WHERE `User` = @ul  AND `Object` LIKE @ob", db.checkConnection());
cmdT.Parameters.Add("@ul", MySqlDbType.VarChar).Value = DataBank.UserNameGlb;
cmdT.Parameters.Add("@ob", MySqlDbType.VarChar).Value = DataBank.ObjectGlb;

Tadapter.SelectCommand = cmdT;
Tadapter.Fill(Ttable);

db.establishConnection();
textBox1.Text = Ttable.Rows[0].Field<float>("UtrBud").ToString();
textBox2.Text = (Ctable.Rows[0].Field<float>("PostachTeplo") * Ttable.Rows[0].Field<float>("PostachTeplo")).ToString();
textBox3.Text = Ttable.Rows[0].Field<float>("AbonObslug").ToString();
textBox5.Text = Ttable.Rows[0].Field<float>("VivozVidh").ToString();
textBox6.Text = Ttable.Rows[0].Field<float>("RozpGazu").ToString();
textBox7.Text = (Ctable.Rows[0].Field<float>("SpozhGaz") * Ttable.Rows[0].Field<float>("SpozhGaz")).ToString();
textBox8.Text = (Ctable.Rows[0].Field<float>("ZentrVodovidvGar") * Ttable.Rows[0].Field<float>("PostachGar")).ToString();
textBox9.Text = (Ctable.Rows[0].Field<float>("PostachGar") * Ttable.Rows[0].Field<float>("PostachGar")).ToString();
textBox10.Text = (Ctable.Rows[0].Field<float>("ZentrPostachHol") * Ttable.Rows[0].Field<float>("ZentrPostachHol")).ToString();
PostachTeplo.Text = Ttable.Rows[0].Field<float>("AbonObslugVodokan").ToString();
textBox11.Text = (Ctable.Rows[0].Field<float>("Energy") * Ttable.Rows[0].Field<float>("Energy")).ToString();
db.cutConnection();
```

Рисунок 3.15 – блок-схема функції внесення оплат

Поля, що не потребують переданих показників лічильників будуть підставлені у відповідні TextBox-и навіть якщо програма не знайде передачі показань по цій даті.

Пріоритет автоматичного розрахування чи підгрузки вже існуючих показників для редагування можна встановити власноруч.

Варто зазначити, що як на цій формі, так і на усіх інших формах, у полях де очікуються цифри – можна ввести лише цифри та крапку. У зв'язку з тим, що тарифи/результати обчислень інколи мають більш ніж дві цифри після крапи, на кількість цифр після крапки немає жорсткого ліміту.

За виключенням автоматичного розрахування суми для сплати, ця форма працює за тими ж методами що й перераховані вище функції програми. Завдяки швидкодії обміну даних з базою даних MAMP, процес передачі займає менше секунди після натискання кнопки “Надіслати”, навіть не зважаючи на те, що одночасно передається велика кількість даних.

4. ОПИС ФУНКЦІОНУВАННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1. Опис роботи системи

У цьому розділі буде розглянуто сценарій використання програми користувачем. Перш за все, після ввімкнення програми, користувача зустріне форма авторизації. Ми розглянемо варіант, коли користувач новий та ще не має облікового запису. У такому випадку йому доведеться створити його через меню реєстрації. Обидві форми можна побачити на рисунку 4.1.

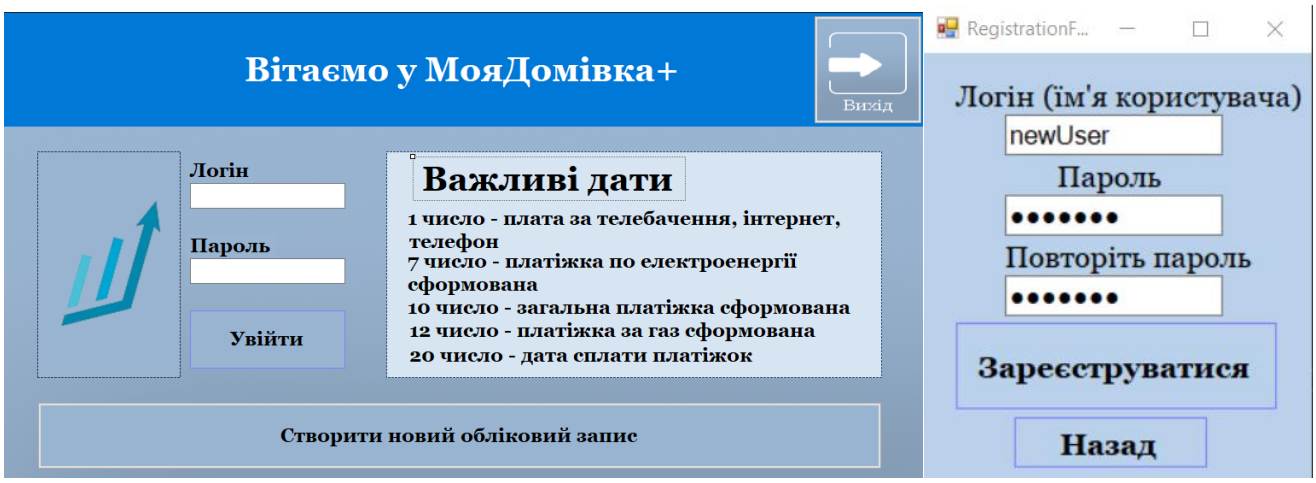


Рисунок 4.1 – форми авторизації та реєстрації

Під час реєстрації користувач має ввести однаковий пароль два рази для підтвердження. Після цього він зможе зареєструвати аккаунт і увійти до нього. Так як у додатку не використовуються системи підтвердження, аккаунт буде доступний для використання одразу після реєстрації.

На меню авторизації також можна помітити важливі дати, що можуть нагадати користувачу о необхідності сплати платежів або передачі показань. З цими датами також можна буде ознайомитись і в середині додатку на формі “Календар”. Ці дати є єдиними для всіх користувачів, а отже є незмінними.

Після реєстрації новий користувач буде доданий у базу даних і в майбутньому зможе проходити авторизацію без необхідності нової реєстрації.

Після авторизації, користувач опиниться у головному меню застосунку. Він ще не має жодного об’єкту, проте якщо він не планує використовувати функціонал декількох об’єктів – він може не створювати новий.

Рисунок 4.2 – Головне меню додатку

Для створення нового об’єкту потрібно зайти у форму “Змінити об’єкт”. На цій формі можна побачити список існуючих об’єктів для користувача, а також можливість створити новий, змінити обраний чи видалити непотрібний об’єкт. Для створення нового об’єкту необхідно ввести його адресу(чи назву). Користувачем буде створено 2 об’єкти. Обидва було занесено у базу даних.

Перемкнемося на об’єкт Дім 2. Надпис “Обраний об’єкт” зміниться на цю назву. Повернемося до головного меню. Незважаючи на зміну форми, додаток пам’ятає як користувача, так і обраний будинок.

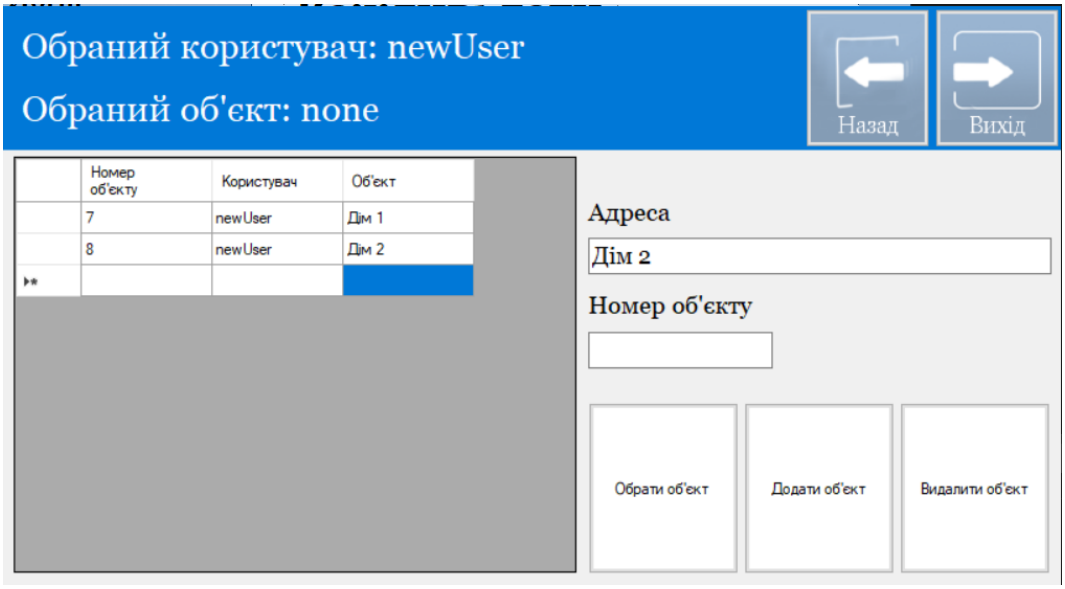


Рисунок 4.3 – Меню роботи з об’єктами

Рисунок 4.4 – головне меню додатку, користувач створив і обрав об’єкт

Перевіримо меню тарифів. Для цього потрібно перейти у вкладинку “Тарифи”. Не дивлячись на те, що користувач новий, базові тарифи автоматично введені, і тому користувач вже готовий до роботи з додатком. Поки-що залишимо тарифи за замовченням.

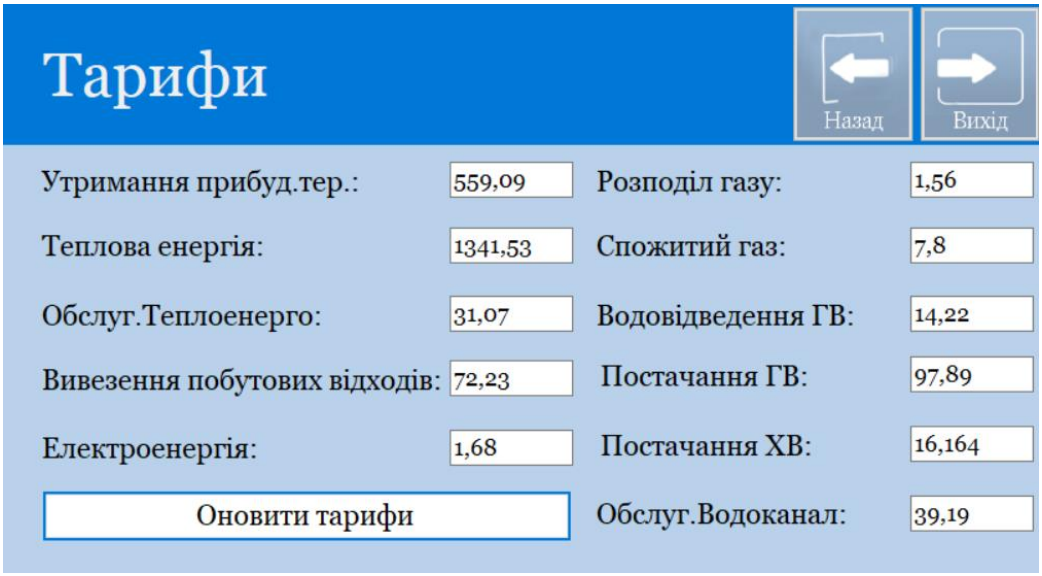


Рисунок 4.5 – Меню тарифів

Перейдемо до меню передачі показників. Так як це приклад, введемо якісь значення та передами дані до бази даних декілька разів.

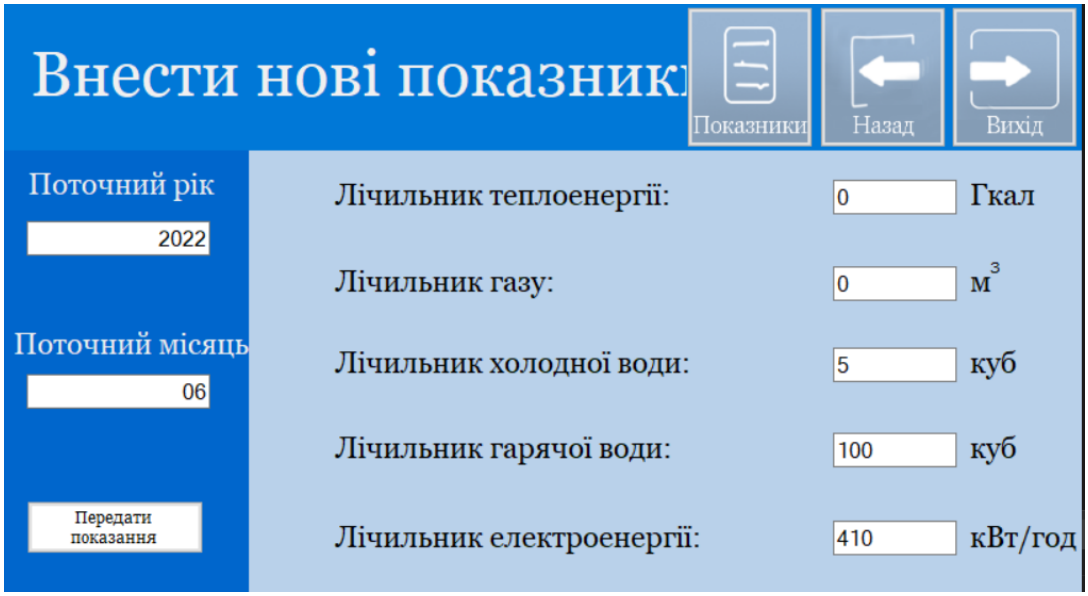


Рисунок 4.6 – Меню передачі показників

Перейшовши до меню попередніх показників можна побачити, що ці дані було прийнято та збережено у системі. Вони були записані у базу даних та виведені у таблицю. Записи сортуються за своїм унікальним ідентифікатором, проте

користувач може відсортувати їх за місяцем. Для перевірки відсортуємо дані у зворотньому порядку.

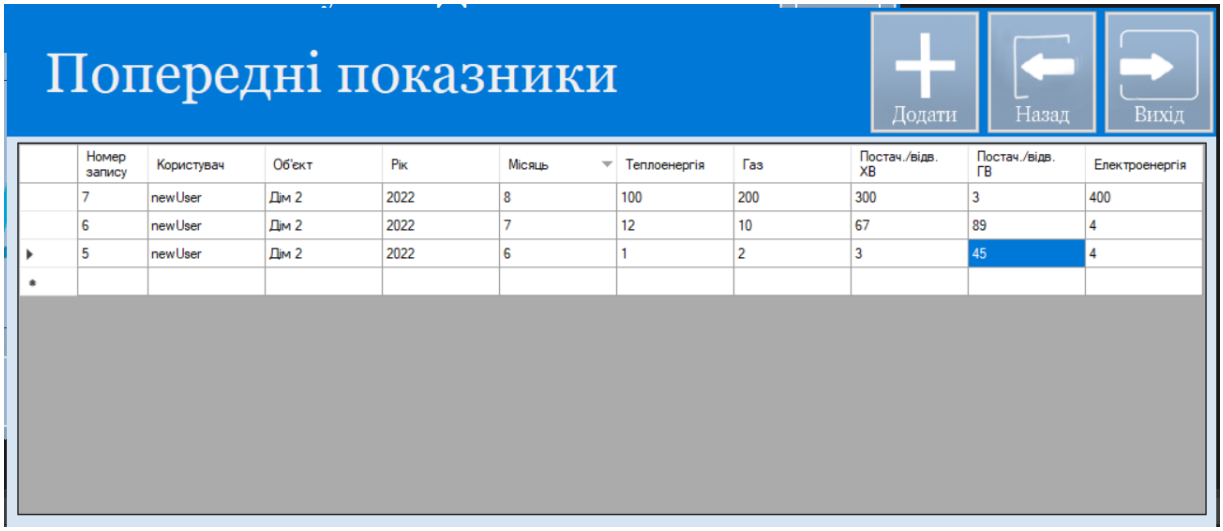


Рисунок 4.7 – таблиця з показниками лічильників

Перейдемо до внесення даних про оплату. Для цього повернемось в головне меню, та перейдемо до меню “Розрахунок оплати”.

Якщо обрано місяць, що ще немає показників – усі поля будуть пустими (рис. 4.8).

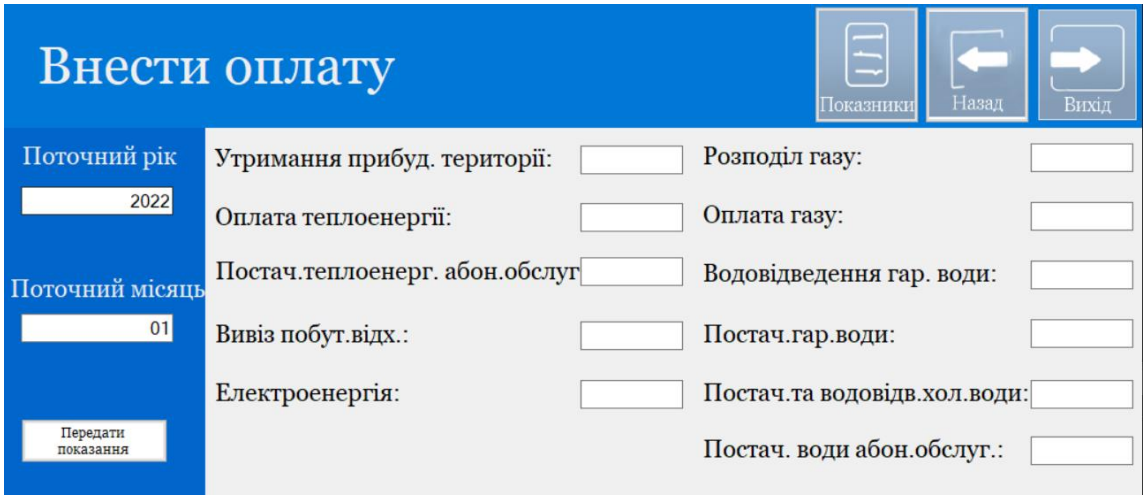


Рисунок 4.8 – Пуста таблиця внесення оплати

Проте як тільки переключаємось на інший місяць, що вже має передані показники лічільників для обрахування – поля автоматично заповнюються обрахованими даними (рис. 4.9). Передамо показники за 6 та 7 місяці.

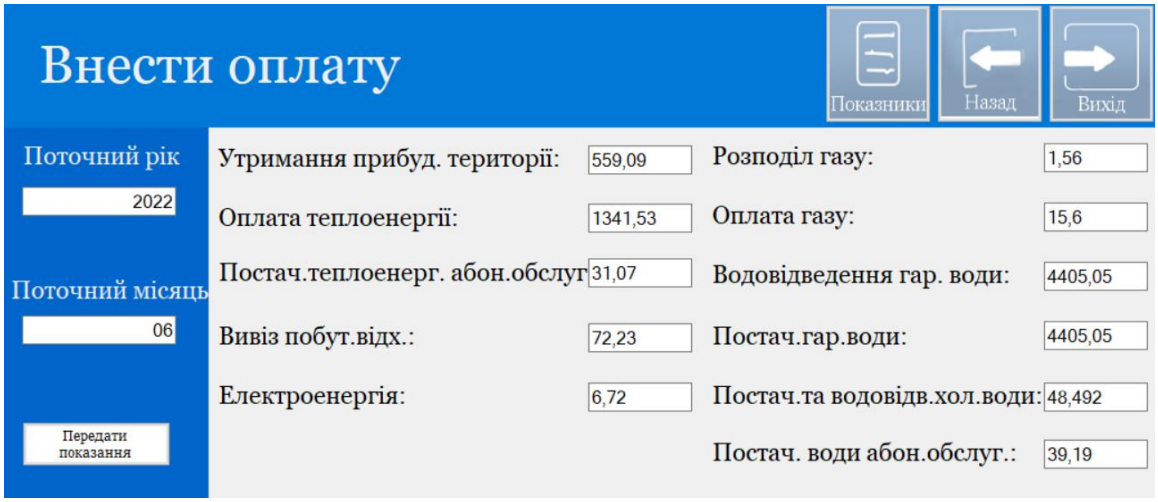


Рисунок 4.8 – Розрахована таблиця внесення оплати

Перед поданням 8-го місяцю, повернемось до тарифів і створимо користувацькі тарифи, де кожний показник дорівнює 1. Після цього передамо 8-й місяць і перейдимо до таблиці переданих показників оплат. Результат передачі оплат можна побачити на рисунку 4.9.

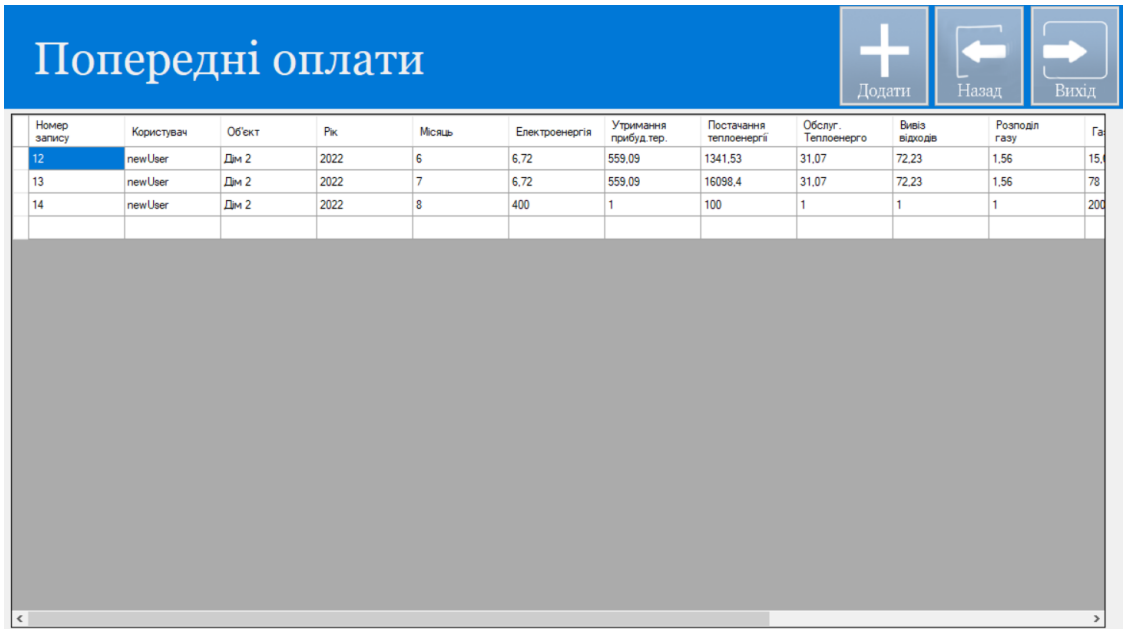


Рисунок 4.9 – Таблиця переданих оплат для Дому-2

Як можна побачити на малюнку, дані за 8-й місяць були передані вже з іншими тарифами, використаними при автоматичному розрахунку. Це особливо помітно у полях, де немає показників лічильників, а дані беруться одразу з тарифу.

Тепер користувач змінить об’єкт на Дім-1 і повернеться до цього меню. Так як дані не належать до об’єкту Дім-1, табличка знову стане пустою. За бажанням користувач може додати дані до цього об’єкту і перевірити чи будуть вони показані.

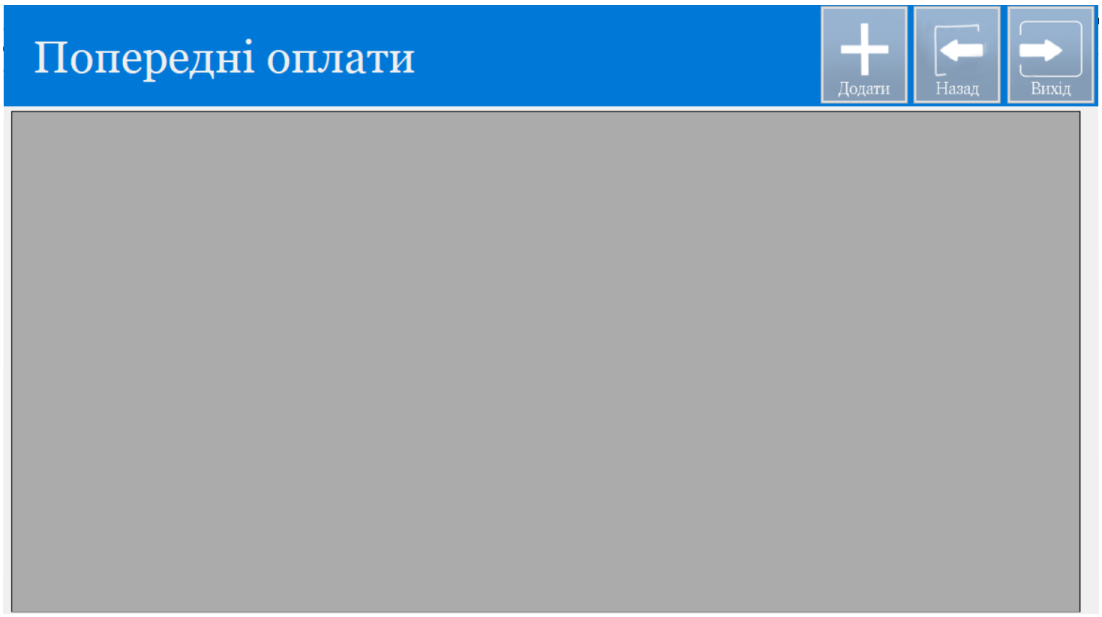


Рисунок 4.10 – таблиця переданих оплат для Дому-1

Наостанок спробуємо відкрити одне з посилань у меню та перейдемо у меню “Корисні посилання”. Тут можна побачити посилання на сервіси, що можуть знадобитися у процесі роботи з комунальними платіжками. У цьому випадку спробуємо перейти за посиланням на заміну лічильників.

Натиснемо на посилання та зачекаємо на результат. У браузері, що назначений браузером за замовченням відкриється сайт zruchnotut, де користувач зможе виконати необхідні для нього дії.

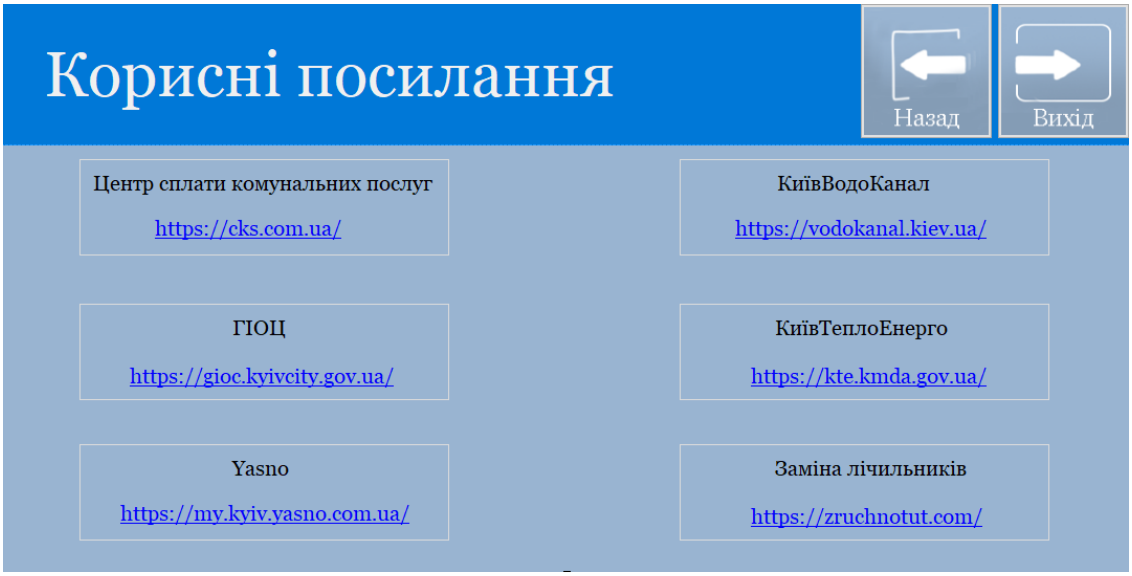


Рисунок 4.11 – Меню корисних посиланнь

Рисунок 4.12 – Результат переходу за посиланням.

При переході за посиланням основний додаток не буде закрито, він продовжить функціонувати та користувач може продовжити їм користуватись за наявності потреби.

На цьому тестовий сценарій використання додатку користувачем завершується. Користувач може закрити додаток натиснувши кновпку “Вихід”, або будь-яким іншим зручним для нього способом. Шлях виходу користувача не впливає на збереження даних та на роботу додатку у майбутньому.

4.2. Набір тестових сценаріїв для забезпечення якості продукту

З метою підтримки високого рівня якості розроблюваного програмного забезпечення було створено check-list, що містить у собі тестові ситуації для відтворення у програмі.

	A	B	C	D
1	№	Перевірка	Результат	Коментарі
2	1	LoginForm		
3	1.1	Завантаження	Passed	
4	1.2	Вихід	Passed	
5	1.3	Авторизація з вірними даними	Passed	
6	1.4	Помилка авторизації з не вірними даними	Passed	
7	2	RegistrationForm		
8	2.1	Реєстрація з вірними даними	Passed	
9	2.2	Помилка реєстрації з не вірними даними	Passed	
10	2.3	Кнопка назад	Passed	
11	3	MainMenu		
12	3.1	Від без обраного об'єкту	Passed	
13	3.2	Вхіж з обраним об'єктом	Passed	
14	3.2	Перехід у "Змінити об'єкт"	Passed	
15	3.3	Перехід у "Змінити користувача"	Passed	
16	3.4	Перехід у "Розрахунок оплати"	Passed	
17	3.5	Перехід у "Історія сплат"	Passed	
18	3.6	Перехід у "Календар подій"	Passed	
19	3.7	Перехід у "Передати показання лічильників"	Passed	
20	3.8	Перехід у "Історія показань лічильників"	Passed	
21	3.9	Перехід у "Тарифи"	Passed	
22	3.10	Перехід у "Корисні посилання"	Passed	

Рисунок 4.13 – Частина Check-list

У зв’язку з векликою кількістю ситуацій для перевірки було розроблено сам чек-ліст, адже він дозволяє покрити більшість тестових ситуацій за найменший проміжок часу. Для перевірки use-case-ів, що найбільш вірогідно виникнуть при експлуатації продукту було створено більш детальний список test-case.

Для додаткової перевірки програми було проведено тестування за допомогою методів чорного ящику та білого ящику. Тестування допомголо перевірити роботоспособність додатку, виправити помічені помилки у коді програми.

Тестування проводилось базуючих на методах граничних значень, класів еквівалентності а також стохастичне тестування. Було проведене структурне тестування коду програми за допомогою методів маршрутного тестування.

Не дивлячись на те, що тестування не може виявити усі можливі помилки у додатку, воно допомоголо підвищити якість кінцевого продукту.

4.3. Результати апробації та подальший розвиток проекту

Завдяки роботі з програмним забезпеченням та проведенні різноманітних тестів, було зроблено висновки що до сильних та слабких сторін додатку.

З сильних сторін можна відзначити:

- об’єднання дій пов’язаних з обліком комунальних платежів у одному додатку;
- простота роботи з великою кількістю об’єктів;
- швидкодія додатку;
- інтуїтивність інтерфейсу;
- зручне групування даних.

Було складено план подальшого розвитку продукту. В нього війшли такі пункти як:

- введення можливості видалення користувачів та усієї пов’язаної з ними інформації;
- введення можливості оплати комунальних рахунків прямо з додатку;
- введення додаткових можливостей по роботі з даними комунальних платежів.

Даний план включає як покращення поточного продукту, так і можливості для розширення функціоналу додатку у майбутньому.

ВИСНОВКИ

1. Проведено аналіз розробки програмного забезпечення для автоматизації ведення обліку сплати комунальних платежів. На основі проведеного аналізу визначено основні проблеми, з якими може зіштовхнутись користувач при роботі з комунальними платежами.

2. Проведено аналіз існуючих додатків для автоматизації обліку комунальних платежів та роботи з комунальних платежів, визначено їх переваги та недоліки.

3. Розроблено архітектуру додатку та бази даних, пов’язаної з ним, під час розробки було використано мову програмування C#, можливості Windows Forms, базу даних MAMP. Розробка велась у додатку Visual Studio на мові C# з використанням Windows Forms. Графічний інтерфейс додатку та його візуальні елементи було створено у графічному редакторі Adobe Photoshop.

Для контролю процесу розробки методом Kanban було використано веб-додаток WorkSection. У процесі роботи було створено Kanban дошку, шо використовувалась для контролю виконання задач та керування ходом розробки продукту.

Для формулювання вимог до проекту та постановки завдань було створено різноманітні діаграми, включаючи діаграму варіантів використання, ER-діаграму та діаграму класів.

4. Завдяки комплексним методикам тестування, що включають створення чек-лісту та test-case-ів, та багаторазовому практичному тестуванню, було проаналізовано та виявлено ряд переваг та визначено перспективи розвитку системи.

Результати дослідження бакалаврської роботи апробовані на всеукраїнських науково-технічних конференціях: “XIV науково-технічна конференція студентів та молодих вчених: «Сучасні інфокомунікаційні технології” та “Застосування програмного забезпечення в інфокомунікаційних технологіях”.

ПЕРЕЛІК ПОСИЛАНЬ

1) Головне про зручність в інтерфейсах [Електронний ресурс] // Creative Practices. – 2021. – Режим доступу до ресурсу: https://cases.media/article/golovne-pro-zruchnist-v-interfeisakh.
2) What Languages Should You Use to Develop Web Applications in 2021? [Електронний ресурс] // Avianei. – 2021. – Режим доступу до ресурсу: https://www.avianet.aero/what-languages-should-you-use-to-develop-web-applications-in-2021/.
3) Мови програмування для мобільної розробки [Електронний ресурс] // evantoTuts+. – 2017. – Режим доступу до ресурсу: https://code.tutsplus.com/uk/articles/mobile-development-languages--cms-29138.
4) Центр Комунальних Послуг [Електронний ресурс] – Режим доступу до ресурсу: https://cks.com.ua/cabinet/objects/.
5) Yasno [Електронний ресурс] – Режим доступу до ресурсу: https://my.kyiv.yasno.com.ua/summary.
6) Коммуналочка [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.blogspot.accountingutilities&hl=ua&gl=US.
7) Додаток Дах [Електронний ресурс] – Режим доступу до ресурсу: https://dah-online.com/tenants.
8) Додаток "Комуналка" [Електронний ресурс] – Режим доступу до ресурсу: https://cks.com.ua/komunalka/.
9) Benefits of C# [Електронний ресурс] // Codeguru. – 2021. – Режим доступу до ресурсу: https://www.codeguru.com/csharp/benefits-of-c/.
10) Платформа .NET та її застосування для ООП Додати до моєї бази знань [Електронний ресурс] // Портал Знань – Режим доступу до ресурсу: http://www.znannya.org/?view=csharp-dotNET

11) C# 4.0 Unleashed – Bart: Sams Publishing, 2011. – (Sams Publishing). – (Chapter 5).

12) The 4 Best Local WordPress Development Environments in 2021: XAMPP vs MAMP vs Local vs DesktopServer [Електронний ресурс] // VP Migrate. – 2017. – Режим доступу до ресурсу: https://deliciousbrains.com/xampp-mamp-local-dev/.

13) LucidChart [Електронний ресурс] – Режим доступу до ресурсу: https://www.lucidchart.com/pages/product.

14) ER-діаграма, Entety-Relations діаграми [Електронний ресурс] // KaguTech – Режим доступу до ресурсу: https://ukr.kagutech.com/3922610-the-er-diagram-is-..-description-views-construction-rules.

15) A Review Of The Minimum Viable Product Approach [Електронний ресурс] // Forbes – Режим доступу до ресурсу: https://www.forbes.com/sites/theyec/2021/12/08/a-review-of-the-minimum-viable-product-approach/?sh=23af49072e20.

16) The Kanban system for agile software development explained [Електронний ресурс] // K&C. – 2022. – Режим доступу до ресурсу: https://kruschecompany.com/kanban-method-agile-software-development/.

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для автоматизації обліку сплати комунальних послуг громадян засобами мови C#

Виконав студент 4 курсу
групи ПД-41
Панібратов Андрій Іванович
Керівник роботи
К.т.н., доц., доцент кафедри ІПЗ Золотухіна Оксана Анатоліївна

Київ – 2022

Аналіз аналогів

Застосунок		Переваги	Недоліки
	ЦКС	- Великий набір даних - Автоматичний розрахунок платежів	- Відсутність можливості налаштування тарифів при розрахунку - Перевантаження інформацією
Yasno	Yasno	- Сконцентрованість на одному типу комунальних платежів - Простий та доступний інтерфейс	- Недостатньо детальна інформація по минулих платежах - Лише один об'єкт на користувача
	Коммуналочка	- Швідке налаштування тарифів та проведення розрахунків - Детальна історія платежів	- Заплутаний інтерфейс - Лімітована кількість об'єктів на користувача
	Дах	- Зручне перемикання між об'єктами - Можливість створення коментарів	- Перевантаження функціоналом, що не відноситься до роботи з комунальними послугами
	Комуналка	- Мінімалістичний інтерфейс - Можливість створення комплексних тарифів	- Інформація занадто децентралізована - Незрозумілість інтерфейсу

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – підвищення зручності обліку сплати комунальних послуг громадян за рахунок автоматизації процесу із використанням програмного забезпечення

Об’єкт дослідження – процес обліку сплати комунальних послуг громадян

Предмет дослідження – програмне забезпечення для автоматизації обліку сплати комунальних послуг громадян

3

ТЕХНІЧНІ ЗАВДАННЯ

1. Внесення нових записів по комунальним платежах.
2. Доступ до історії минулих платежів.
3. Встановлення власних тарифів.
4. Автоматичний розрахунок платіжки по поточному місяцю.
5. Ведення звітності по декільком об’єктам для одного користувача.
6. Посилання на засоби оплати.

4

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Visual Studio

Багатофункціональний застосунок для написання самого коду додатку

MAMP

Застосунок для роботи з базами даних, Зокрема з локальними БД

Photoshop

Потужний графічний Редактор для створення елементів інтерфейсу

Мова C#

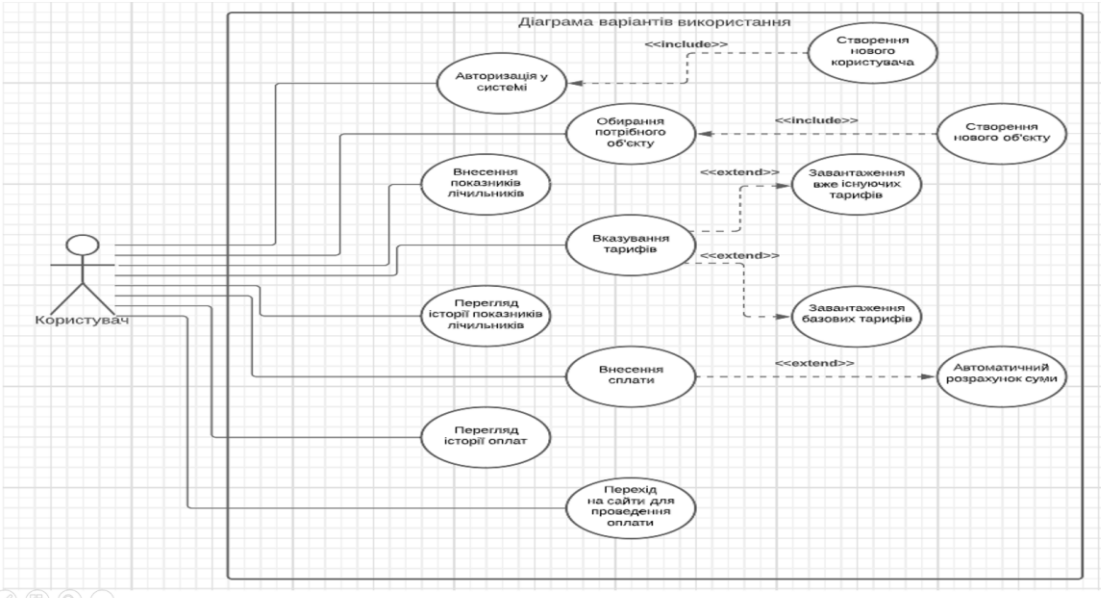
Сучасна мова програмування для створення комп'ютерних та мобільних додатків

MySQL

Система бази даних

5

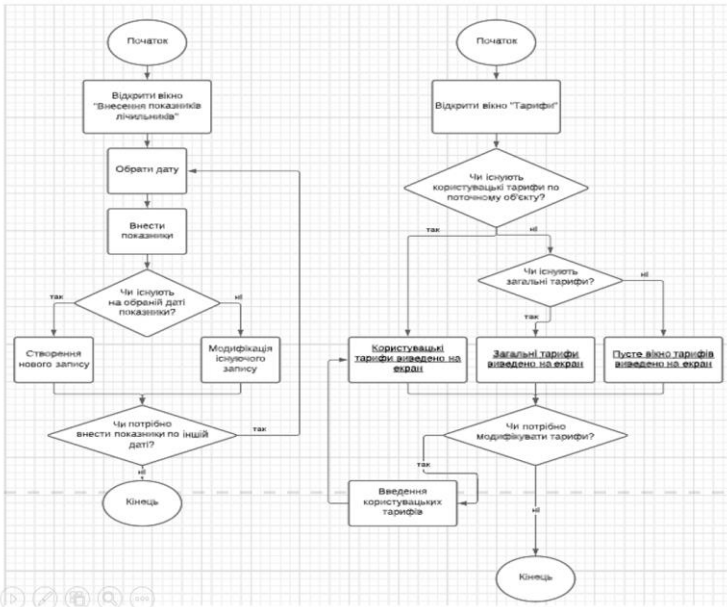
Діаграма варіантів використання



6

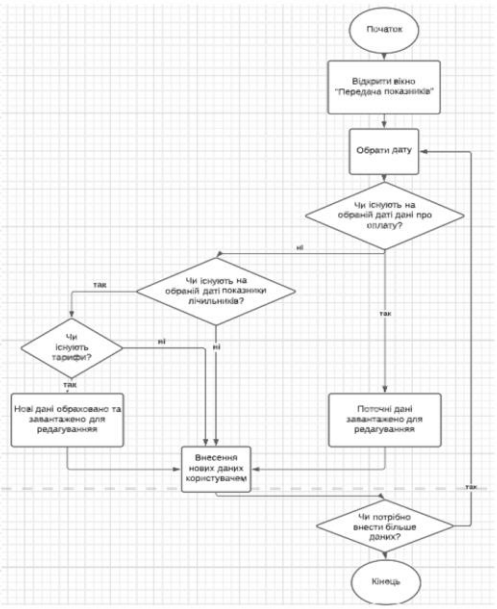
Блок-схеми головних функцій

Внесення показників лічильників



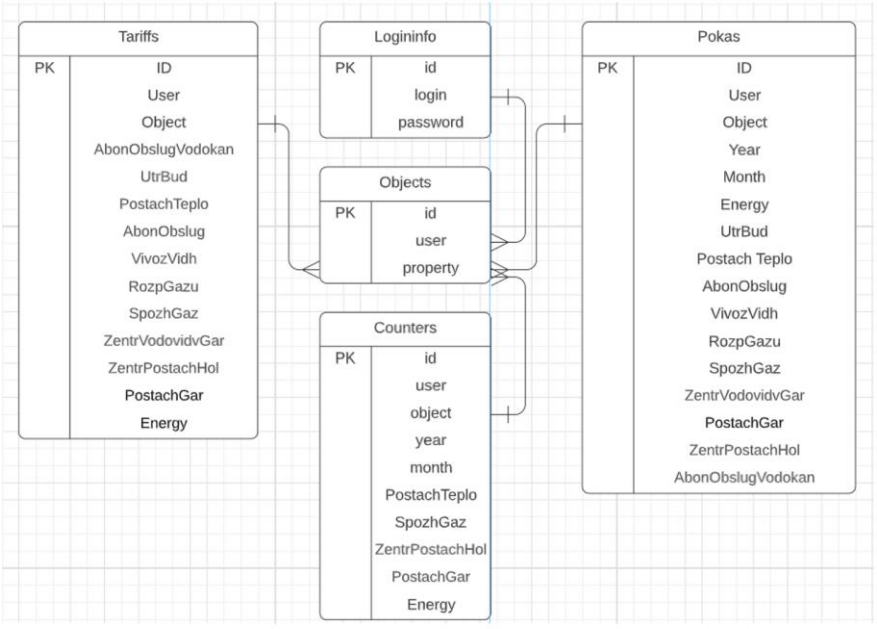
Перегляд та редагування тарифів

Розрахунок та передача даних про сплати



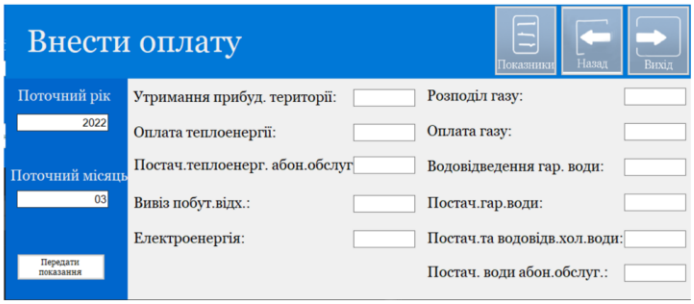
7

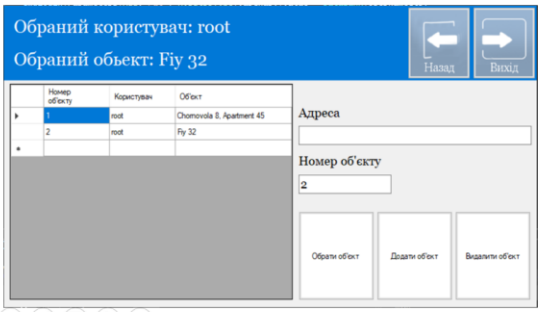
ER-діаграма бази даних

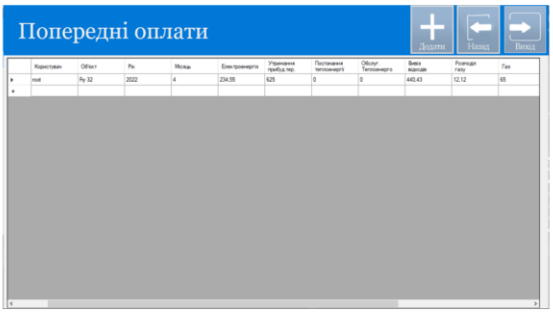


8

Екранні форми

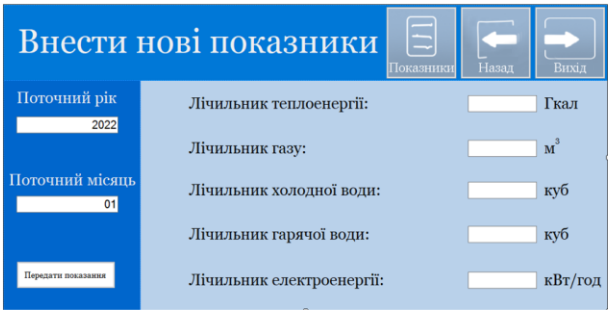


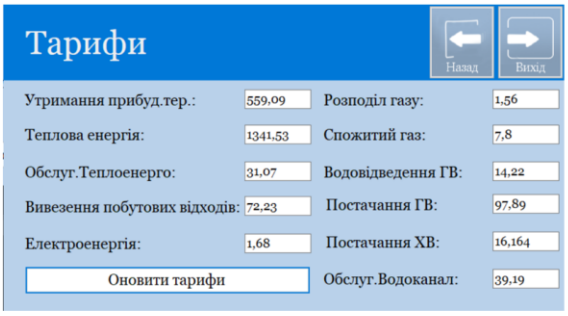


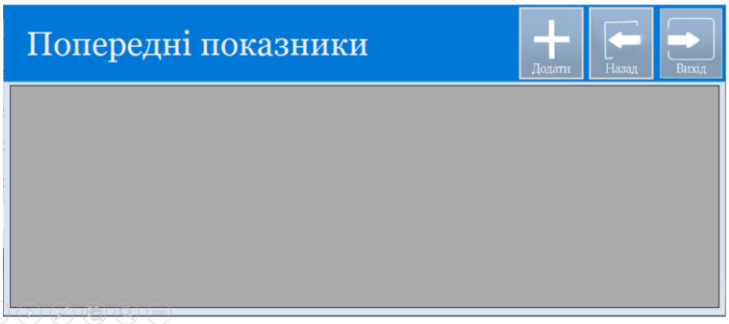


9

Екранні форми







10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Панібратов А.І. Огляд та аналіз інтерфейсу додатків для автоматизації обліку комунальних послуг / А.І. Панібратов, О.А. Золотухіна / XIV науково-технічна конференція студентів та молодих вчених : Сучасні інфокомунікаційні технології. Збірник тез, 19.05.2022, ДУТ, м.Київ – К.: ДУТ, 2022, с.36.

2. Панібратов А.І. Аналіз програмного забезпечення для автоматизації сплати та обліку комунальних послуг громадян / А.І. Панібратов, О.А. Золотухіна / Науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технологіях". Збірник тез, 20.04.2022, ДУТ, м.Київ – К: ДУТ, 2022

ВИСНОВКИ

1. Проведено аналіз розробки програмного забезпечення для автоматизації ведення обліку сплати комунальних платежів. На основі проведеного аналізу визначено основні проблеми, з якими може зіштовхнутись користувач при роботі з комунальними платежами.
2. Проведено аналіз існуючих додатків для автоматизації обліку комунальних платежів та роботи з комунальних платежів, визначено їх переваги та недоліки.
3. Було розроблено архітектуру додатку та бази даних, пов’язаної з ним, під час розробки було використано мову програмування C#, можливості Windows Forms, базу даних MAMP.
4. Завдяки комплексним методикам тестування, що включають створення чек-листу та test-case-ів, та багаторазовому практичному тестуванню, було проаналізовано та виявлено ряд переваг та визначено перспективи розвитку системи.

ДЯКУЮ ЗА УВАГУ!