Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного застосунку для оптимізації процесів управління персоналом»

Виконав: студент IV курсу
групи 1ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Борецький В. О.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Ткаченко О. М.
(прізвище та ініціали)

Рецензент:. к.т.н., доц. каф ОТ Кадук О. В.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – «Інформаційні технології»
Спеціальність 121 – «Інженерія програмного забезпечення»
Освітньо-професійна програма – «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
ФОП Михальнюк О. О.
«21» березня 2025 року

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор Романюк О. Н.
«21» березня 2025 року

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

Борецькому Віталію Олеговичу

1. Тема роботи – розробка програмного застосунку для оптимізації процесів управління персоналом.
Керівник роботи: Ткаченко Олександр Миколайович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від «20» березня 2025 р. № 97

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: дані про співробітників, кадрові процеси, вимоги до алгоритмів, специфікація інтерфейсу, структура даних для MySQL, тестові дані, зображення інтерфейсу.

4. Зміст текстової частини: аналіз технологій управління персоналом, порівняння аналогів, методи автоматизації, постановка задач, розробка моделей, алгоритмів, інтерфейсу, модулів (реєстрація, облік завдань, меню), тестування, структура програми, інструкції, висновки, література, додатки.

5. Перелік ілюстративного матеріалу: галузі застосування, етапи розробки, схеми алгоритмів, інтерфейс програми, схема бази даних, діаграми модулів, результати тестування, аналітика продуктивності.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ткаченко О. М., к.т.н., доцент кафедри ПЗ	24.03.2025	01.06.2025

7. Дата видачі завдання 24 березня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Аналіз розвитку технологій управління персоналом	25.03.25 – 02.04.25	Вик.
2.	Розробка архітектури та методів програмного продукту	03.04.25 – 14.04.25	Вик.
3.	Розробка програмних модулів застосунку	15.04.25 – 02.05.25	Вик.
4.	Тестування і практичне застосування системи	03.04.25 – 26.04.25	Вик.
5.	Оформлення матеріалів до захисту БКР	27.05.25 – 30.05.25	Вик.

Студент (підпис) **Борецький В. О.** (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи (підпис) **Ткаченко О. М.** (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.912.032.26

Борецький В. О. Розробка програмного застосунку для оптимізації процесів управління персоналом: магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 104 с.

На укр. мові. Бібліогр.: 27 назв; рис.: 27; табл. 4.

У бакалаврській кваліфікаційній роботі розроблено програмні модулі застосунку для оптимізації процесів управління персоналом. Проєкт включає розробку модулів для обліку співробітників, автоматизації кадрових процесів та аналітики ефективності персоналу. Додаток призначений для підвищення точності та швидкості прийняття управлінських рішень.

Програмні модулі, розроблені в проєкті, можуть бути використані у відділах кадрів підприємств, рекрутингових агенціях та інших організаціях, де важливо ефективне управління персоналом.

Додаток розроблено з використанням мов програмування Python та Dart у середовищах VS Code та Android Studio відповідно. Для зберігання та обробки даних використовується база даних MySQL.

ANNOTATION

Boretskyi, V. O. Development of a Software Application for Optimizing Personnel Management Processes: Master's qualification thesis in the specialty 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2025. 103 p.

In Ukrainian. Bibliography: 27 titles; figures: 27; tables: 4.

In the bachelor's thesis, software modules of an application for optimizing personnel management processes were developed. The project includes the development of modules for employee record-keeping, automation of HR processes, and personnel performance analytics. The application is designed to enhance the accuracy and speed of managerial decision-making.

The software modules developed in the project can be used in HR departments of enterprises, recruiting agencies, and other organizations where effective personnel management is crucial.

The application was developed using the Python and Dart programming languages in the VS Code and Android Studio environments, respectively. MySQL was used as the database for data storage and processing.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасні технологічні тенденції потребують постійного вдосконалення підходів до управління персоналом, особливо з урахуванням цифрової трансформації бізнес-процесів. Одним із важливих викликів у цій сфері є підвищення ефективності роботи з кадрами шляхом автоматизації ключових завдань та впровадження інноваційних рішень для аналізу продуктивності співробітників [1].

Багато організацій досі використовують застарілі методи обліку та управління кадрами, що призводить до збільшення витрат часу, зниження точності даних і неефективного розподілу ресурсів. У цьому контексті впровадження програмних модулів для автоматизації кадрових процесів дозволяє значно спростити управлінські завдання, мінімізувати ризик людських помилок і покращити прийняття стратегічних рішень [2].

Сучасні програмні застосунки для управління персоналом можуть не тільки забезпечувати ведення бази співробітників, але й аналізувати їх продуктивність, прогнозувати кадрові зміни та оптимізувати навантаження на окремі відділи компанії. Більшість доступних рішень на ринку є комерційними, що робить розробку власного програмного забезпечення актуальною та економічно вигідною альтернативою, тому актуальною є розробка даного застосунку [3].

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та задачі дослідження. Метою бакалаврської кваліфікаційної роботи є розширення функціональних можливостей програмного застосунку для управління персоналом шляхом розробки програмних модулів, які автоматизують облік співробітників, оптимізують кадрові процеси та підвищують точність аналітики продуктивності персоналу, зокрема за допомогою інтеграції фінансового трекера для відстеження доходів і витрат.

Основними задачами дослідження є:

- розробка алгоритмів обробки кадрових даних, що забезпечать високу точність, узгодженість і швидкість управління персоналом;

- розробка модуля для автоматизації кадрових процесів, який буде вести особисті дані співробітників та облік робочого часу;

- створення зручного графічного інтерфейсу, який забезпечить інтуїтивно зрозумілу взаємодію з системою;

- інтеграція всіх розроблених модулів у єдину систему для ефективного управління персоналом та прийняття обґрунтованих рішень.

Об'єктом дослідження є процес автоматизації управління персоналом у комп’ютерних системах.

Предметом дослідження є методи та засоби розробки програмних модулів для оптимізації кадрових процесів і аналізу ефективності персоналу.

Методи дослідження. В процесі виконання роботи використовувалися методи аналізу даних для розробки алгоритмів обробки кадрової інформації; принципи проєктування баз даних для ефективного зберігання даних; об’єктно-орієнтоване програмування (Python, Dart) для створення програмних модулів і інтерфейсу; методи тестування для перевірки функціональності застосунку; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Вперше запропоновано архітектуру програмного застосунку, особливістю якої є інтеграція клієнт-серверної моделі з локальним кешуванням даних у Flutter (Dart), що дозволяє відстежувати фінансові дані в реальному часі навіть за відсутності стабільного інтернет-з’єднання.

2. Отримав подальший розвиток метод динамічного програмування, який, на відміну від існуючих, було застосовано для оптимізації процесів управління персоналом, зокрема при розподілі завдань і підрахунку доходів працівників, що дозволило скоротити час на аналіз фінансових даних порівняно з традиційними системами.

Практична значення одержаних результатів. На основі теоретичних положень, викладених у бакалаврській кваліфікаційній роботі, розроблено алгоритми та програмні модулі для автоматизації процесів управління персоналом. Ці засоби забезпечують ефективний облік співробітників, автоматизацію кадрових процесів і аналіз продуктивності, що дозволяє підвищити точність і швидкість прийняття управлінських рішень. Модулі придатні для адаптації в мобільних додатках, що забезпечує гнучкість і зручність використання.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. Наукові праці опубліковано одноосібно.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні наукові результати, викладені у бакалаврській кваліфікаційній роботі, доповідалися на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2025) і XIII Міжнародній науково-практичній конференції з інформаційних систем та технологій Infocom Advanced Solutions 2025 (Київ, 2025 р.).

Публікації. Основні результати досліджень було опубліковано у матеріалах LIV Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету і XIII Міжнародної науково-практичної конференції з інформаційних систем та технологій Infocom Advanced Solutions 2025 [4, 5].

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ УПРАВЛІННЯ ПЕРСОНАЛОМ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану технологій управління персоналом з використанням програмних модулів

У сучасному бізнес-середовищі управління персоналом є однією з найважливіших функцій, яка безпосередньо впливає на ефективність організації. З кожним роком вимоги до цієї сфери зростають, і традиційні методи управління все частіше замінюються автоматизованими рішеннями, що забезпечують більш високий рівень ефективності та точності в обробці даних.

Одним з найбільш перспективних напрямків є розробка програмних модулів для оптимізації процесів управління персоналом. Сучасні інформаційні технології дозволяють автоматизувати безліч операцій, таких як облік робочого часу, планування відпусток, розрахунок заробітної плати, а також аналіз продуктивності співробітників. Завдяки цьому, організації можуть зменшити витрати часу та ресурсів, а також значно знизити ймовірність помилок, які часто виникають при ручному введенні даних.

Особливу увагу слід приділити інтеграції цих програмних модулів з іншими системами компанії, що дозволяє створювати єдину платформу для управління всіма аспектами роботи з персоналом. Такі рішення можуть включати в себе бази даних співробітників, звіти про їхню продуктивність, автоматизовані процеси найму та адаптації нових співробітників.

Використання програмних модулів для управління персоналом не тільки оптимізує внутрішні процеси організації, але й забезпечує більш точний моніторинг та аналіз діяльності кожного співробітника, що дозволяє керівникам приймати обґрунтовані рішення для покращення роботи компанії в цілому.

Крім того, впровадження програмних модулів для управління персоналом відкриває нові можливості для стратегічного планування та розвитку. Збирання та аналіз даних про співробітників у реальному часі дозволяє не лише оперативно вирішувати поточні завдання, а й прогнозувати потреби в кадрах, визначати

ключові напрямки для розвитку персоналу та створювати індивідуальні плани кар'єрного зростання. Це дозволяє організаціям не тільки підвищувати ефективність наявних ресурсів, а й залучати нові таланти, що сприяє сталому росту та інноваційному розвитку бізнесу в умовах високої конкуренції [6-8].

Отже, розробка програмного застосунку для управління персоналом є важливою складовою сучасних організацій, яка дозволяє значно підвищити ефективність їхньої діяльності, забезпечити прозорість процесів та створити умови для розвитку співробітників.

1.2 Порівняльний аналіз аналогів

У сучасних умовах автоматизація управління персоналом відіграє ключову роль у забезпеченні ефективного функціонування підприємств. Використання спеціалізованих програмних рішень дозволяє знизити адміністративне навантаження, підвищити точність обліку даних і покращити взаємодію між співробітниками та керівництвом.

На ринку представлені різні програмні продукти, такі як Square, CalendarSpots, Mindbody, а також власний розроблений модуль. Кожна з цих систем має свої особливості, переваги та недоліки, що визначають їхню доцільність використання в конкретних бізнес-середовищах.

Однак, як показує проведений аналіз, розроблений власний модуль має низку значних переваг над аналогами, зокрема в аспектах гнучкості налаштувань, автоматизація кадрового обліку, аналітичні можливості, простота використання та унікальні можливості. Завдяки цьому він є найбільш оптимальним вибором для підприємств, які прагнуть отримати індивідуальне рішення, що повністю відповідає їхнім потребам.

Square – це універсальна платформа для бізнесу, що включає інструменти для керування персоналом, фінансами та оплатою послуг. Вона дозволяє власникам малого бізнесу легко відстежувати графіки роботи співробітників, керувати платежами та автоматизувати процеси реєстрації клієнтів. Square також забезпечує можливість інтеграції з іншими сервісами, такими як бухгалтерські програми та

CRM-системи (див. рисунок 1.1) [9].

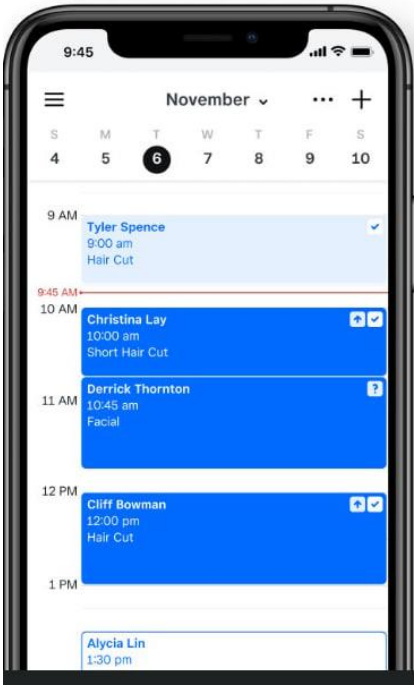


Рисунок 1.1 – Приклад роботи застосунку Squre

CalendarSpots є спеціалізованим інструментом для планування розкладів та управління записами на зустрічі. Ця система орієнтована на сфери, де важливо точно планувати робочий час, наприклад у медичних установах, салонах краси та фітнес-центрах. CalendarSpots дозволяє клієнтам самостійно бронювати час, автоматично нагадує про зустрічі та надає інструменти для аналізу завантаженості персоналу (див. рисунок 1.2) [10].

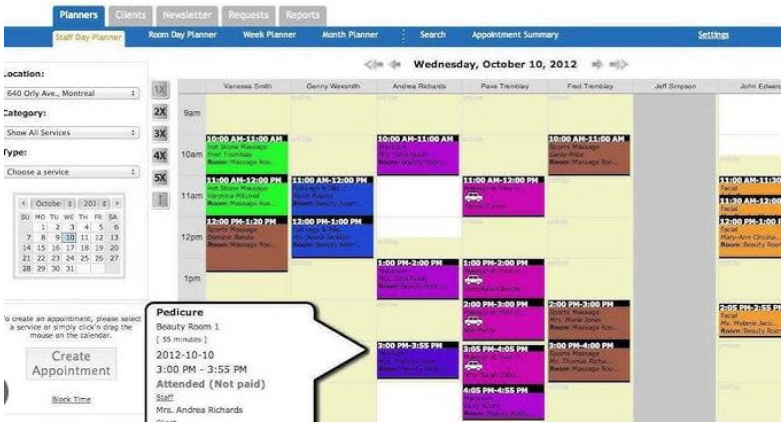


Рисунок 1.2 – Приклад роботи застосунку CalendarSpots

Mindbody – це платформа, яка розроблена для бізнесів у сфері здоров'я та фітнесу. Вона поєднує функції управління персоналом, планування записів, фінансового обліку та взаємодії з клієнтами. Однією з головних переваг Mindbody є можливість інтеграції з маркетинговими інструментами та системами лояльності, що робить її зручною для фітнес-клубів, йога-студій та салонів краси (див. рисунок 1.3) [11].

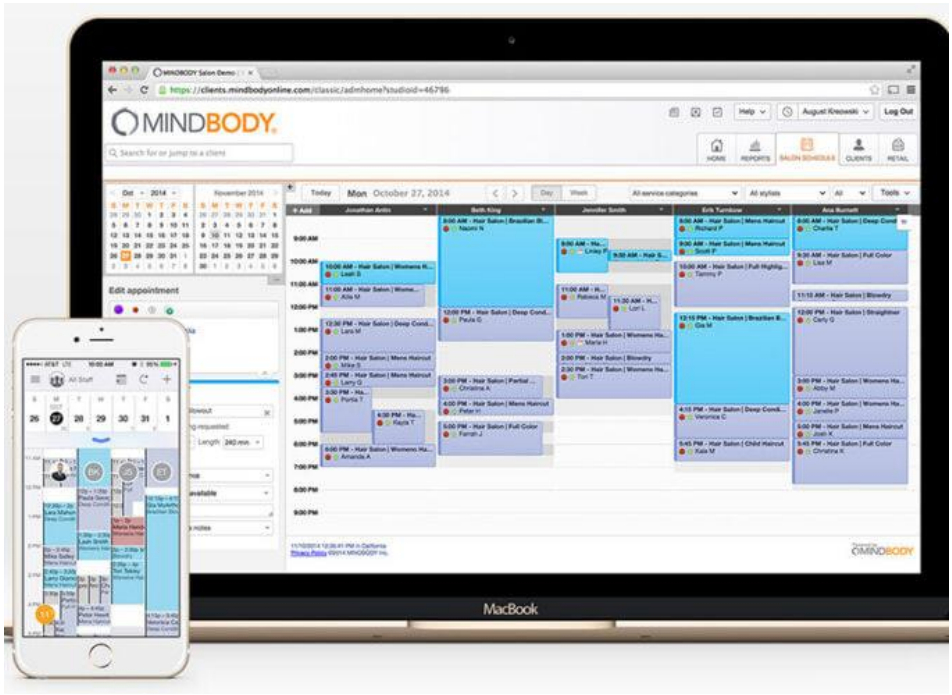


Рисунок 1.3 – Приклад роботи застосунку Mindbody

Розробка програмного застосунку для оптимізації процесів управління персоналом вимагає досвіду роботи з управлінським програмним забезпеченням, розробки функціональних модулів та дизайну інтерфейсу користувача. Це передбачає створення внутрішньої системи, яка буде здатна автоматизувати процеси кадрового обліку, аналізувати продуктивність та керувати кадровими ресурсами. Модулі повинні забезпечувати гнучкість налаштувань для різних організацій, включати аналітичні можливості для прийняття рішень. Складова інтерфейсу користувача передбачає розробку інтуїтивно зрозумілих інтерфейсів, що дозволять користувачам ефективно взаємодіяти з розробленими алгоритмами та здійснювати управлінські функції без складнощів.

Результати порівняння аналогів зведено в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	Squre	CalendaSpots	Mindbody	Власний застосунок
Гнучкість налаштувань	+	+	-	+
Автоматизація кадрового обліку	-	+	+	+
Аналітичні можливості	+	-	+	+
Простота використання	+	-	-	+
Унікальні функції	-	-	-	+
Загальна оцінка	60%	40%	60%	100%

Відповідно до таблиці порівняльних характеристик, розробка власного програмного застосунку для оптимізації процесів управління персоналом є доцільною. Отриманий продукт зможе усунути недоліки існуючих рішень та забезпечити розширені можливості для автоматизації кадрового обліку, аналізу продуктивності та управління кадровими ресурсами в організації.

Отже, розробка програмного застосунку для управління персоналом відкриває перед бізнес-середовищем широкі можливості для покращення ефективності використання людських ресурсів. Завдяки інтеграції інтуїтивно зрозумілих інтерфейсів і аналітичних функцій, керівники та спеціалісти можуть приймати більш обґрунтовані та швидкі рішення щодо оптимізації роботи персоналу, що є критичним для успішного функціонування організації.

1.3 Аналіз методів розв’язання задачі

Розробка програмного застосунку для оптимізації процесів управління персоналом вимагає комплексного підходу до вирішення завдань, пов'язаних з автоматизацією та аналізом даних про співробітників, їх ефективність та розвиток. Існують різні методи, кожен з яких має свої переваги та недоліки. У цьому розділі проаналізовано найбільш ефективні методи розробки таких програмних модулів.

Один з основних аспектів розробки програмного застосунку для оптимізації управління персоналом – це автоматизація збору та аналізу даних про працівників. Для цього можуть бути використані алгоритми збору та обробки даних із різних джерел, таких як внутрішні системи обліку часу та показники ефективності. Проте, цей підхід має певні обмеження, оскільки він вимагає великих обчислювальних потужностей та може мати низьку точність, якщо не правильно налаштовані процеси збору даних. Тому для такого підходу потрібно розробляти додаткові механізми верифікації даних.

Інший підхід полягає в використанні алгоритмів на основі машинного навчання для аналізу великих обсягів даних про персонал. Ці алгоритми дозволяють виявляти закономірності та тренди, що допомагають у прийнятті управлінських рішень. Наприклад, можна застосовувати методи кластеризації для групування співробітників за різними характеристиками або прогнозувати ефективність роботи в залежності від різних факторів. Однак цей підхід потребує великої кількості історичних даних для навчання моделей і може бути обмежений у випадках, коли дані є неповними або неякісними.

Найефективнішим підходом є використання інтегрованих платформ для управління персоналом, які об'єднують різні модулі для збору, зберігання та аналізу даних. Ці платформи дозволяють забезпечити єдину базу для обліку всіх аспектів роботи з персоналом. Такі системи дозволяють здійснювати моніторинг ключових показників персоналу, автоматизувати процеси звітності та надавати зручні інтерфейси для керівників [12].

Розглянувши переваги та недоліки кожного методу, було прийнято рішення комбінувати алгоритми машинного навчання з інтегрованими платформами для

управління персоналом. Це дозволить отримати більш точні та ефективні результати в оптимізації процесів управління персоналом, зокрема завдяки автоматизації збору та аналізу даних, а також прогнозуванню ефективності роботи персоналу. Такий підхід гарантує високий рівень точності та гнучкості у прийнятті управлінських рішень, що робить систему ефективною для різних організацій.

1.4 Постановка задач бакалаврської кваліфікаційної роботи

Проаналізувавши переваги та недоліки існуючих програмного застосунку для оптимізації процесів управління персоналом, було визначено наступні завдання, які необхідно виконати для успішної розробки програмного забезпечення:

- розробка алгоритмів обробки кадрових даних, що забезпечать високу точність, узгодженість і швидкість управління персоналом;
- розробка модуля для автоматизації кадрових процесів, який буде вести особисті дані співробітників та облік робочого часу;
- створення зручного графічного інтерфейсу, який забезпечить інтуїтивно зрозумілу взаємодію з системою;
- інтеграція всіх розроблених модулів у єдину систему для ефективного управління персоналом та прийняття обґрунтованих рішень.

1.5 Висновки

У першому розділі було розглянуто існуючі програмні застосунки для управління персоналом, такі як SAP SuccessFactors, Workday та BambooHR, що використовуються для автоматизації процесів збору даних та оцінки продуктивності працівників. Вивчено їхні переваги та недоліки, що дозволило визначити основні проблеми, які потребують вирішення.

На основі аналізу існуючих програмного застосунку для управління персоналом було визначено завдання для розробки власного програмного модуля, спрямованого на оптимізацію процесів управління персоналом. Основними завданнями є автоматизація збору даних про ефективність працівників, розробка

системи для управління кар'єрним розвитком, навичками та плануванням трудових ресурсів

Ці завдання дозволяють створити ефективний інструмент для поліпшення управління персоналом, підвищення продуктивності та оптимізації внутрішніх процесів у компанії.

2 РОЗРОБКА МОДЕЛЕЙ ТА МЕТОДІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних даних системи

Розробка програмного застосунку для оптимізації процесів управління персоналом потребує використання сучасних технологій і підходів, які забезпечать ефективну роботу системи. Для цього буде створена основа, що включатиме базу даних для зберігання всієї необхідної інформації, серверну частину для обробки даних і клієнтську частину для зручної взаємодії користувачів із програмою. База даних, побудована на основі SQL, дозволить організовано зберігати всі відомості, які стосуються роботи з персоналом, і забезпечить швидкий доступ до них. Серверна частина буде розроблена з використанням Python у середовищі VSCode, що дасть змогу обробляти запити та управляти інформацією. Для створення інтерфейсу користувача буде використано мову Dart у середовищі Android Studio, що допоможе зробити програму зручною та доступною для використання.

Процес створення програми передбачає кілька важливих етапів, пов’язаних із аналізом даних, які надходитимуть у систему. Спочатку буде розроблено частину програми, яка відповідатиме за взаємодію з користувачами, щоб вони могли легко увійти в систему та почати роботу. Цей компонент забезпечить базові функції для доступу до програми та роботи з основними можливостями. Далі буде створено модуль, який займатиметься обробкою та збереженням інформації, що надходить від користувачів, наприклад, даних про працівників, їхні робочі графіки чи інші пов’язані відомості.

Наступним кроком стане розробка модуля, який допомагатиме автоматизувати певні процеси, пов’язані з управлінням персоналом. Цей модуль братиме введені дані, аналізуватиме їх і формуватиме потрібні результати, які можна буде використовувати для подальшої роботи. Також буде створено частину програми, яка відповідатиме за відображення інформації у зрозумілому вигляді, щоб користувачі могли швидко переглядати потрібні дані та приймати рішення на їх основі.

Для реалізації всіх цих частин програми знадобиться добре розібратися в

технологіях, які використовуються, і правильно організувати роботу з даними. У процесі розробки будуть застосовані різні інструменти, які допоможуть створити серверну частину, інтерфейс і базу даних. Після завершення основних етапів розробки буде проведено перевірку всіх компонентів, щоб переконатися, що система працює стабільно і відповідає потребам користувачів [13-18].

Отже, створення програми для управління персоналом із використанням SQL, Python і Dart – це завдання, яке потребує уважного підходу до роботи з даними та їх обробки. Усі частини системи мають бути добре пов’язані між собою, щоб користувачі могли легко працювати з програмою та отримувати потрібні результати. Завдяки правильному вибору технологій і інструментів можна створити застосунок, який допоможе спростити процеси управління персоналом і зробить їх більш ефективними.

2.2 Розробка моделі системи

Для кращого розуміння роботи програмного додатку з оптимізації процесів управління персоналом було створено модель, яка описує основні етапи взаємодії користувача з системою (див. рисунок 2.1). Модель представлена у вигляді діаграми діяльності, що відображає послідовність дій від запуску програми до завершення роботи .

Спочатку користувач запускає програму та проходить авторизацію, вводячи свої дані. Система перевіряє, чи є користувач зареєстрованим, і якщо ні, пропонує пройти реєстрацію. У разі успішного входу користувач потрапляє на головний екран, де доступні основні функції.

Далі користувач обирає потрібну функцію, наприклад, перегляд робочих графіків або управління персоналом. Якщо обрано перегляд графіків, система відображає календар із розкладами, відпустками чи завданнями. Якщо ж обрано управління персоналом, адміністратор може додавати нових співробітників або налаштовувати їхні дані.

Після виконання потрібних дій користувач може завершити роботу, вибравши відповідну опцію. Система завершує сеанс і повертається до початкового

стану.

Розробка моделі системи була виконана шляхом аналізу всіх етапів роботи програми, що дозволило забезпечити її логічну структуру та ефективне функціонування.

Рисунок 2.1 – Діаграма діяльності

Розробка моделі системи була виконана з використанням інструментів для створення схем.

2.3 Розробка архітектури системи фінансового трекінгу

Як важливу складову програмного застосунку було розроблено архітектуру системи фінансового трекінгу, яка забезпечує відстеження доходів, витрат і ресурсів для бізнесів у сфері послуг, таких як салони краси чи медичні заклади. Архітектура базується на багатошаровому підході, що розмежовує інтерфейс,

логіку обробки даних і збереження інформації, забезпечуючи зручність і простоту оновлень. Система інтегрується з іншими модулями додатка, такими як календар (Routes.calendar) і авторизація, що створює цілісне рішення для користувачів.

Основні компоненти системи включають введення фінансових даних через зручний інтерфейс із текстовими полями, подібними до phone_input_field.dart, із перевіркою коректності введеної інформації. Обробка транзакцій відбувається через API-запити (api_login_calls_helper.dart), а локальне збереження реалізовано через SharedPreferences (name_shared_preferences.dart). Користувачі можуть переглядати зведення транзакцій і баланс на сторінці Routes.spendingIncome, визначеній у name_routes.dart. На рисунку 2.2 продемонстровано головну сторінку додатка, де відображено доступ до фінансових даних і поточний баланс.

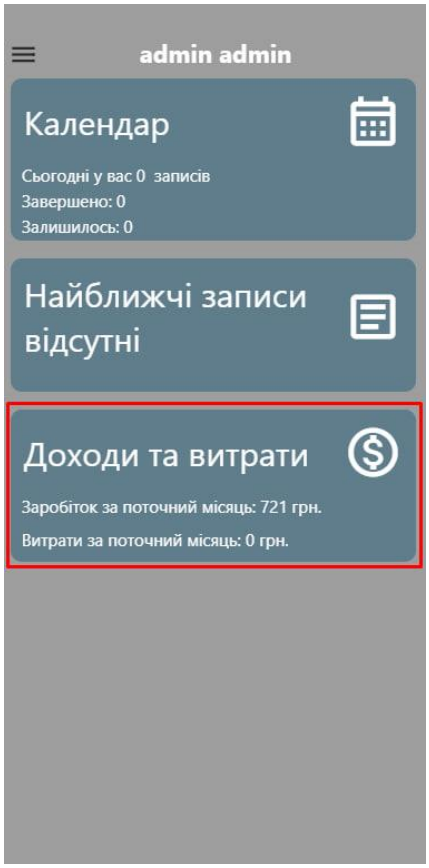


Рисунок 2.2 – Головна сторінка додатку

Також система дозволяє переглядати деталізовані звіти про доходи та витрати за обраний період, що відображається у зручному інтерфейсі. На рисунку

2.3 екран демонструє сторінку зведення фінансових даних, де користувач може вибрати період, переглянути загальний дохід і витрати, а також деталізацію транзакцій, таких як заробітна плата для адміністраторів.

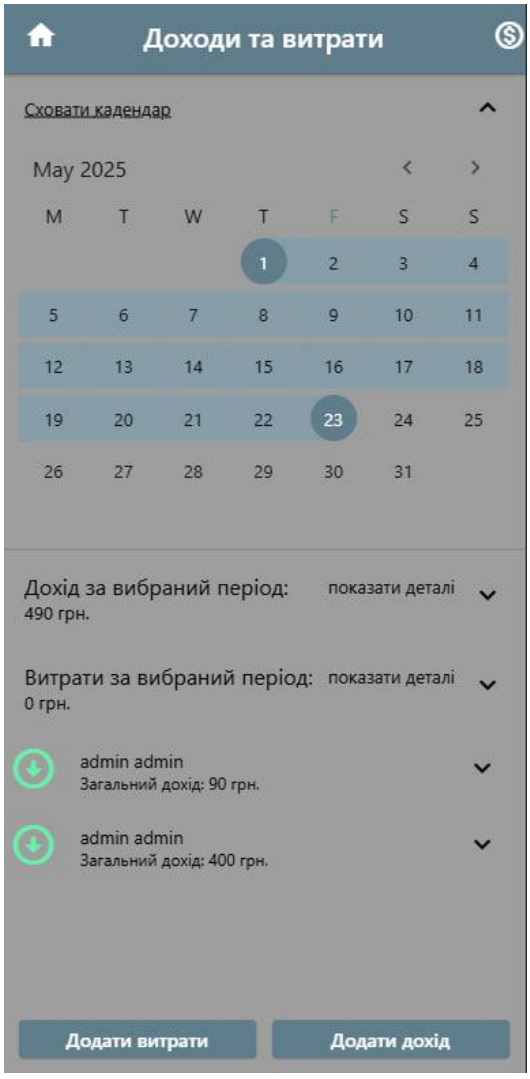


Рисунок 2.3 – Сторінка зведення фінансових даних

Крім того система дозволяє вводити доходи та витрати, враховуючи витратні матеріали (load_config.dart), із синхронізацією даних через API та сповіщеннями про операції через SnackBarHelper (snack_bar_helper.dart). Навігація між сторінками забезпечується через CustomRouteObserver (navigation_storage.dart), а управління станом реалізовано через Provider. На рисунку 2.4 продемонстровано приклад форми додавання доходу, де користувач може вказати суму, категорію та дату транзакції.

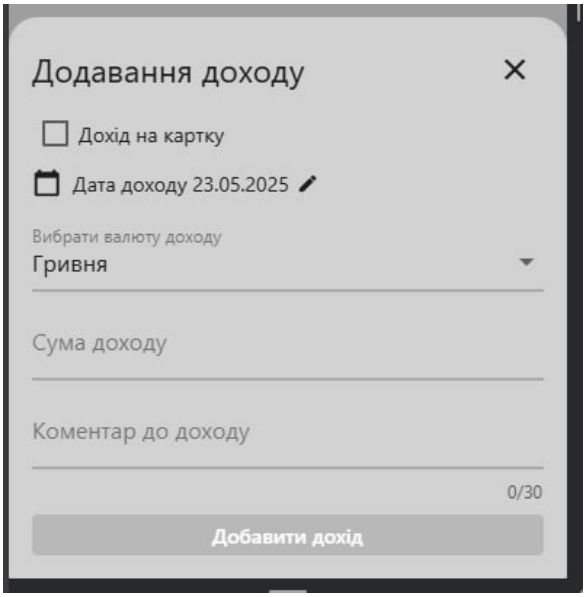


Рисунок 2.4 – Форма додавання доходу

Аналогічно працює форма для витрат, що дозволяє фіксувати витрати на матеріали та пов’язувати їх із процедурами. На рисунку 2.5 Цей інтерфейс наочно показує, як додаток спрощує облік витрат.

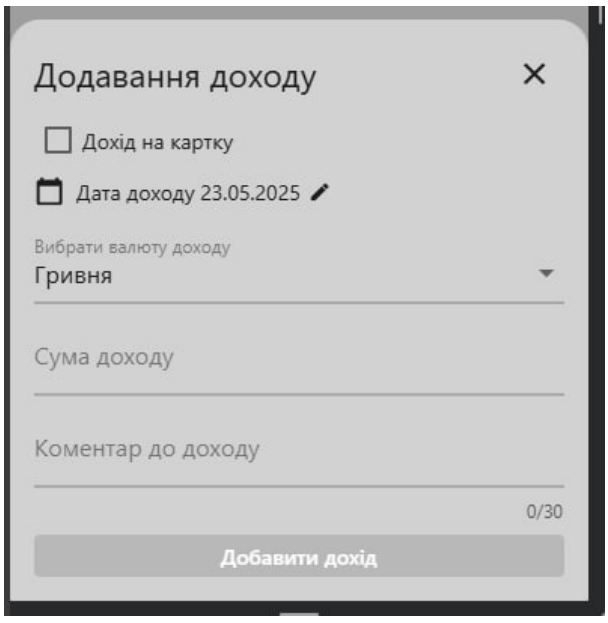


Рисунок 2.5 – Форма додавання витрат

Отже, розроблена архітектура системи фінансового трекінгу забезпечує ефективне управління доходами, витратами та ресурсами завдяки багатошаровій структурі, інтеграції з модулями календаря й авторизації та локальному кешуванню

даних із синхронізацією через API. Ця система пропонує гнучке й масштабоване рішення, яке підвищує зручність фінансового аналізу для користувачів.

2.4 Вдосконалення методу динамічного програмування

Динамічне програмування є одним із найпотужніших алгоритмічних підходів, який дозволяє ефективно вирішувати складні задачі шляхом їх декомпозиції на простіші підзадачі з подальшим збереженням отриманих результатів для повторного використання. Цей метод значно зменшує кількість обчислень, уникаючи повторного розв'язання однакових підзадач. Його широко застосовують у задачах оптимізації, теорії графів, обробці рядків та інших галузях програмування.

Динамічне програмування базується на принципах розв’язання задач, які мають властивості оптимальної підструктури та накладання підзадач. Оптимальна підструктура передбачає, що оптимальне рішення задачі можна отримати шляхом комбінації оптимальних рішень її підзадач. Накладання підзадач означає, що одні й ті ж підзадачі виникають неодноразово в процесі обчислень, що робить доцільним збереження їх результатів для уникнення надлишкових обчислень.

У рамках бакалаврської кваліфікаційної роботи було використано ітеративний підхід до реалізації динамічного програмування, відомий як метод «знизу вгору». Цей підхід передбачає поступове заповнення таблиці результатів, починаючи від найпростіших базових випадків і просуваючись до складніших підзадач. Основні етапи реалізації методу включають (див. рисунок 2.2):

1. Визначення стану, який характеризує поточну підзадачу.

2. Формулювання рекурентного співвідношення, що описує залежність рішення поточного стану від попередніх.

3. Ініціалізація базових випадків, які слугують відправною точкою для обчислень.

4. Заповнення таблиці результатів для отримання оптимального рішення.

У розробленому застосунку метод динамічного програмування було застосовано для вирішення задачі оптимізації розкладу клієнтів у системі

управління записами, яка є частиною програмного модуля календаря Routes.calendar.

У структурі додатка, описаній у файлах api_login_calls_helper.dart, navigation_storage.dart та name_shared_preferences.dart, реалізовано модуль управління записами клієнтів. Задача полягала в розробці методу, який оптимально розподіляє записи між майстрами, мінімізуючи часові простої, враховуючи тривалість процедур та доступність майстрів у певні часові (statConsumablesId у load_config.dart). Крім того, враховувалися пріоритети клієнтів, наприклад, термінові записи, які потребують негайного обслуговування.

Було розроблено метод динамічного програмування, який враховує такі параметри:

1. Тривалість і ціна кожної процедури, яка зберігається в структурі даних record (name_shared_preferences.dart).

2. Доступні часові слоти для кожного майстра, отримані через API-запити (api_login_calls_helper.dart).

Для демонстрації застосування методу динамічного програмування у межах розділу 2 було розроблено приклад його використання для задачі оптимального розподілу записів між майстрами з урахуванням тривалості процедур та їх вартості. У прикладі розглядається 3 працівники, які виконують такі процедури: масаж шиї - 45 хвилин (1 інтервал), 100 гривень; масаж ніг - 60 хвилин (2 інтервали), 250 гривень; масаж спини - 90 хвилин (2 інтервали), 350 гривень. Вартість процедур помножена на 3, що відповідає кількості працівників, і наведена в таблиці 2.1. Приклад включає один робочий день для працівників, тривалістю 6 годин, що складається з 8 інтервалів по 45 хвилин кожен.

Таблиця 2.1 – Оптимізація розкладу процедур

	1	2	3	4	5	6	7	9
Шия	300	600	900	1200	1500	1800	2100	2400
Ноги	300	750	1050	1350	1800	2250	2700	3000
Спина	300	1050	1350	2100	2550	3150	3600	**4200**

Максимальна сума 4200 гривень відповідає доходу від 4 процедур масажу спини, виконаних трьома майстрами за 8 інтервалів часу. Це демонструє ефективність розподілу робочого часу з урахуванням тривалості та вартості процедур.

Застосування методу динамічного програмування в розробленому програмного застосунку дозволило ефективно вирішити задачу оптимізації програмного застосунку, забезпечивши мінімальні простої та раціональне використання ресурсів. Інтеграція методу з модулями календаря, обробки API-запитів і збереження даних у SharedPreferences створила зручне та функціональне рішення для користувачів додатка.

2.5 Розробка алгоритмів роботи системи

Для створення програмного застосунку, призначеного для оптимізації процесів управління персоналом, було розроблено загальний алгоритм роботи системи, який забезпечує послідовне виконання ключових етапів: від авторизації користувача до взаємодії з модулями програми, такими як головна сторінка, сповіщення, налаштування та адміністративні функції. Алгоритм роботи системи включає авторизацію, відновлення стану після оновлення сторінки, навігацію між модулями та взаємодію з API для оновлення даних про персонал.

Загальний алгоритм роботи програмного застосунку для оптимізації процесів управління персоналом описує основні етапи взаємодії користувача з системою: перевірка авторизації, відновлення стану, обробка навігації та завершення роботи. Спочатку програма запускається, ініціалізуючи робоче середовище для взаємодії користувача з функціоналом застосунку. Далі завантажується робоче середовище програми, яке включає ініціалізацію маршрутів та конфігурацію. Перевіряється, чи користувач авторизований: якщо так, відновлюється попередній стан шляхом перенаправлення на останню відвідану сторінку, якщо ні – користувач переходить на сторінку входу. На сторінці входу або реєстрації користувач вводить дані, після чого перевіряється вибір пункту бічного меню: сповіщення, налаштування, адмін-панель або вихід. У разі вибору сповіщень відображається сторінка з

повідомленнями про низький запас витратних матеріалів, при виборі налаштувань – сторінка управління параметрами програми, а при виборі адмін-панелі – модуль для управління користувачами та ролями. Перед завершенням роботи програма зберігає стан, після чого завершує роботу.

Загальна робота програми забезпечує зручне управління персоналом шляхом автоматизації ключових процесів, таких як контроль витратних матеріалів, налаштування робочих параметрів та адміністрування користувачів, що дозволяє підвищити ефективність роботи менеджерів та зменшити час на рутинні операції.Блок-схеми алгоритма представлена на рисунку 2.6.

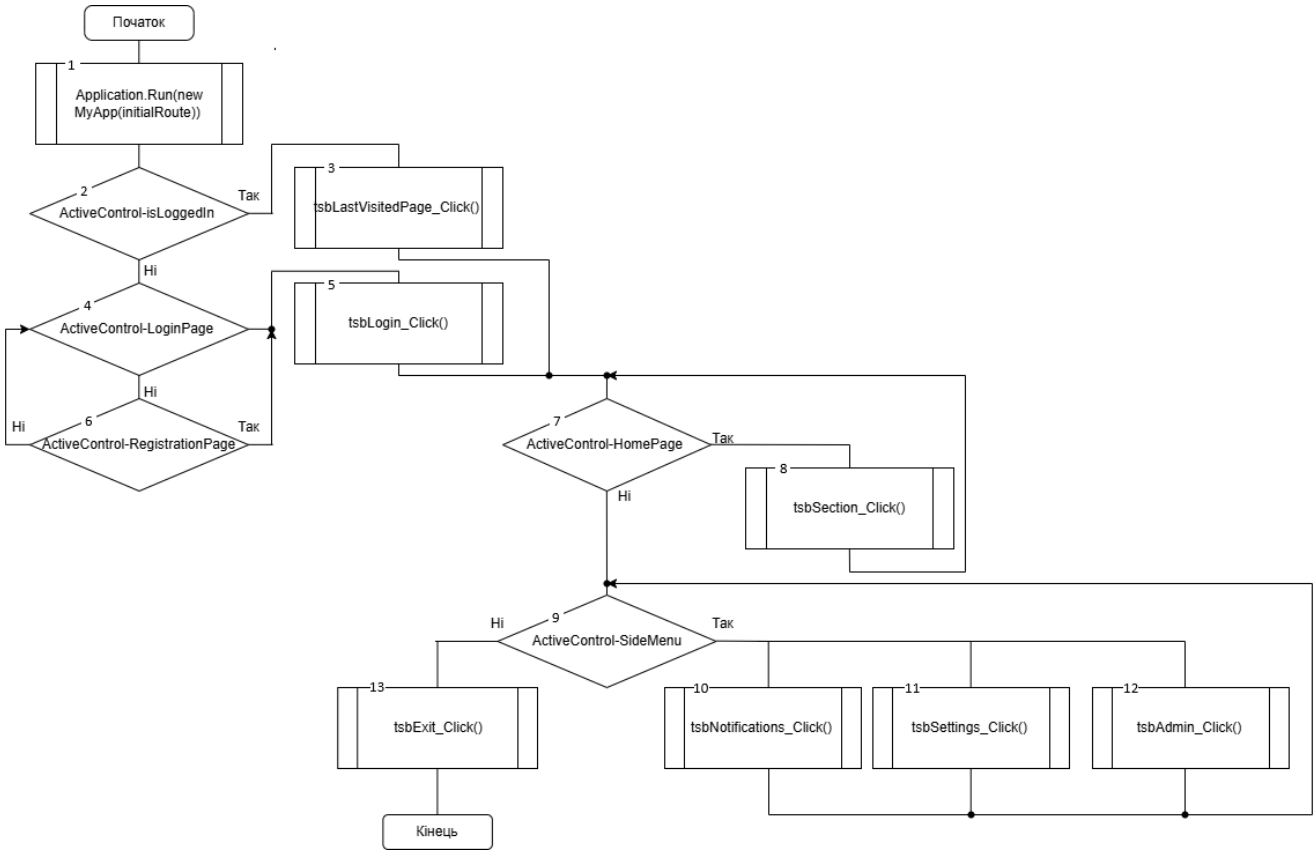


Рисунок 2.6 – Алгоритм роботи програмного застосунку

Алгоритм реєстрації та входу в застосунок описує процес авторизації користувача, який є першим етапом взаємодії з системою управління персоналом. Спочатку програма запускається, після чого завантажується форма для реєстрації або входу. Перевіряється, яку дію обрав користувач: зареєструватися чи увійти.

Якщо обрано реєстрацію, перевіряється валідність введених даних: у разі помилок відображається відповідне повідомлення, і користувач повертається до форми. Якщо дані валідні, перевіряється співпадіння паролів: при невідповідності виводиться повідомлення про помилку, і користувач повертається до форми. Далі перевіряється згода на обробку даних: якщо згода не надана, відображається повідомлення, і користувач повертається до форми. У разі згоди виконується API-запит для реєстрації, після чого перевіряється відповідь: при успішній реєстрації користувач перенаправляється на сторінку входу, а при помилках (наприклад, користувач уже зареєстрований, знайдено кілька записів або сталася помилка) відображається відповідне повідомлення, і користувач повертається до форми. Якщо обрано вхід, користувач одразу перенаправляється на сторінку входу, після чого процес завершується.

Завдяки продуманій структурі алгоритму, користувач отримує чітке й послідовне керування своїми діями в інтерфейсі, що мінімізує ймовірність помилок та підвищує загальну зручність взаємодії із системою. Це особливо важливо в контексті управління персоналом, де точність і безпека даних мають критичне значення.

Ці алгоритми забезпечують ефективну роботу системи управління персоналом, дозволяючи користувачам авторизуватися, відновлювати стан після оновлення сторінки та переходити між модулями програми для управління даними співробітників.

2.6 Висновки

У другому розділі було виконано детальне дослідження всіх аспектів, необхідних для створення програмного застосунку, спрямованого на оптимізацію процесів управління персоналом. Зокрема, проведено аналіз даних, які надходять у систему, що допомогло визначити вимоги до їх обробки та зберігання. Окрім того, розроблено модель системи, що відображає її структуру та основні етапи роботи, а також створено алгоритми, які визначають логіку функціонування застосунку, включаючи авторизацію, навігацію та обробку даних.

Усе це дозволило сформувати чітке розуміння того, як повинна працювати система. Це стало основою для подальшої реалізації функціоналу та перевірки його ефективності.

3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ЗАСТОСУНКУ ОПТИМІЗАЦІЇ УПРАВЛІННЯ ПЕРСОНАЛОМ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

При розробці програмного застосунку для оптимізації процесів управління персоналом ключовим завданням було обрати технології, які забезпечують ефективність, безпеку та зручність як для розробників, так і для користувачів. Вибір інструментів ґрунтувався на їх здатності вирішувати задачі управління даними, забезпечувати зручний інтерфейс і гарантувати безпеку. Серверна частина створена з використанням мови Python у середовищі Visual Studio Code, а клієнтська – на мові Dart із фреймворком Flutter у середовищі Android Studio, що разом створюють комплексне рішення для управління персоналом.

Python обрано для серверної частини через його простоту, гнучкість і широкий набір бібліотек, які дозволяють ефективно працювати з базами даних, обробляти запити та створювати API. Розробка велася в Visual Studio Code, яке забезпечує зручне середовище завдяки підтримці розширень для налагодження, автодоповнення коду та інтеграції з системами контролю версій. Для створення restful API використано бібліотеку Flask, яка є легким фреймворком для обробки HTTP-запитів. Flask дозволяє швидко налаштувати маршрути для авторизації, реєстрації та роботи з даними, що робить її ідеальним вибором для серверної частини програми. Для роботи з базою даних MySQL застосована бібліотека mysql-connector-python, яка забезпечує стабільне з’єднання та виконання SQL-запитів. MySQL обрано через її швидкість, надійність і поширеність для роботи зі структурованими даними, такими як інформація про користувачів, записи та витратні матеріали.

Безпека даних користувачів є пріоритетом, тому для хешування паролів використано бібліотеку bcrypt. Вона забезпечує надійний захист паролів завдяки стійкому алгоритму хешування, що запобігає несанкціонованому доступу. Бібліотека json використовується для обробки та передачі даних у форматі JSON,

що є стандартом для обміну інформацією між сервером і клієнтом, забезпечуючи швидкий і зручний обмін даними. Для моніторингу роботи системи застосована бібліотека logging, яка веде журнал подій, що дозволяє швидко виявляти та усувати проблеми в роботі програми.

Клієнтська частина розроблена на мові Dart із фреймворком Flutter у середовищі Android Studio, що є оптимальним рішенням для створення кросплатформених мобільних застосунків. Flutter дозволяє створювати швидкі та адаптивні інтерфейси, які працюють на Android і iOS, зменшуючи час розробки. Android Studio забезпечує зручне середовище для роботи з Flutter, надаючи інструменти для дизайну інтерфейсу, тестування та налагодження. Для збереження локальних даних, таких як стан авторизації чи налаштування, використовується бібліотека shared_preferences, яка дозволяє зберігати дані у форматі ключ-значення та відновлювати їх після перезапуску програми. Бібліотека image_picker забезпечує можливість роботи з медіафайлами, дозволяючи користувачам завантажувати зображення, що є важливим для модулів фотографування. Для управління станом програми застосована бібліотека provider, яка забезпечує реактивне оновлення даних, наприклад, при зміні стану секундоміра чи сповіщень .

У програмі реалізовано кілька ключових алгоритмів для забезпечення її функціональності. Алгоритми авторизації та реєстрації перевіряють унікальність логіна, хешують паролі за допомогою bcrypt і обробляють помилки, такі як невідповідність паролів чи наявність кількох записів у базі даних. Алгоритми роботи з секундоміром обчислюють тривалість сесії, визначають її вартість на основі тарифів і зберігають стан для відновлення після перезапуску програми. Алгоритми обробки сповіщень аналізують рівень витратних матеріалів і автоматично повідомляють користувача про їх низький запас, що допомагає вчасно реагувати на потреби [19-23].

Поєднання Python із бібліотеками Flask, mysql-connector-python, bcrypt, json і logging на сервері та Dart із Flutter, shared_preferences, image_picker і provider на клієнтській стороні дозволяє створити ефективний програмний засіб для управління персоналом. Python і його бібліотеки забезпечують надійну обробку

даних і безпечну авторизацію, тоді як Flutter із бібліотеками створює зручний і адаптивний інтерфейс користувача. Реалізовані алгоритми забезпечують швидку та точну обробку даних, що є важливим для оптимізації процесів управління персоналом.

Отже, вибір технологій для розробки програмного забезпечення для управління персоналом із використанням Python, Flask, MySQL, Dart, Flutter та відповідних бібліотек є обґрунтованим рішенням. Поєднання цих інструментів забезпечує комплексний підхід до розробки, дозволяючи створити продукт із високим рівнем функціональності, безпеки та зручності використання, що відповідає сучасним вимогам до систем управління персоналом.

3.2 Розробка інтерфейсу програмного застосунку

При створенні інтерфейсу програмного застосунку для оптимізації процесів управління персоналом особливу увагу було приділено його простоті та зручності для користувачів. Основним кольором інтерфейсу обрано сіро-блакитну палітру, оскільки ці відтінки асоціюються з професійністю та технологічністю, що відповідає тематиці управління персоналом. Допоміжними кольорами стали білий і темно-сірий, які забезпечують контрастність і допомагають виділити ключові елементи інтерфейсу.

Усі елементи інтерфейсу розроблено з урахуванням принципів мінімалізму, щоб забезпечити інтуїтивно зрозумілу навігацію. При запуску програми користувач спочатку потрапляє на екран авторизації (див. рисунок 3.1), де відображається назва системи та основні поля для входу. Цей екран створений для швидкого та зрозумілого доступу до програми. Для підвищення зручності екран авторизації містить функцію автоматичного заповнення даних, що зменшує час на вхід для регулярних користувачів. Додатково передбачено опцію «Забули пароль» для швидкого відновлення доступу. Також інтерфейс адаптовано під різні роздільні здатності екрану, що забезпечує комфортну та безперебійну роботу на різних пристроях.

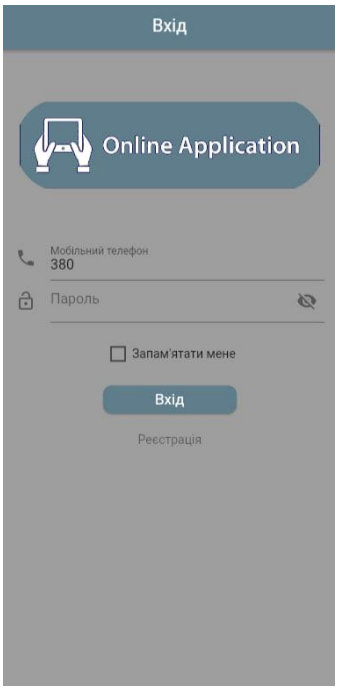


Рисунок 3.1 – Екран авторизації

Також було створено екран реєстрації (див. рисунок 3.2), де користувачі можуть ввести необхідні дані для створення облікового запису, що забезпечує легкий доступ до системи для нових користувачів.

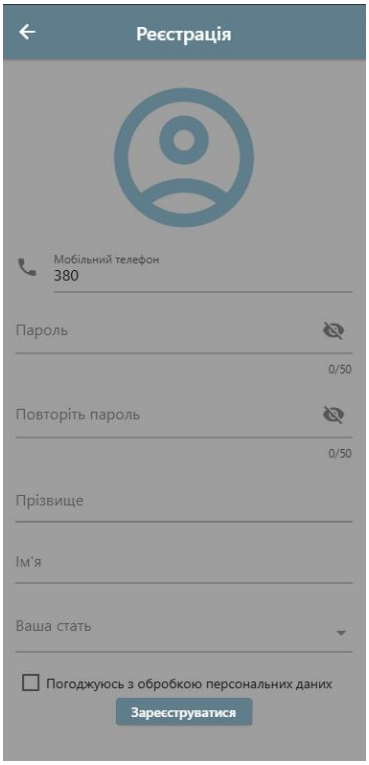


Рисунок 3.2 – Екран реєстрації

Для зручної роботи з користувачем було розроблено основне меню програми (див. рисунок 3.3), яке дозволяє швидко орієнтуватися між різними модулями системи, такими як облік співробітників, автоматизація кадрових процесів та аналітика продуктивності. Основне меню створено з урахуванням принципів інтуїтивної навігації, що забезпечує легкий доступ до всіх ключових функцій системи для нового користувача.

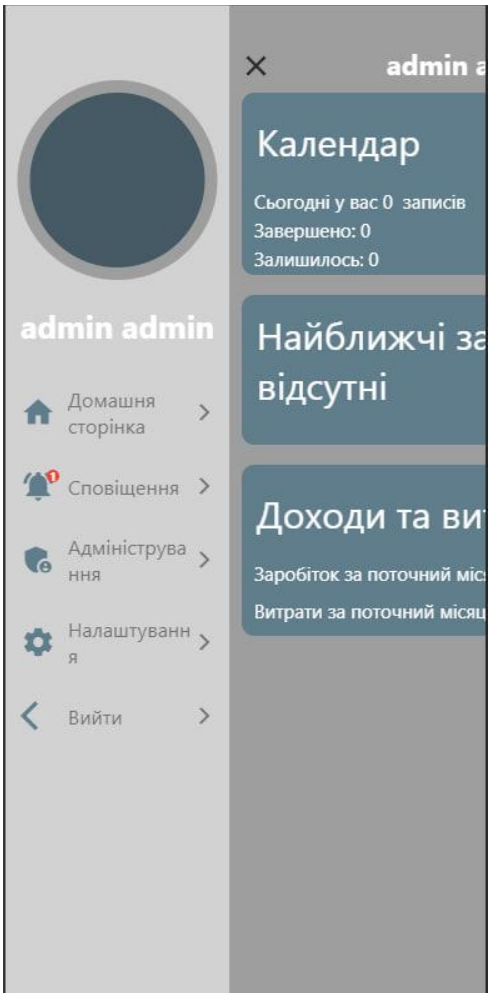


Рисунок 3.3 – Основне меню програми

Після входу в систему користувач потрапляє на головний екран, де відображається ключова інформація, така як записи, графік роботи та фінансові дані. Цей екран створений для того, щоб користувач одразу отримував загальний огляд системи. Це дозволяє швидко переглядати актуальні дані та переходити до потрібного розділу (див. рисунок 3.4).

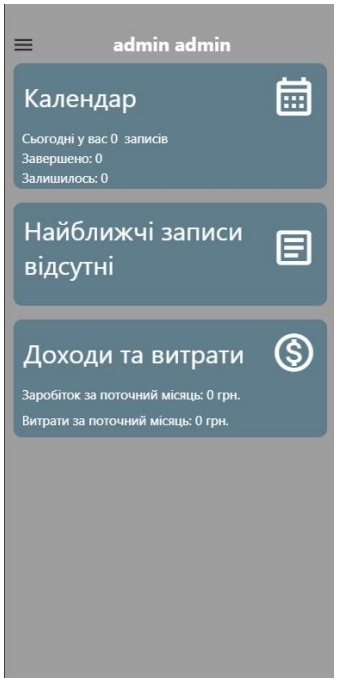


Рисунок 3.4 – Головний екран програми

Для детальнішого управління доступний екран налаштувань, де можна змінювати параметри роботи програми, переглядати доступні опції та адаптувати систему під свої потреби (див. рисунок 3.5).

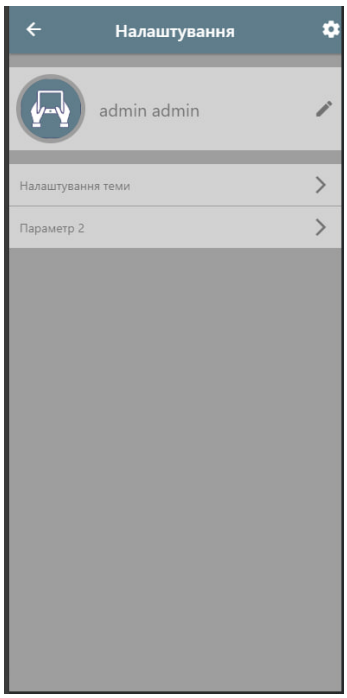


Рисунок 3.5 – Екран налаштувань

Додатково було розроблено екран сповіщень (див. рисунок 3.6), який дозволяє користувачам отримувати важливі повідомлення, наприклад про повідомлення, що закінчуються розхідники. Цей розділ допомагає бути в курсі всіх подій, пов’язаних із управлінням персоналом.

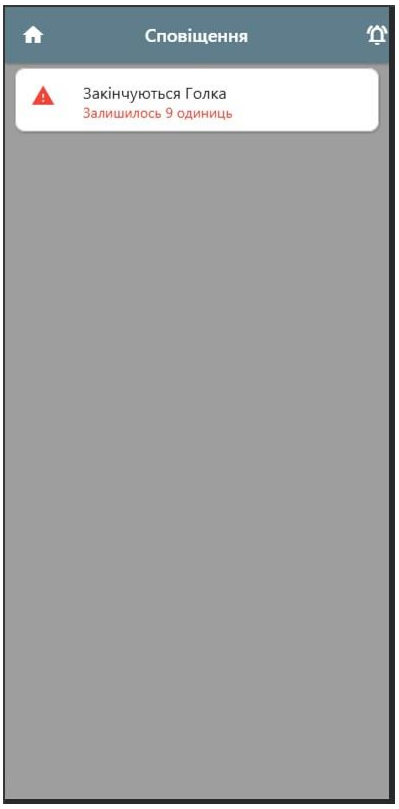


Рисунок 3.6 – Головний екран програми

У програмі також передбачено екран адміністратора (див. рисунок 3.7), який відкриває доступ до розширених функцій управління. На екрані представлено бічне меню з кількома опціями, такими як «Зміна ролі користувача», «Налаштування правил», «Налаштування цін», «Додавання користувача» та «Управління розсилками». Кожна з цих опцій дозволяє адміністратору виконувати специфічні завдання: змінювати права доступу для користувачів, налаштовувати правила роботи системи, встановлювати фінансові параметри, додавати нових користувачів до системи або керувати розсилками для інформування персоналу. Цей екран створений для забезпечення повного контролю над системою та її користувачами, що є важливим для ефективного управління персоналом.

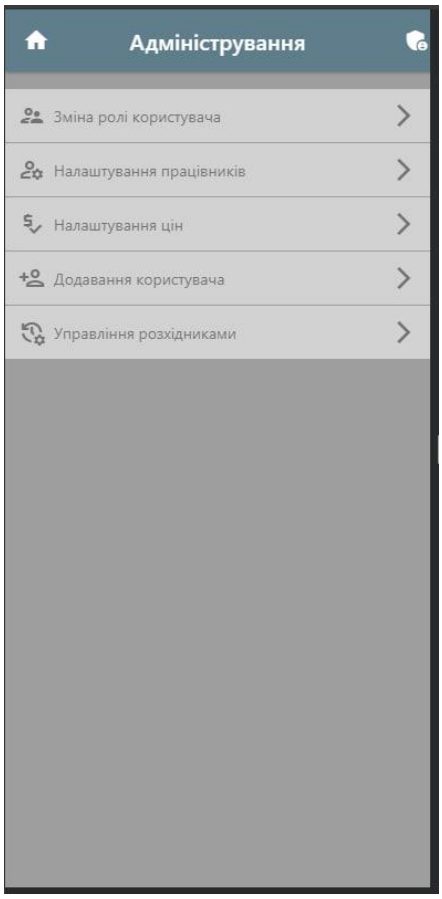


Рисунок 3.7 – Екран адміністратора

Розробка графічного інтерфейсу програмного додатку була проведена у середовищі Android Studio з використанням мови програмування Dart, що дозволило створити адаптивний і зручний інтерфейс для користувачів.

3.3 Розробка модуля реєстрації та авторизації користувача

При розробці модуля реєстрації та авторизації користувача для програмного застосунку з оптимізації процесів управління персоналом необхідно враховувати ключові аспекти безпеки, зручності та коректності обробки даних. Основною задачею модуля є забезпечення можливості реєстрації нових користувачів, авторизації в системі, валідації введених даних та збереження стану авторизації для подальшого використання.

Для цього було створено серверні класи у файлах authorizationUser.py та registrationUser.py, які обробляють запити на авторизацію та реєстрацію через REST API, створені за допомогою Flask. На клієнтській стороні у файлах

login_page.dart та registration_page.dart реалізовано інтерфейси для введення даних користувачем, а також їх валідацію перед відправленням на сервер. У файлі connectionDatabase.py забезпечено підключення до бази даних MySQL через бібліотеку mysql-connector-python, що дозволяє зберігати та отримувати дані користувачів.

Для забезпечення доступу до системи першим кроком є створення облікового запису. Функція реєстрації дозволяє користувачу створити новий обліковий запис у системі. На клієнтській стороні користувач вводить персональні дані через форму, яка включає валідацію та перевірку збігу паролів. Дані надсилаються на сервер, де перевіряється унікальність номера телефону. Пароль хешується за допомогою бібліотеки bcrypt, після чого дані зберігаються в базі MySQL. У разі успішної реєстрації користувач перенаправляється на сторінку входу, зображено на рисунку 3.8.

```
def register(self, userName, userPass, lastName, firstName, gender, mobilephone, procPersonalData):
    connect = self._create_connection()
    database = DatabaseHelper(connect)

    users, columnName, rowtype = database.selectForRegisterUser(userName=userName)

    if len(users) == 0:
        print("Користувач не знайдений")
        salt = bcrypt.gensalt()
        hashed_password = bcrypt.hashpw(userPass.encode('utf-8'), salt)

        result = database.insertRegisterUser(userName, hashed_password, salt, lastName, firstName, gender, mobilephone, procPersonalData)
        data = {
            "Status": result
        }
        connect.close()
        jsonString = json.dumps(data)
        return jsonString

    elif len(users) == 1:
        print("Користувач з таким логіном вже зареєстрований!!!")
        data = {
            "Status": -1
        }
        connect.close()
        jsonString = json.dumps(data)
        return jsonString
```

Рисунок 3.8 – Функція реєстрації нового користувача

Після реєстрації користувач може увійти в систему. Функція авторизації забезпечує вхід користувача в систему шляхом перевірки введених даних. На клієнтській стороні користувач вводить номер телефону та пароль, які відправляються на сервер через REST API. На сервері за допомогою bcrypt перевіряється відповідність введеного пароля збереженому хешу в базі даних. У

разі успішної авторизації користувач отримує доступ до системи, а його дані завантажуються для подальшої роботи.

Щоб забезпечити коректність введених даних, реалізована функція валідації. Вона відповідає за перевірку коректності введених даних перед їх відправленням на сервер. На клієнтській стороні валідація включає перевірку заповнення обов’язкових полів, наприклад, пароля, який має бути не коротшим за 6 символів, а також збіг паролів під час реєстрації. На сервері додатково перевіряється унікальність номера телефону. Ця функція підвищує надійність системи, запобігаючи помилкам при введенні даних. Крім того, валідація допомагає уникнути дублювання облікових записів, що забезпечує цілісність даних у системі. Також вона включає перевірку формату номера телефону, щоб відповідати міжнародним стандартам. Це дозволяє користувачам уникнути типових помилок і забезпечує безперебійну роботу програми.

Для підвищення зручності користувача додано функцію збереження стану авторизації. Вона дозволяє зберігати дані авторизації локально, якщо користувач обирає опцію «Запам’ятати мене».

Дані, такі як номер телефону та пароль, зберігаються за допомогою бібліотеки shared_preferences, що дозволяє автоматично заповнювати поля при наступному вході, зменшуючи час на авторизацію.

Таким чином, модуль реєстрації та авторизації користувача забезпечує повноцінну роботу з обліковими записами, включаючи їх створення, авторизацію, валідацію даних і збереження стану входу, що дозволяє користувачам безпечно та зручно взаємодіяти з системою управління персоналом.

3.4 Розробка модуля збору інформації про виконання завдань

Модуль управління завданнями працівників у програмному застосунку з оптимізації процесів управління персоналом розроблено для забезпечення ефективного розподілу задач, моніторингу їх виконання та фінансового аналізу діяльності працівників.

Основною задачею модуля є надання адміністраторам і працівникам інструментів для налаштування параметрів роботи, перегляду доходів і витрат, а також оцінки продуктивності на основі виконаних завдань, що сприяє підвищенню прозорості та ефективності управління персоналом. Для реалізації цього модуля використано клієнтську частину на Flutter, яка забезпечує зручний інтерфейс, а також інтеграцію з сервером через REST API для обробки даних, що гарантує актуальність і синхронізацію інформації. Ключові функції модуля реалізовано у файлах employee_settings.dart, employee_salary.dart та spending_income.dart, а маршрутизацію між сторінками забезпечує файл main.dart.

Для початку роботи з модулем адміністратору необхідно налаштувати параметри працівників. Функція налаштування параметрів працівників дозволяє встановлювати індивідуальні фінансові параметри для кожного співробітника, такі як вартість сеансу за хвилину та відсоток від вартості сеансу, що забезпечує справедливий розподіл доходів. У файлі employee_settings.dart реалізовано список працівників із можливістю пошуку за ім’ям, а також форму для редагування їхніх параметрів, де зміни зберігаються через API-запити до сервера, що дозволяє оперативно вносити оновлення. Ця функція забезпечує персоналізований підхід до управління персоналом, сприяючи підвищенню мотивації працівників через прозорий механізм нарахування винагороди (див. рисунок 3.9).

```
/// кнопка зберегти зміни
ElevatedButton(
  style: ElevatedButton.styleFrom(
      backgroundColor: AppColors.themColor
  ),
  onPressed: isDataChanged ?  () {
    if (_formKey.currentState?.validate() ?? false) {
      setState(() {
        _restApiBD.updateEmpolCost(selectedUser['user_id'].toString(), _costMinuteEmp.text, _percentEmp.text).then((DataUser){
          context.read<DataProvider>().updateAllUsers(json.decode(DataUser));
          userList = dataModel.allUsers;
          filteredUserList = dataModel.allUsers;
          selectedUser = {};
        });
      });
    }
  } : null,
  child: Text('Зберегти зміни', style: TextStyle(color: AppColors.textColor),),
),  // ElevatedButton
```

Рисунок 3.9 – Функція налаштування параметрів працівників

Після налаштування параметрів адміністратор може аналізувати фінансовий внесок працівників. Функція аналізу заробітку працівника дозволяє відстежувати дохід кожного співробітника за вибраний період, що допомагає оцінити їхню продуктивність і ефективність.

У файлі employee_salary.dart реалізовано календар для вибору діапазону дат, після чого система підраховує загальний дохід працівника на основі завершених сеансів, враховуючи витрати на розхідники, такі як голки та анестезія, а також відсоток працівника від вартості сеансу, що забезпечує точний і прозорий розрахунок. Список сеансів відображається з деталями, включаючи дату, загальну вартість, чистий дохід працівника та роботодавця, що дозволяє адміністратору аналізувати розподіл доходів.

Для повного контролю над фінансами, пов’язаними з діяльністю працівників, у модулі передбачено функцію управління доходами та витратами. Ця функція дозволяє адміністратору відстежувати фінансові операції, включаючи доходи від сеансів і витрати на різні потреби, що сприяє оптимізації фінансових ресурсів.

У файлі spending_income.dart реалізовано календар для вибору періоду, який дозволяє адміністратору легко відстежувати фінансові операції за конкретний часовий проміжок, а також список транзакцій із можливістю додавання, редагування та видалення записів про доходи чи витрати. Функція підтримує різні валюти і дозволяє вказувати спосіб оплати, що забезпечує гнучкість у роботі з міжнародними та локальними фінансовими операціями. Ця функціональність дає повний контроль над фінансовими потоками, допомагаючи адміністратору приймати обґрунтовані рішення щодо розподілу ресурсів, планування витрат і прогнозування бюджетних потреб підприємства.

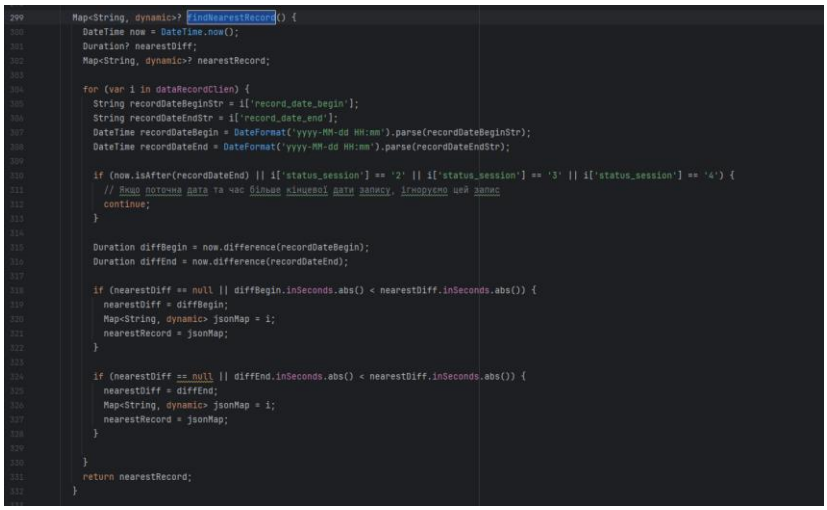
Таким чином, модуль збору інформації про виконання завдання забезпечує комплексний підхід до адміністрування персоналу, поєднуючи налаштування індивідуальних параметрів, таких як робочий час і продуктивність, аналіз заробітної плати з урахуванням відпрацьованих годин, а також контроль фінансових потоків через інтегрований модуль транзакцій. Такий підхід сприяє підвищенню ефективності роботи завдяки автоматизації рутинних завдань,

забезпечує прозорість у розподілі ресурсів шляхом детального звітування та мотивує працівників через чітке відображення їхнього внеску в загальний результат. Крім того, інтеграція з календарем дозволяє адаптувати систему до різних умов, що є особливо цінним для компаній із розгалуженою структурою.

3.5 Розробка модуля головного та бічного меню

Модуль головного та бічного меню у програмному застосунку з оптимізації процесів управління персоналом розроблено для забезпечення зручної навігації та доступу до ключових функцій системи. Основною задачею модуля є надання користувачам інтуїтивно зрозумілого інтерфейсу для переходу між різними сторінками, такими як календар, картка клієнта, налаштування, адміністрування тощо, залежно від ролі користувача. Модуль реалізований на Flutter, що забезпечує кросплатформений досвід, а для управління станом та передачі даних використано бібліотеку Provider. Головне меню відображається на сторінці home_page.dart, бічне меню інтегровано за допомогою пакету flutter_slider_drawer, а функціонал адміністрування доступний через admin_module.dart. Нижче описано ключові функції модуля [24].

Для відображення основної інформації та швидкого доступу до активних завдань на головній сторінці реалізовано функцію пошуку найближчого запису. Функція findNearestRecord у файлі home_page.dart визначає найближчий активний запис користувача, порівнюючи поточний час із часом початку та закінчення записів, що зберігаються у списку dataRecordClien. Вона фільтрує завершені записи та ті, що мають статус відмінених чи виконаних, і повертає найближчий запис, який відображається у блоці «Картка клієнта». Це дозволяє користувачу швидко переглянути деталі, такі як ім’я клієнта, час і зона запису, та перейти до повної інформації про клієнта (див. рисунок 3.10). Функція також підтримує оновлення даних у реальному часі через API-запити, що забезпечує актуальність інформації. Для зручності користувача додано можливість прямого переходу до редагування запису з блоку «Картка клієнта».

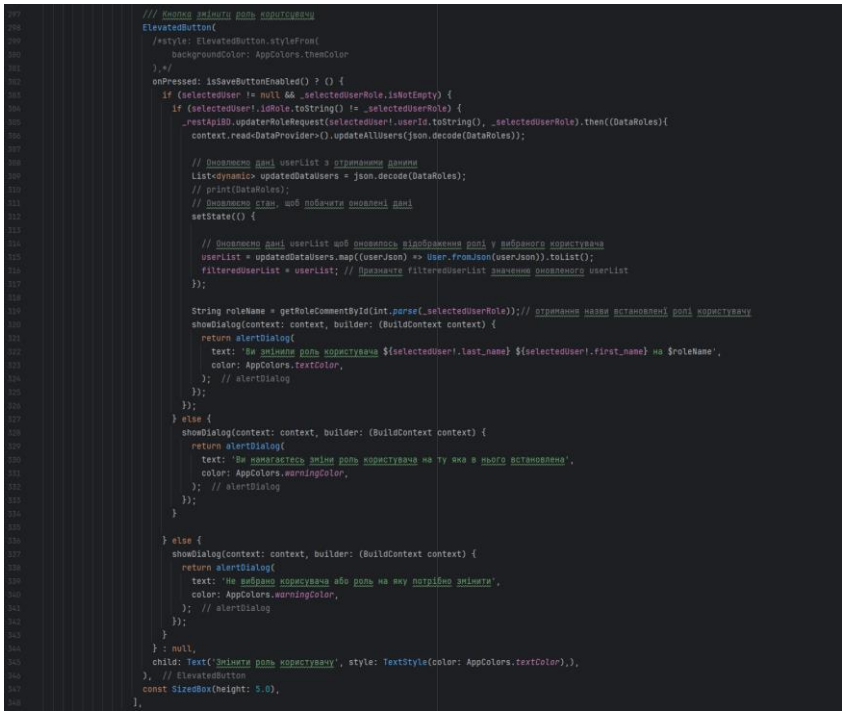


```
Map<String, dynamic>? findNearestRecord() {
  DateTime now = DateTime.now();
  Duration? nearestDiff;
  Map<String, dynamic>? nearestRecord;

  for (var i in dataRecordClien) {
    String recordDateBeginStr = i['record_date_begin'];
    String recordDateEndStr = i['record_date_end'];
    DateTime recordDateBegin = DateFormat('yyyy-MM-dd HH:mm').parse(recordDateBeginStr);
    DateTime recordDateEnd = DateFormat('yyyy-MM-dd HH:mm').parse(recordDateEndStr);

    if (now.isAfter(recordDateEnd) || i['status_session'] == '2' || i['status_session'] == '3' || i['status_session'] == '4') {
      // Якщо поточна дата та час більше кінцевої дати запису, ігноруємо цей запис
      continue;
    }

    Duration diffBegin = now.difference(recordDateBegin);
    Duration diffEnd = now.difference(recordDateEnd);

    if (nearestDiff == null || diffBegin.inSeconds.abs() < nearestDiff.inSeconds.abs()) {
      nearestDiff = diffBegin;
      Map<String, dynamic> jsonMap = i;
      nearestRecord = jsonMap;
    }

    if (nearestDiff == null || diffEnd.inSeconds.abs() < nearestDiff.inSeconds.abs()) {
      nearestDiff = diffEnd;
      Map<String, dynamic> jsonMap = i;
      nearestRecord = jsonMap;
    }

  }
  return nearestRecord;
}
```

Рисунок 3.10 – Функція пошуку найближчого запису

Для управління профілем користувача з бічного меню реалізовано функцію редагування профілю, доступну через сторінку налаштувань. Функція у файлі edit_profile.dart дозволяє користувачу оновлювати особисті дані, такі як ім’я, прізвище, стать і номер телефону, а також змінювати пароль через діалогове вікно PasswordConfirmationDialog. Дані валіюються перед відправленням на сервер через API-запит, а оновлення відображаються у реальному часі завдяки Provider. Ця функція підвищує зручність користувача, дозволяючи швидко адаптувати профіль до змін.

Для адміністраторів через бічне меню доступна функція зміни ролі користувача. Функція у файлі change_role.dart дозволяє адміністратору вибрати користувача зі списку, відфільтрованого за ім’ям, і призначити нову роль через випадаючий список, створений на основі даних із dataRols. Після вибору ролі зміни зберігаються через API-запит, а оновлений список користувачів відображається у реальному часі. Функція забезпечує гнучке управління доступом, виключаючи можливість зміни ролі авторизованого користувача (див. рисунок 3.11). Для підвищення безпеки система записує всі зміни ролей у журнал подій, що дозволяє відстежувати дії адміністраторів. Крім того, передбачено повідомлення про успішне оновлення ролі, що підтверджує коректність виконаної операції.



```
/// Кнопка змінити роль користувачу
ElevatedButton(
  /*style: ElevatedButton.styleFrom(
      backgroundColor: AppColors.themColor
  ),*/
  onPressed: isSaveButtonEnabled() ? () {
    if (selectedUser != null && _selectedUserRole.isNotEmpty) {
      if (selectedUser!.idRole.toString() != _selectedUserRole) {
        _restApiBD.updaterRoleRequest(selectedUser!.userId.toString(), _selectedUserRole).then((DataRoles){
          context.read<DataProvider>().updateAllUsers(json.decode(DataRoles));

          // Оновлюємо дані userList з отриманими даними
          List<dynamic> updatedDataUsers = json.decode(DataRoles);
          // print(DataRoles);
          // Оновлюємо стан, щоб побачити оновлені дані
          setState(() {

            // Оновлюємо дані userList щоб оновилось відображення ролі у вибраного користувача
            userList = updatedDataUsers.map((userJson) => User.fromJson(userJson)).toList();
            filteredUserList = userList; // Призначте filteredUserList значення оновленого userList
          });

          String roleName = getRoleCommentById(int.parse(_selectedUserRole));// отримання назви встановленої ролі користувачу
          showDialog(context: context, builder: (BuildContext context) {
            return alertDialog(
              text: 'Ви змінили роль користувача ${selectedUser!.last_name} ${selectedUser!.first_name} на $roleName',
              color: AppColors.textColor,
            );  // alertDialog
          });
        });
      } else {
        showDialog(context: context, builder: (BuildContext context) {
          return alertDialog(
            text: 'Ви намагаєтесь змінити роль користувача на ту яка в нього встановлена',
            color: AppColors.warningColor,
          );  // alertDialog
        });
      }

    } else {
      showDialog(context: context, builder: (BuildContext context) {
        return alertDialog(
          text: 'Не вибрано користувача або роль на яку потрібно змінити',
          color: AppColors.warningColor,
        );  // alertDialog
      });
    }
  } : null,
  child: Text('Змінити роль користувачу', style: TextStyle(color: AppColors.textColor),),
),  // ElevatedButton
const SizedBox(height: 5.0),
],
```

Рисунок 3.11 – Функція зміни ролі користувача

Таким чином, модуль головного та бічного меню забезпечує зручну навігацію, швидкий доступ до найближчих записів, редагування профілю та управління ролями, що підвищує ефективність роботи користувачів і адаптивність системи до їхніх потреб.

3.6 Висновки

У третьому розділі було обґрунтовано вибір технологій, таких як Python із Flask, MySQL, Dart із Flutter, а також бібліотеки bcrypt, provider і shared_preferences, що забезпечують ефективну розробку програмного застосунку для управління персоналом, поєднуючи безпеку, кросплатформність і зручність інтерфейсу. Розроблені модулі реєстрації, авторизації, управління завданнями, головного та бічного меню, реалізовані у файлах authorizationUser.py, registrationUser.py, home_page.dart, edit_profile.dart та інших, дозволяють користувачам безпечно авторизуватися, ефективно управляти завданнями, швидко переглядати записи та налаштовувати профіль, що підвищує продуктивність і зручність роботи з системою.

4 ТЕСТУВАННЯ І ПРАКТИЧНЕ ЗАСТОСУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Тестування системи

Тестування програмного застосунку з оптимізації процесів управління персоналом було виконано для гарантування його надійності, якості та відповідності заявленим вимогам. Основна мета полягала у виявленні недоліків, перевірці стабільності роботи та оцінці зручності використання перед запуском системи в експлуатацію.

Процес розпочався з встановлення тестової версії програми на пристроях з операційними системами Windows, а також підготовки плану тестування та документації з вимогами. Тестування охоплювало кілька етапів: перевірку функціональності, стійкості до збоїв, швидкості роботи та сумісності системи.

Функціональне тестування зосереджувалося на перевірці коректності виконання основних операцій, таких як створення облікового запису та вхід у систему. Тестувалися сценарії з коректними та некоректними даними, наприклад, введення занадто короткого пароля чи спроба реєстрації з уже існуючим номером телефону, а також правильність фінансових розрахунків у модулі аналізу доходів.

Наступним етапом стала перевірка стійкості до збоїв. Зокрема, перевірялася реакція програми під час пошуку активних записів, щоб переконатися, що вона здатна коректно обробляти помилки та продовжувати роботу.

Останнім етапом стало тестування зручності використання (usability testing), яке оцінювало інтуїтивність інтерфейсу та зручність взаємодії з програмою для кінцевих користувачів. Особливу увагу приділили логічності розташування елементів, швидкості доступу до основних функцій і зрозумілості повідомлень про помилки, щоб забезпечити комфортний досвід для користувачів із різним рівнем технічної підготовки [25-26].

Першим ділом було здійснено функціональне тестування, під час тестування перевірялася коректність авторизації: при введенні правильного номера телефону та пароля користувач успішно входив у систему, а при введенні некоректних даних відображалося повідомлення про помилку. Наприклад, тестування показало, що

при введенні неіснуючого номера телефону або неправильного пароля система видає попередження, що користувача з таким номером телефону не зареєстровано, як зазначено в повідомленні «Користувача з таким номером телефону ще не зареєстровано!!!» (див. рисунок 4.1).

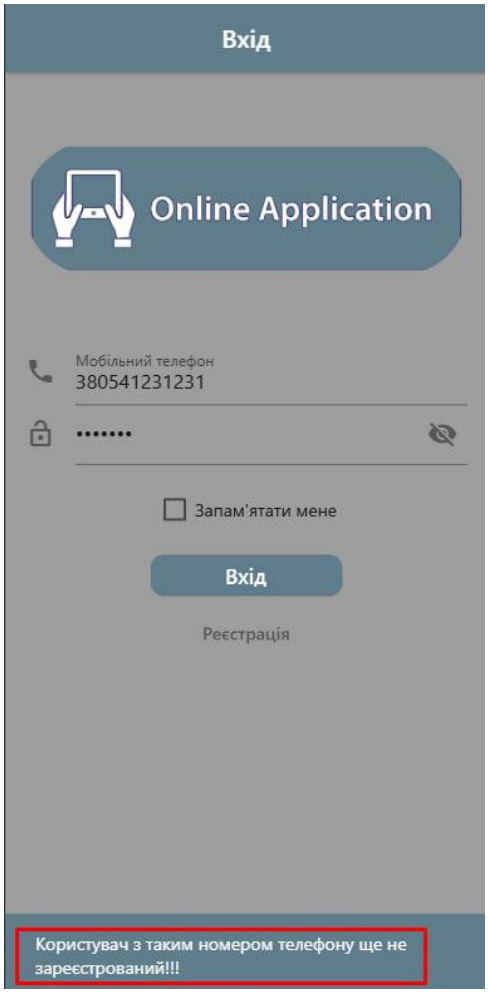


Рисунок 4.1 – Форма входу з повідомленням про помилку

Друге функціональне тестування включало перевірку валідації полів: поле номера телефону перевірялося на відповідність формату (починається з «+380» і має 12 символів), а пароль – на мінімальну довжину (6 символів). При введенні номера телефону, що не відповідає формату (наприклад, лише «380»), система виводила повідомлення «Номер телефону має бути 12 символів». Аналогічно, при введенні пароля з 4 символів з'являлося повідомлення «Пароль має містити більше 6 символів». Також тестувалася відповідність паролів у полях «Пароль» та

«Повторіть пароль» – при невідповідності відображалося повідомлення «Паролі не співпадають». Успішна реєстрація завершувалася переходом до форми входу, якщо всі дані введено коректно (див. рисунок 4.2).

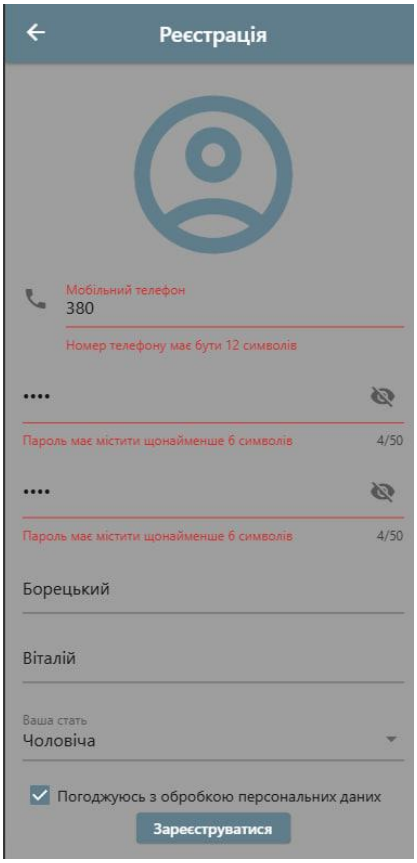


Рисунок 4.2 – Форма реєстрації з валідаційними помилками

Після успішної перевірки функціональності програмного забезпечення на коректність обробки даних, наступним кроком стало тестування адаптивності інтерфейсу на різних розмірах екранів. У процесі тестування було виявлено проблему з масштабуванням: на формі входу в систему нижня частина інтерфейсу обрізається, а замість неї відображається біле поле. На рисунку 4.3 зображено приклад такої помилки, коли елементи форми, такі як посилання на реєстрацію чи повідомлення про помилку, стають недоступними через некоректне відображення. Результати тестування показали, що програма потребує доопрацювання для забезпечення правильної адаптації інтерфейсу до різних розмірів екранів, зокрема шляхом використання засобів для динамічного масштабування та прокручування.

Це дозволить уникнути обрізання контенту та забезпечить зручність використання для всіх користувачів, незалежно від типу пристрою.

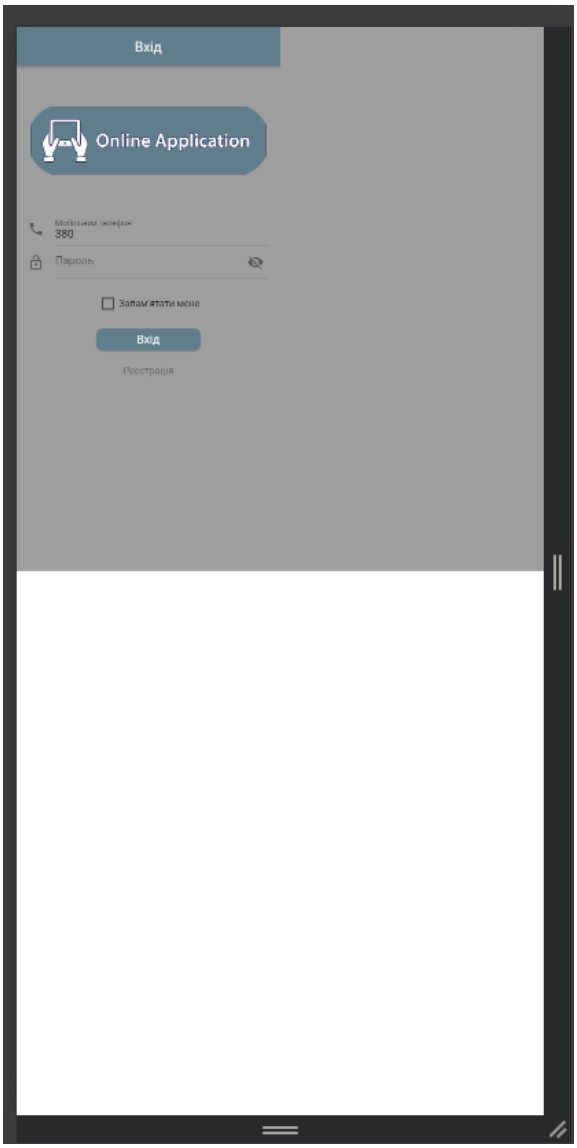


Рисунок 4.3 – Некоректне масштабування форми входу

На рисунку 4.4 зображено головну сторінку програми для ролі адміністратора, яка використовувалася під час тестування зручності використання (usability testing). Інтерфейс включає основні блоки: «Календар» для перегляду записів, «Найближчий запис» для швидкого доступу до актуальних сеансів, а також «Дохід та витрати» для фінансового аналізу.

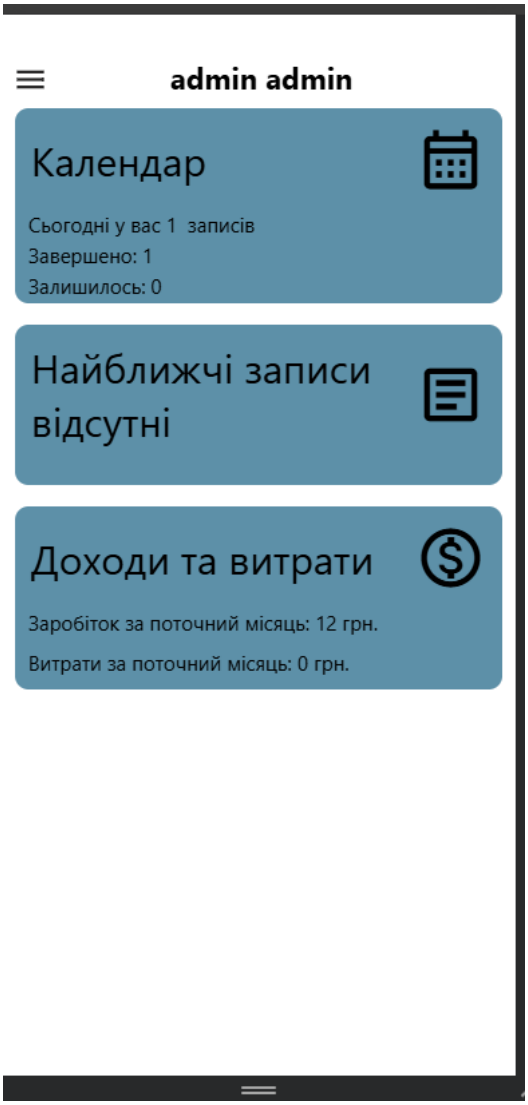


Рисунок 4.4 – Основні блоки інтерфейсу

Тестування показало, що розташування елементів є логічним, а інформація про кількість завершених і запланованих записів представлена чітко, що полегшує швидкий доступ до ключових функцій. Проте було відмічено, що відсутність підказок для нових користувачів може ускладнити знайомство з інтерфейсом, що потребує доопрацювання для підвищення інтуїтивності.

4.2 Розробка інструкції користувача

Інструкція користувача включає перелік технічних вимог для коректного запуску програмного продукту. Інформація про мінімальну та рекомендовану конфігурацію наведена в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i5	Intel Core i3
Оперативна пам'ять	8 ГБ RAM	4 ГБ RAM
Вільний простір на диску	10 ГБ	4 ГБ
Відеокарта	NVIDIA GeForce GTX 1050	NVIDIA GeForce GTX 980
Операційна система	Windows 10	Windows 7

Інструкція користувача для програмного застосунку з оптимізації процесів управління персоналом створена, щоб допомогти користувачам швидко освоїти її функціонал і ефективно використовувати систему. Вона охоплює всі основні етапи роботи: від першого входу до виконання специфічних дій, таких як редагування профілю, перегляд сповіщень, управління записами та адміністрування, забезпечуючи чіткі кроки для користувачів із різними ролями – від звичайних працівників до адміністраторів.

Для початку роботи з програмою необхідно завантажити і встановити на пристрій із операційною системою Android. Після запуску відкриється форма входу, де користувач має ввести свій номер телефону (наприклад, у форматі «+380») та пароль. Опція «Запам’ятати мене» дозволяє зберегти дані для швидкого доступу при наступних входах, що зручно для регулярного використання. Якщо облікового запису немає, користувач може натиснути на посилання «Реєстрація», щоб створити новий профіль, заповнивши необхідні поля, такі як номер телефону, ім’я, прізвище, стать і пароль, із подальшим підтвердженням введених даних.

Після успішного входу користувач потрапляє на головну сторінку, яка залежить від його ролі. Для адміністраторів інтерфейс включає ключові блоки: «Календар» для перегляду всіх запланованих записів, «Найближчий запис», де відображається інформація про кількість завершених і запланованих сеансів

наприклад, «Завершено: 0, Заплановано: 0», а також «Дохід та витрати", що показує фінансовий звіт за місяць наприклад, «Заробіток за поточний місяць: 0 грн» (див. рисунок 4.5).

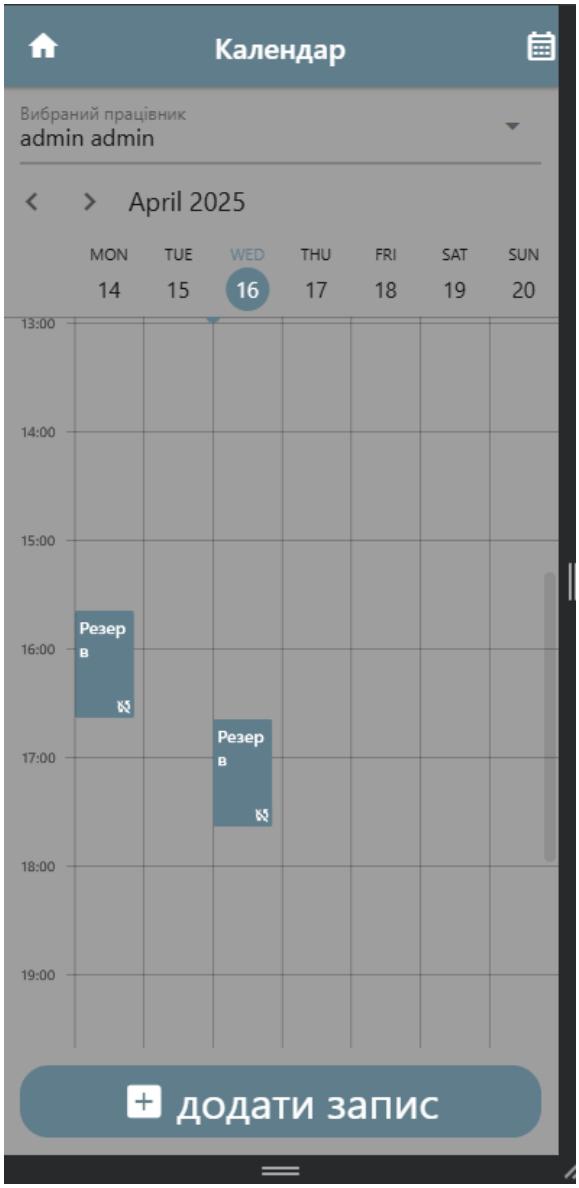


Рисунок 4.5 – Сторінка календаря

Бічне меню, яке відкривається натисканням на іконку в лівому верхньому куті, містить розділи «Додому», «Сповіщення», «Адміністрування», «Налаштування» та «Вихід», дозволяючи швидко переходити між основними функціями програми (див. рисунок 4.6).

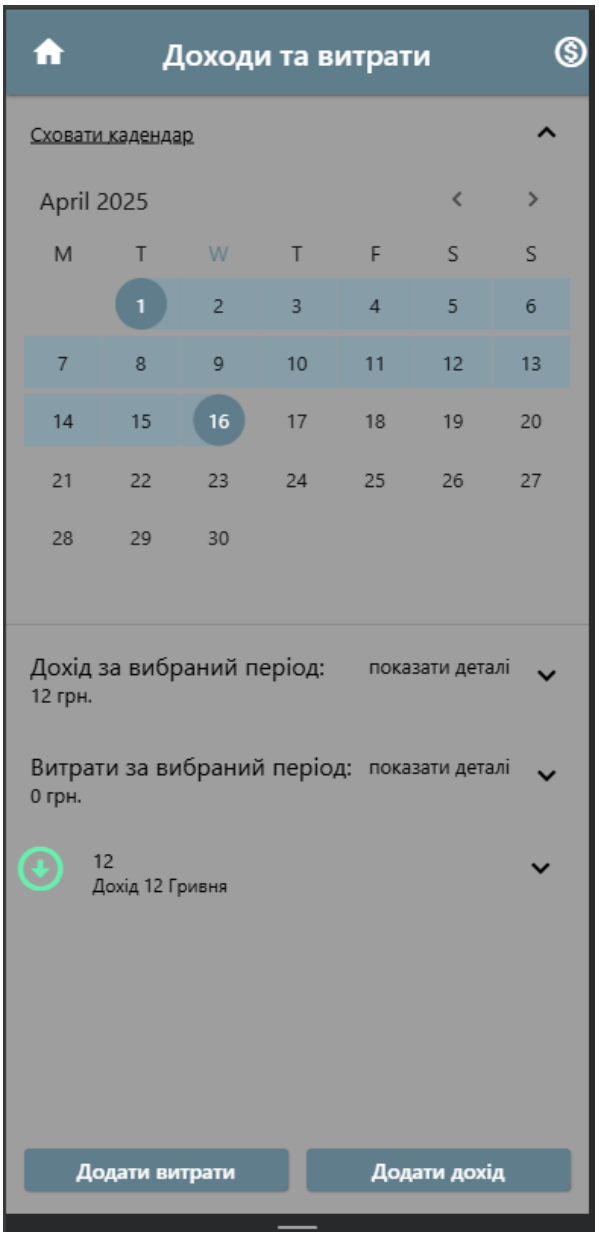


Рисунок 4.6 – Сторінка доходів та витратів

У розділі «Налаштування» користувач може відредагувати свій профіль, ввівши або змінивши дані, такі як ім’я, прізвище, номер телефону чи пароль. Наприклад, поле «Мобільний телефон» має бути у форматі «+380», а пароль – не менше 6 символів. Перед збереженням змін необхідно погодитися з обробкою персональних даних, поставивши галочку у відповідному пункті, після чого натиснути «Зареєструватися» або «Зберегти». У разі помилок, таких як невідповідність формату номера телефону, система видасть повідомлення для виправлення.

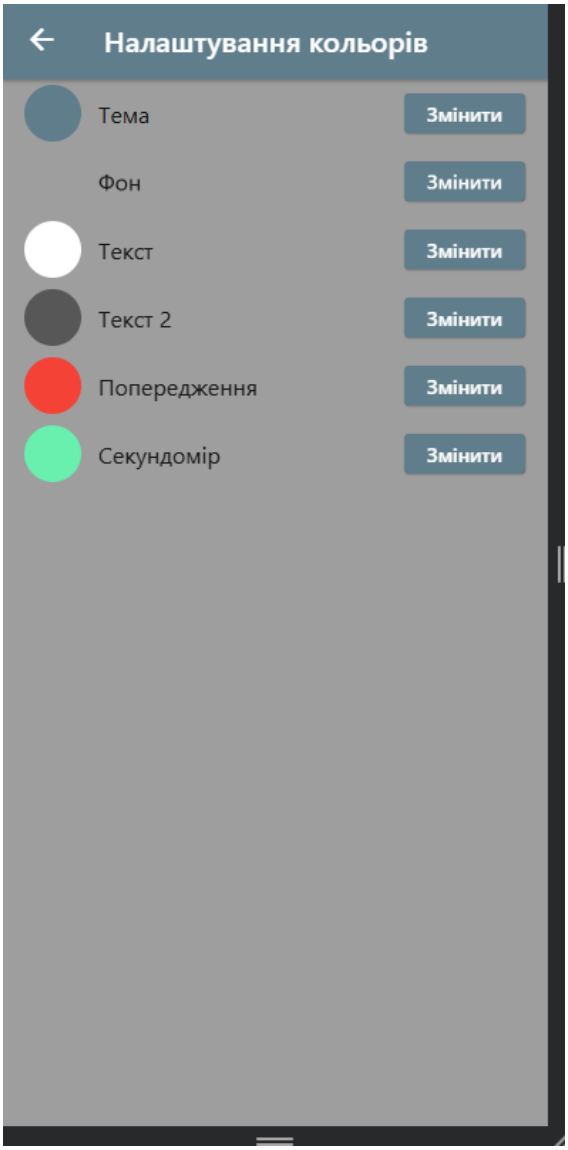


Рисунок 4.7 – Один з параметрів вкладки налаштування

Розділ «Сповіщення» дозволяє користувачам отримувати важливі повідомлення, наприклад, про низький запас матеріалів, як-от «Закінчується Голка, Запас 9 одиниць». Це допомагає вчасно реагувати на потреби, такі як поповнення запасів. Адміністратори можуть скористатися розділом «Адмістрування», щоб управляти ролями користувачів, наприклад, через опцію «Зміна ролі користувача», налаштовувати параметри працівників чи переглядати фінансові звіти, обираючи потрібний період для аналізу доходів і витрат. Ці функції забезпечують гнучке управління персоналом і фінансами, роблячи програму зручною для різних сценаріїв використання.

4.3 Практичне застосування програмного модуля

У сучасних умовах цифрової трансформації бізнесу все більше підприємств прагнуть автоматизувати внутрішні процеси управління персоналом. Це зумовлено необхідністю обробки великого обсягу інформації, підвищенням вимог до точності та оперативності прийняття рішень, а також забезпеченням прозорості кадрових процесів. Ефективні інструменти обліку, планування та аналітики персоналу стають критично важливими для успішного ведення бізнесу. В цьому контексті розроблений застосунок є актуальним та практично виправданим рішенням.

Для оцінки ефективності розробленого програмного продукту було проведено порівняння з існуючими популярними системами управління персоналом – Square, CalendarSpots та Mindbody. Ці платформи широко використовуються на ринку і надають базові можливості для кадрового менеджменту в різних сферах.

Система Square пропонує функції керування графіками роботи, оплатою та базовою фінансовою аналітикою. Однак вона не підтримує фінансовий трекінг у реальному часі, таймер для відстеження тривалості сеансів, а також не дозволяє гнучко налаштовувати зарплату для кожного працівника. Незважаючи на наявність системи ролей і прав доступу, Square має обмеження щодо персоналізації фінансових параметрів.

CalendarSpots спеціалізується на плануванні зустрічей і записах клієнтів, що робить її корисною для закладів з попереднім записом. Водночас ця система не надає можливостей для фінансового трекінгу в реальному часі, а також не має вбудованого таймера та гнучкого налаштування зарплати. Крім того, система не підтримує розподіл прав доступу за ролями.

Mindbody володіє розширеним функціоналом, включаючи фінансовий облік і маркетинг, а також підтримує таймер для відстеження тривалості сеансів. Проте вона не забезпечує фінансовий трекінг у реальному часі і не дозволяє гнучко налаштовувати заробітну плату. Крім того, система не має гнучкої ролі і прав доступу для користувачів.

Тому, на відміну від згаданих рішень, розроблений застосунок містить ряд

унікальних функцій, які підвищують його конкурентоспроможність, зведено в таблиці 4.2.

Таблиця 4.2 – Унікальні функції застосунку

Критерії	Squre	CalendaSpots	Mindbody	Власний застосунок
Фінансовий трекінг у реальному часі	-	-	-	+
Таймер для відстеження тривалості сеансу	-	-	+	+
Гнучке налаштування зарплати	-	+	-	+
Гнучка система ролей і прав доступу для користувачів	+	-	-	+
Загальна оцінка	25%	25%	25%	100%

Таким чином, практичне впровадження розробленого програмного модуля показало його високу ефективність і доцільність використання. Застосунок дозволяє автоматизувати ключові кадрові процеси, зменшує адміністративне навантаження, забезпечує прозорість у розрахунках та сприяє прийняттю обґрунтованих управлінських рішень. Усе це робить його не лише корисним, але й необхідним інструментом у сучасному підході до управління персоналом.

4.4 Висновки

У четвертому розділі було проведено комплексне тестування програмного застосунку для оптимізації процесів управління персоналом. Перевірено коректність роботи системи за різних сценаріїв використання, включаючи обробку

некоректних вхідних даних, збої мережевого з’єднання та адаптивність інтерфейсу до різних розмірів екранів. Крім того, розроблено детальну інструкцію користувача, яка містить рекомендації щодо початку роботи з програмою, опис основних функцій і технічні вимоги до її використання. Окремий підрозділ розділу присвячено практичному застосуванню програмного модуля, де продемонстровано його ефективність у порівнянні з аналогами та обґрунтовано доцільність впровадження в реальному середовищі.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було досягнуто основну мету – створено програмний застосунок для оптимізації процесів управління персоналом. Реалізовано функціонал, що дозволяє автоматизувати облік співробітників, оптимізувати кадрові процедури, підвищити точність аналітики продуктивності персоналу та інтегрувати фінансовий трекер для відстеження доходів і витрат.

У процесі реалізації проєкту були повністю виконані всі поставлені задачі. Розроблено алгоритми обробки кадрових даних, які забезпечують високу точність і швидкість. Створено модуль автоматизації кадрових процесів, що охоплює ведення особистих даних співробітників і облік робочого часу. Реалізовано зручний графічний інтерфейс із інтуїтивною навігацією, який забезпечує комфортну взаємодію з системою. Усі програмні модулі об’єднано в єдину систему для ефективного управління персоналом і підтримки процесу прийняття рішень.

Функціональність і стабільність програмного забезпечення підтверджено в результаті тестування, яке охоплювало перевірку роботи всіх модулів, адаптивності інтерфейсу та стійкості до збоїв. Також підготовлено інструкцію користувача, яка містить технічні вимоги та пояснення щодо використання застосунку, що забезпечує зручність його впровадження для широкого кола користувачів. Робота виконана відповідно до правил [27].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Новікова М. М., Швед А. Б. Сучасні тенденції розвитку HR-менеджменту на підприємствах України [Електронний ресурс]. Проблеми економіки. 2021. №4. С. 127–133. Режим доступу до ресурсу: https://doi.org/10.32983/2222-0712-2021-4-127-133 (дата звернення: 05.11.2024).

2. Балабанова Л. В., Сардак О. В. Управління персоналом: підручник [Електронний ресурс]. Київ: Центр навчальної літератури, 2011. 468 с. Режим доступу до ресурсу: http://surl.li/abc123 (дата звернення: 05.11.2024).

3. Дашко І. М., Михайліченко Л. В. Сучасні підходи та особливості управління персоналом в умовах воєнного стану [Електронний ресурс]. Економіка та суспільство. 2024. № 59. Режим доступу до ресурсу: http://surl.li/ctqesy (дата звернення: 05.11.2024).

4. Борецький В.О., Програмний застосунок для оптимізації процесів управління персоналом. LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії Вінниця: ВНТУ, 2025. 2с. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper /view. (дата звернення 08.11.2025)

5. Борецький В.О., Розробка системи управління персоналом. XIII Міжнародна науково-практична конференція з інформаційних систем та технологій Infocom Advanced Solutions 2025. Київ: КПІ, 2025. 2с. [Електронний ресурс] – Режим доступу до ресурсу: https://ist.kpi.ua/uk/konferencziya-infocom/ (дата звернення 02.03.2025)

6. Human Resource Management Systems: A Comparative Analysis / John D. Smith, Anna K. Brown – New York: HR Tech Press, 2022. – 156 p.

7. HR Automation Trends: Software Solutions for Efficiency [Електронний ресурс] / Laura M. Evans, Peter J. Clark. – Chicago: HR Digital Press, 2023. – Режим доступу до ресурсу: https://hrdigitalpress.com/automation-trends-2023 (дата звернення: 15.04.2025).

8. Strategic Workforce Planning with Digital Tools [Електронний ресурс] / Rachel K. Patel, Steven L. Harris. – Boston: Workforce Tech Journal, 2024. – Режим доступу до ресурсу: https://workforcetechjournal.org/strategic-planning-2024 (дата звернення: 20.04.2025).

9. Square Business Platform [Електронний ресурс] – Режим доступу до ресурсу: https://squareup.com/us/en (дата звернення: 10.04.2025).

10. CalendarSpots Scheduling Software [Електронний ресурс] – Режим доступу до ресурсу: https://www.calendarspots.com/ (дата звернення: 10.04.2025).

11. Mindbody Wellness Business Software [Електронний ресурс] – Режим доступу до ресурсу: https://www.mindbodyonline.com/ (дата звернення: 10.04.2025).

12. Modern Approaches to Software Development for HR Management / Elena V. Petrova – London: Tech Innovations Publishing, 2023. – 112 p.

13. Python Official Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/ (дата звернення: 10.04.2025).

14. Visual Studio Code Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/docs (дата звернення: 10.04.2025).

15. Dart Programming Language [Електронний ресурс] – Режим доступу до ресурсу: https://dart.dev/guides (дата звернення: 10.04.2025).

16. Flutter Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://flutter.dev/docs (дата звернення: 10.04.2025).

17. Android Studio Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio (дата звернення: 10.04.2025).

18. MySQL Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://dev.mysql.com/doc/ (дата звернення: 10.04.2025).

19. Flask Web Framework Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/3.0.x/ (дата звернення: 10.04.2025).

20. MySQL Connector/Python Developer Guide [Електронний ресурс] – Режим доступу до ресурсу: https://dev.mysql.com/doc/connector-python/en/ (дата звернення: 10.04.2025).

21. bcrypt Python Library [Електронний ресурс] – Режим доступу до ресурсу: https://pypi.org/project/bcrypt/ (дата звернення: 10.04.2025).

22. Introduction to REST API with Flask [Електронний ресурс] – Режим доступу до ресурсу: https://realpython.com/flask-connexion-rest-api/ (дата звернення: 10.04.2025).

23. shared_preferences Package for Flutter [Електронний ресурс] – Режим доступу до ресурсу: https://pub.dev/packages/shared_preferences (дата звернення: 10.04.2025).

24. Provider Package for Flutter [Електронний ресурс] – Режим доступу до ресурсу: https://pub.dev/packages/provider (дата звернення: 10.04.2025).

25. Effective Testing for Mobile Apps [Електронний ресурс] – Режим доступу до ресурсу: https://www.smashingmagazine.com/2023/05/effective-testing-mobile-apps/ (дата звернення: 10.04.2025).

26. Usability Testing Guide for Mobile Applications [Електронний ресурс] – Режим доступу до ресурсу: https://www.nngroup.com/articles/usability-testing-mobile/ (дата звернення: 10.04.2025).

27. Романюк О. Н., Чорноволик Г. О., Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення»: методичні – вказівки Вінниця : ВНТУ, 2022. 51 с.

ДОДАТКИ

59

Додаток А. Технічне завдання

Міністерство освіти і науки
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
ФОП Михальнюк О. О.
«25» березня 2025 року

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор Романюк О. Н.
«25» березня 2025 року

Технічне завдання
на бакалаврську кваліфікаційну роботу
«Розробка програмного застосунку для оптимізації процесів управління персоналом»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ. О. М. Ткаченко
«24» березня 2025р.

Виконав:
студент гр. 1ПІ-21б В. О. Борецький
«24» березня 2025р.

м. Вінниця – 2025

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмного застосунку для оптимізації процесів управління персоналом»

Галузь застосування – оптимізації управління персоналом.

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ 20 березня 2025р. №97 ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки

Метою бакалаврської кваліфікаційної роботи є розширення функціональних можливостей програмного застосунку для управління персоналом шляхом розробки програмних модулів, які автоматизують облік співробітників, оптимізують кадрові процеси та підвищують точність аналітики продуктивності персоналу, зокрема за допомогою інтеграції фінансового трекера для відстеження доходів і витрат.

Призначення роботи – автоматизація обліку персоналу, оптимізація кадрових процесів та покращення аналітики продуктивності працівників.

4. Вихідні дані для проведення БКР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Новікова М. М., Швед А. Б. Сучасні тенденції розвитку HR-менеджменту на підприємствах України [Електронний ресурс]. Проблеми економіки. 2021. №4. С. 127–133. Режим доступу до ресурсу: https://doi.org/10.32983/2222-0712-2021-4-127-133 (дата звернення: 05.11.2024).

2. Балабанова Л. В., Сардак О. В. Управління персоналом: підручник [Електронний ресурс]. Київ: Центр навчальної літератури, 2011. 468 с. Режим доступу до ресурсу: http://surl.li/abc123 (дата звернення: 05.11.2024).

3. Дашко І. М., Михайліченко Л. В. Сучасні підходи та особливості управління персоналом в умовах воєнного стану [Електронний ресурс]. Економіка та суспільство. 2024. № 59. Режим доступу до ресурсу: http://surl.li/ctqesy (дата звернення: 05.11.2024).

4. HR Automation Trends: Software Solutions for Efficiency [Електронний ресурс] / Laura M. Evans, Peter J. Clark. – Chicago: HR Digital Press, 2023. – Режим доступу до ресурсу: https://hrdigitalpress.com/automation-trends-2023 (дата звернення: 15.04.2025).

5. Технічні вимоги

Методи збереження даних, методи авторизації та аутентифікації, методи обробки інформації про завдання, методи інтеграції з REST API, методи побудови графічного інтерфейсу, методи аналітики, методи тестування, методи кешування, методи ведення журналу подій, методи динамічного програмування, методи резервного копіювання.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт

1. Пояснювальна записка до БКР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1.	Аналіз розвитку технологій управління персоналом	25.03.25 – 02.04.25
2.	Розробка архітектури та методів програмного продукту	03.04.25 – 14.04.25
3.	Розробка програмних модулів застосунку	15.04.25 – 02.05.25
4.	Тестування і практичне застосування системи	03.04.25 – 26.04.25
5.	Оформлення матеріалів до захисту БКР	27.05.25 - 30.05.25

10. Порядок контролю та прийняття

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

63

Додаток Б. Протокол перевірки кваліфікаційної роботи

Назва роботи: «Розробка програмного застосунку для оптимізації процесів управління персоналом»

Тип роботи: Бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: кафедра програмного забезпечення, ФІТКІ, 1ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 3%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ к.т.н., доц. кафедри ПЗ Ткаченко О. М.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Борецький В. О.
(підпис) (прізвище, ініціали)

Додаток В. Акт впровадження

УЗГОДЖЕНО

ФОП Михальнюк О. О.

«2» червня 2025 року

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ

д.т.н., професор Романюк О. Н.

«2» червня 2025 року

АКТ ВПРОВАДЖЕННЯ №___
результатів науково-дослідних робіт

Замовник: ФОП Михальнюк О. О.

Цим актом підтверджується, що результати роботи – «Розробка програмного застосунку для оптимізації процесів управління персоналом»,

що виконав студент гр. 1ПІ-21б ВНТУ Борецький В. О.

за договором про творчу співдружність без взаємних грошових розрахунків на громадських засадах відповідно до теми, затвердженої наказом № 97 від 20 березня 2025р., виконано з 25 березня по 30 травня.

Та впроваджено у ФОП Михальнюк О. О.

1. Вид впроваджених результатів: експлуатація програмного забезпечення
2. Характеристика масштабу впровадження: одиничне
3. Форма впровадження: дослідний зразок
4. Новизна результатів науково-дослідної роботи: якісно нові
5. Впроваджені: в системі аналізу продуктивності обладнання
6. Соціальний та науково-технічний ефект: спеціальне призначення

Від виконавця:

студент групи 1ПІ-21б

_____ Борецький В. О.

Керівник: к.т.н., доц. кафедри ПЗ

Від ФОП Михальнюк О. О.:

_____ Михальнюк О. О.

_____ Ткаченко О. М.

Додаток Г. Лістинг програми

responseREST.py

```
import zlib


from flask import Flask, request, jsonify
from flask_restful import Api, Resource, reqparse
from flask_cors import CORS
from authorizationUser import UserAuthorizer
from deleteData import deleteData
from registrationUser import UserRegistrar
from updateUser import UserUpdater
import conversionJson
from selectUser import selectUserData
from insertUser import insertUserData
import logging

# Налаштування логування
logging.basicConfig(filename='app.log', level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')




app = Flask(__name__)
api = Api()
CORS(app)

class Main(Resource):
    def get(self):
```

```
    try:
        userName = request.args.get('userName')
        userPass = request.args.get('userPass')
        userId = request.args.get('userId')
        typeOperation = request.args.get('typeOperation')

        logging.info("GET request received with parameters: userName=%s, userId=%s, typeOperation=%s", userName, userId, typeOperation)

        # if (typeOperation == 'authorize'):
        #     authorization = UserAuthorizer()
        #     status, dataDB, columnName, rowtype  = authorization.authorize(userName=userName, userPass=userPass)
        #     if dataDB is not None and columnName is not None and rowtype is not None and status == 0:
        #         dataForJson = dataDB, columnName, rowtype
        #         resultConvertJson = conversionJson.convertJson(dataForJson)
        #         return resultConvertJson, 200
        #     else:
        #         resultConvertJson = conversionJson.convertJson(status)
        #         return resultConvertJson, 500
        #
        #     # result = conversionJson.convertJson(users)
        #     logging.info("Authorization operation for user: %s", userName)
        #     # return result

        if (typeOperation == 'selectAllRole'):
            select = selectUserData()
```

```
        users = select.selectAllRole()
        result = conversionJson.convertJson(users)
        logging.info("Selecting all role", userId)

        # print(result)
        return result
    elif (typeOperation == 'selectRecordClient'):
        select = selectUserData()
        users = select.selectRecordClient()
        result = conversionJson.convertJson(users)
        logging.info("Selecting all record client", userId)

        # print(result)
        return result
    elif (typeOperation == 'selectAllZone'):
        select = selectUserData()
        users = select.selectAllZone()
        result = conversionJson.convertJson(users)
        logging.info("Selecting all zone", userId)

        # print(result)
        return result
    elif (typeOperation == 'selectStatusSessionDirectory'):
        select = selectUserData()
        users = select.selectStatusSessionDirectory()
        result = conversionJson.convertJson(users)
        logging.info("Selecting statusstssion directori", userId)
```

```
    # print(result)
    return result

elif (typeOperation == 'selectCostSessionDirectory'):
    select = selectUserData()
    users = select.selectCostSessionDirectory()
    result = conversionJson.convertJson(users)
    logging.info("Selecting cost sessiob directory", userId)

    # print(result)
    return result

elif (typeOperation == 'selectSpendingIncome'):
    select = selectUserData()
    users = select.selectSpendingIncome()
    result = conversionJson.convertJson(users)
    logging.info("Selecting spending income", userId)

    # print(result)
    return result

elif (typeOperation == 'selectCurrencyDirectory'):
    select = selectUserData()
    users = select.selectCurrencyDirectory()
    result = conversionJson.convertJson(users)
    logging.info("Selecting currency directory", userId)

    # print(result)
```

```
            return result

        elif (typeOperation == 'selectAllUsers'):
            select = selectUserData()
            users = select.selectAllUsers()
            result = conversionJson.convertJson(users)
            logging.info("Selecting all user", userId)

            # print(result)
            return result


    except Exception as e:
        logging.error("Error occurred in GET request: %s", str(e))
        return {"error": "An error occurred"}, 500



def post(self):
    userName = request.args.get('userName')
    userPass = request.args.get('userPass')
    lastName = request.args.get('lastName')
    firstName = request.args.get('firstName')
    gender = request.args.get('gender')
    mobilephone = request.args.get('mobilephone')
    procPersonalData = request.args.get('procPersonalData')
    employeId = request.args.get('employeId')
```

```
    userId = request.args.get('userId')
    recUserName = request.args.get('recUserName')
    recordDateBegin = request.args.get('recordDateBegin')
    recordDateEnd = request.args.get('recordDateEnd')
    zoneClient = request.args.get('zoneClient')
    statusSession = request.args.get('statusSession')
    typeTransaction = request.args.get('typeTransaction')
    sumaOperation = request.args.get('sumaOperation')
    dateOperation = request.args.get('dateOperation')
    idCurrency = request.args.get('idCurrency')
    coments = request.args.get('coments')
    payCard = request.args.get('payCard')
    typeOperation = request.args.get('typeOperation')

    if (typeOperation == 'authorize'):
        authorization = UserAuthorizer()
        status, dataDB, columnName, rowtype = authorization.authorize(userName=userName,
userPass=userPass)

        if dataDB is not None and columnName is not None and rowtype is not None and status == 0:
            dataForJson = dataDB, columnName, rowtype
            resultConvertJson = conversionJson.convertJson(dataForJson)
            return resultConvertJson, 200
        else:
            resultConvertJson = conversionJson.convertJson(status)
            return resultConvertJson, 500

        # result = conversionJson.convertJson(users)
        logging.info("Authorization operation for user: %s", userName)
```

```
        # return result


    if (typeOperation == 'UserRegistrar'):
        registration = UserRegistrar()
        users = registration.register(
            userName=userName, userPass=userPass, lastName=lastName, firstName=firstName,
            gender=gender, mobilephone=mobilephone, procPersonalData=procPersonalData
        )
        result = conversionJson.convertJson(users)
        return result


    elif (typeOperation == 'insertRecordCient'):
        insert = insertUserData()
        users = insert.insertRecordCient(employeId = employeId, userId = userId, recUserName =
recUserName, recordDateBegin = recordDateBegin,
                                         recordDateEnd = recordDateEnd, zoneClient = zoneClient,
statusSession=statusSession)
        result = conversionJson.convertJson(users)
        print(result)
        return result


    elif (typeOperation == 'insertTimeReservationEmpl'):
        insert = insertUserData()
        users = insert.insertRecordCient(employeId = employeId, userId = userId, recUserName =
recUserName, recordDateBegin = recordDateBegin,
                                         recordDateEnd = recordDateEnd, zoneClient = zoneClient,
statusSession=statusSession)
        result = conversionJson.convertJson(users)
        return result
```

```
        elif (typeOperation == 'insertSpendingIncome'):
            insert = insertUserData()
            users = insert.insertSpendingIncome(typeTransaction=typeTransaction, payCard=payCard,
sumaOperation=sumaOperation,
                                                dateOperation=dateOperation, idCurrency=idCurrency,
coments=coments)
            result = conversionJson.convertJson(users)
            return result



    def put(self):
        userId = request.args.get('userId')
        userPass = request.args.get('userPass')
        lastName = request.args.get('lastName')
        firstName = request.args.get('firstName')
        gender = request.args.get('gender')
        mobilephone = request.args.get('mobilephone')
        userRole = request.args.get('userRole')
        id = request.args.get('id')
        employeId = request.args.get('employeId')
        userName = request.args.get('userName')
        recordDateBegin = request.args.get('recordDateBegin')
        recordDateEnd = request.args.get('recordDateEnd')
        zoneClient = request.args.get('zoneClient')
        costSession = request.args.get('costSession')
        durationSession = request.args.get('durationSession')
        statusSession = request.args.get('statusSession')
        costNeedle = request.args.get('costNeedle')
```

```
costAnesthesia = request.args.get('costAnesthesia')
payCard = request.args.get('payCard')
costMin = request.args.get('costMin')
emplPercent = request.args.get('emplPercent')
cost = request.args.get('cost')
typeTransaction = request.args.get('typeTransaction')
sumaOperation = request.args.get('sumaOperation')
dateOperation = request.args.get('dateOperation')
idCurrency = request.args.get('idCurrency')
coments = request.args.get('coments')
needleType = request.args.get('needleType')
anesthesiaAmount = request.args.get('anesthesiaAmount')
typeOperation = request.args.get('typeOperation')

if typeOperation == 'update_role':
    updater = UserUpdater()
    users = updater.update_role(
        userId=userId, userRole=userRole
    )
    result = conversionJson.convertJson(users)
    return result

elif typeOperation == 'update_user_data':
    updater = UserUpdater()
    users = updater.update_user_data(
        userId=userId, userPass=userPass, lastName=lastName,
        firstName=firstName, gender=gender, mobilephone=mobilephone
    )
```

```
            result = conversionJson.convertJson(users)

            return result


        elif typeOperation == 'updateRecordClient':

            updater = UserUpdater()


            users = updater.updateRecordClient(id = id, employeId = employeId, userId = userId,
userName = userName, recordDateBegin = recordDateBegin, recordDateEnd = recordDateEnd,
zoneClient = zoneClient)


            result = conversionJson.convertJson(users)

            print(result)

            return result


        elif typeOperation == 'updateRecordClientCostStatus':

            updater = UserUpdater()


            users = updater.updateRecordClientCostStatus(id = id, employeId = employeId, costSession =
costSession, durationSession = durationSession,

                                                 statusSession = statusSession, costNeedle = costNeedle,
costAnesthesia = costAnesthesia, payCard = payCard,

                                                 needleType=needleType, anesthesiaAmount=anesthesiaAmount)


            result = conversionJson.convertJson(users)

            print(result)

            return result


        elif typeOperation == 'updateEmpolCost':

            updater = UserUpdater()
```

```
            users = updater.updateEmpolCost(userId=userId, costMin=costMin,
emplPercent=emplPercent)


            result = conversionJson.convertJson(users)

            print(result)

            return result


        elif typeOperation == 'updateCostDirectory':

            updater = UserUpdater()


            users = updater.updateCostDirectory(cost=cost, id=id)


            result = conversionJson.convertJson(users)

            print(result)

            return result


        elif typeOperation == 'updateSpendingIncome':

            updater = UserUpdater()


            users = updater.updateSpendingIncome(id=id, typeTransaction=typeTransaction,
payCard=payCard, sumaOperation=sumaOperation,

                                                 dateOperation=dateOperation, idCurrency=idCurrency,
coments=coments)


            result = conversionJson.convertJson(users)

            print(result)

            return result


    def options(self):
```

```
        return '', 200

    def delete(self):
        id = request.args.get('id')
        employeId = request.args.get('employeId')
        idRecord = request.args.get('idRecord')
        userId = request.args.get('userId')
        typeOperation = request.args.get('typeOperation')

        if typeOperation == 'deleteRecordClient':
            delete = deleteData()

            users = delete.deleteRecordClient(id=id, employeId=employeId)

            result = conversionJson.convertJson(users)
            return result

        elif typeOperation == 'deleteSpendingIncome':
            delete = deleteData()

            users = delete.deleteSpendingIncome(id=id)

            result = conversionJson.convertJson(users)
            return result


class insertMedia(Resource):
    def post(self):
```

```
    userId = request.args.get('userId')
    idRecord = request.args.get('idRecord')
    fileName = request.args.get('fileName')
    fileDate = request.args.get('fileDate')

    insert = insertUserData()
    users = insert.insertMedia(idRecord=idRecord, fileName=fileName, fileDate=fileDate,
                               fileData=request.data, userId=userId)
    result = conversionJson.convertJson(users)
    return result
def get(self):
    try:
        userId = request.args.get('userId')

        select = selectUserData()
        users = select.selectMedia(userId)
        result = conversionJson.convertJson(users)

        # print(result)
        return result

    except Exception as e:
        logging.error("Error occurred in GET request: %s", str(e))
        return {"error": "An error occurred"}, 500


def delete(self):
    idRecord = request.args.get('idRecord')
```

```
        userId = request.args.get('userId')

        delete = deleteData()

        users = delete.deleteMedia(idRecord, userId)

        result = conversionJson.convertJson(users)
        print(result)
        return result


class consumables(Resource):
    def get(self):
        try:

            logging.info("GET request received with parameters")

            select = selectUserData()
            users = select.selectConsumables()
            result = conversionJson.convertJson(users)
            logging.info("Get consumables operation")

            return result


        except Exception as e:
            logging.error("Error occurred in GET request: %s", str(e))
            return {"error": "An error occurred"}, 500
```

```
    def post(self):
        consumableName = request.args.get('consumableName')
        consumableAmount = request.args.get('consumableAmount')
        consumableUsed = request.args.get('consumableUsed')
        consumableDate = request.args.get('consumableDate')



        insert = insertUserData()
        users = insert.insertConsumables(consumableName=consumableName,
consumableAmount=consumableAmount, consumableUsed=consumableUsed,
consumableDate=consumableDate)
        result = conversionJson.convertJson(users)
        return result



    def put(self):
        consumableName = request.args.get('consumableName')
        consumableAmount = request.args.get('consumableAmount')
        consumableUsed = request.args.get('consumableUsed')
        consumableDate = request.args.get('consumableDate')
        id = request.args.get('id')



        updater = UserUpdater()
        users = updater.updateConsumables(consumableName=consumableName,
consumableAmount=consumableAmount, consumableUsed=consumableUsed,
consumableDate=consumableDate, id=id)
        result = conversionJson.convertJson(users)
```

```
        return result



    def delete(self):
        id = request.args.get('id')


        delete = deleteData()


        users = delete.deleteConsumables(id=id,)


        result = conversionJson.convertJson(users)
        print(result);
        return result


class user(Resource):
    def post(self):
        userLogin = request.args.get('userLogin')
        lastName = request.args.get('lastName')
        firstName = request.args.get('firstName')
        gender = request.args.get('gender')
        mobilephone = request.args.get('mobilephone')


        insert = insertUserData()
        status, dataDB, columnName, rowtype = insert.insertAddUser(userLogin=userLogin,
lastName=lastName, firstName=firstName, gender=gender, mobilephone=mobilephone)


        if dataDB is not None and columnName is not None and rowtype is not None and status == 0:
            dataForJson = dataDB, columnName, rowtype
            resultConvertJson = conversionJson.convertJson(dataForJson)
```

```
            return resultConvertJson, 200
        else:
            resultConvertJson = conversionJson.convertJson(status)
            return resultConvertJson, 500




    def options(self):
        return '', 200



api.add_resource(Main, "/authorization")
api.add_resource(insertMedia, "/Media")
api.add_resource(consumables, "/consumables")
api.add_resource(user, "/user")


api.init_app(app)

if __name__ == "__main__":
#    app.run()
    app.run(debug=True, port=3000, host="localhost")
```

globalData.py

```
import 'dart:convert';
```

```
import 'package:flutter/foundation.dart';
import 'package:image_picker/image_picker.dart';
import 'package:shared_preferences/shared_preferences.dart';

import 'components/name_shared_preferences.dart';


class DataProvider extends ChangeNotifier {
  // Локальні змінні
  bool _isRunning = false;
  bool _isStartSession = false;
  Duration _elapsedTime = Duration.zero;
  Duration _pausedTime = Duration.zero;  // Час, який був на момент паузи
  int _costSession = 0;
  DateTime? _startTime;


  Map<String, dynamic> _dataAuthorizer = {};
  List<dynamic> _allRecordClien = [];
  List<dynamic> _dataRecordClien = [];
  List<dynamic> _StatusSessionDirectory = [];
  Map<String, dynamic> _isRuningRecord = {};
  List<dynamic> _dataZone = [];
  List<dynamic> _costDirectory = [];
  List<dynamic> _currencyDirectory = [];
  List<dynamic> _spendingIncome = [];
  List<dynamic> _allUsers = [];
  List<dynamic> _allRole = [];
  List<XFile>? _mediaFileList = []; //добавив знак питання
  List<dynamic> _mediaData = [];
  List<dynamic> _consumablesData = [];
  List<String> _statConsumablesId = [];
  List<dynamic> _notificationData = [];


  // Методи GET для отримання значень параметрів
  bool get isRunning => _isRunning;
  bool get isStartSession => _isStartSession;
  Duration get elapsedTime => _elapsedTime;
  int get costSession => _costSession;

  Map<String, dynamic> get dataAuthorizer => _dataAuthorizer;
  List<dynamic> get allRecordClien =>  _allRecordClien;
  List<dynamic> get dataRecordClien =>  _dataRecordClien;
  List<dynamic> get StatusSessionDirectory =>  _StatusSessionDirectory;
  Map<String, dynamic> get isRuningRecord => _isRuningRecord;
  List<dynamic> get dataZone =>  _dataZone;
  List<dynamic> get costDirectory =>  _costDirectory;
  List<dynamic> get currencyDirectory => _currencyDirectory;
  List<dynamic> get spendingIncome => _spendingIncome;
  List<dynamic> get allUsers => _allUsers;
  List<dynamic> get allRole => _allRole;
```

```
 List<XFile>? get mediaFileList => _mediaFileList; //добавив знак питання
 List<dynamic> get mediaData => _mediaData;
 List<dynamic> get consumablesData => _consumablesData;
 List<String> get statConsumablesId => _statConsumablesId;
 List<dynamic> get notificationData => _notificationData;



 // Методи для управління секундоміром
 void startStopwatch(Map<String, dynamic> isRuningRecord) async {
   SharedPreferences prefs = await SharedPreferences.getInstance();
   if (!_isRunning) { // коли нажали старт
     _isRunning = true;
     await prefs.setBool(NameSP.isRunning, _isRunning);
     _isStartSession = true;
     await prefs.setBool(NameSP.isStartSession, _isStartSession);
     _isRuningRecord = isRuningRecord;
     await prefs.setString(NameSP.isRuningRecord, jsonEncode(_isRuningRecord));

     // Якщо секундомір був на паузі, продовжуємо з того моменту
     if (_pausedTime != Duration.zero) {
       _startTime = DateTime.now().subtract(_pausedTime);
     } else {
       _startTime = DateTime.now();
     }


     await prefs.setString(NameSP.startTime, _startTime!.toIso8601String()); // зберігаємо час
початку сесії
     _updateElapsedTime(prefs);
     notifyListeners();
   } else { // коли нажали пауза
     _isRunning = false;
     await prefs.setBool(NameSP.isRunning, _isRunning);
     _pausedTime = _elapsedTime;  // Зберігаємо поточний час на момент паузи
     await prefs.setInt(NameSP.pausedTime, _pausedTime.inSeconds); // зберігаємо час паузи
     notifyListeners();
   }
 }



 void resetStopwatch() async {
   print('resetStopwatch');
   SharedPreferences prefs = await SharedPreferences.getInstance();
   _elapsedTime = Duration.zero;
   _pausedTime = Duration.zero;  // Очищаємо збережений час паузи
   if (_isRunning) {
     _updateElapsedTime(prefs);
   }
   _isStartSession = false;
   await prefs.setBool(NameSP.isStartSession, _isStartSession);
```

```
    _isRunning = false;
    await prefs.setBool(NameSP.isRunning, _isRunning);
    _costSession = 0;
    await prefs.setInt(NameSP.costSession, _costSession);
    await prefs.remove(NameSP.startTime); // видалити час початку сесії
    await prefs.remove(NameSP.pausedTime); // видалити час паузи
    notifyListeners();
  }

  void _updateElapsedTime(SharedPreferences prefs) {
    int costMin = _dataAuthorizer['cost_min'];
    Future.delayed(const Duration(seconds: 1), () async {
      if (_isRunning) {
        String? storedStartTime = prefs.getString(NameSP.startTime);
        if (storedStartTime != null) {
          DateTime startTime = DateTime.parse(storedStartTime);
          _elapsedTime = DateTime.now().difference(startTime); // розрахунок часу, який пройшов

          // Визначення вартості сесії
          if (_costDirectory[5]['cost'] == '1') {
            _costSession = secondsToMinutes(_elapsedTime) * costMin;
          } else {
            if (_elapsedTime.inMinutes <= 30) {
              _costSession = int.parse(_costDirectory[1]['cost']); // вартість коли менше рівне 30хв
            } else if (_elapsedTime.inMinutes > 30 && _elapsedTime.inMinutes <= 60) {
              _costSession = int.parse(_costDirectory[2]['cost']); // вартість коли більше 30хв менше
рівне 60хв
            } else {
              _costSession = secondsToMinutes(_elapsedTime) * costMin; // вартість за хвилину брати із
БД працівника, потрібно добавити додаткове поле в БД. По замовчуванню в БД для працівників
проставляти 0. В адмін модулі добавити розділ управління працівниками де будем
встановлювати для кожного працівника окремо вартість за ХВ
            }
          }
          await prefs.setInt(NameSP.costSession, _costSession);
          _updateElapsedTime(prefs); // оновлюємо таймер кожну секунду
          notifyListeners();
        }
      }
    });
  }

  /// Відновлення сесії після блокування або повернення до додатку
  void restoreSession() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    _isRuningRecord = jsonDecode(prefs.getString(NameSP.isRuningRecord) ?? '{}');
    String? storedStartTime = prefs.getString(NameSP.startTime);
    int? storedPausedTime = prefs.getInt(NameSP.pausedTime);
    _costSession = prefs.getInt(NameSP.costSession) ?? 0;
    _isRunning =  prefs.getBool(NameSP.isRunning) ?? false;
    _isStartSession = prefs.getBool(NameSP.isStartSession) ?? false;
    if (_isStartSession) {
```

```
    _startTime = DateTime.parse(storedStartTime!);
    if (storedPausedTime != null) {
      _pausedTime = Duration(seconds: storedPausedTime);
      if (_isRunning) {
        _elapsedTime = DateTime.now().difference(_startTime!); // відновлюємо час з паузою
      } else {
        _elapsedTime = _pausedTime;
      }
    } else {
      _elapsedTime = DateTime.now().difference(_startTime!); // розрахунок часу, який пройшов
    }
    _updateElapsedTime(prefs); // починаємо оновлювати таймер
    notifyListeners();
  }
}

int secondsToMinutes(Duration duration) {
  int minutes = duration.inMinutes; // продівський рядок
  // int minutes = duration.inSeconds; // тетсовий рядок щоб не чекати 1хв
  return minutes.ceil();  // Округлення до більшого цілого числа
}


void updateDataRecordClien (List<dynamic> dataRecordClien) {
  _allRecordClien = dataRecordClien;

  _dataRecordClien = dataRecordClien.where((Record) {
    bool hasExcludedUserId = false;

    if (Record['employe_id'] == _dataAuthorizer['user_id']) {
      hasExcludedUserId = true;
    }
    return hasExcludedUserId;
  }).toList();

}

void updateDataAuthorizer (Map<String, dynamic> dataAuthorizer) {
  _dataAuthorizer = dataAuthorizer;
  notifyListeners();
}


void updateDataZone (List<dynamic> dataZone) {
  _dataZone = dataZone;
  notifyListeners();
}

void updateCostDirectory (List<dynamic> costDirectory) {
  _costDirectory = costDirectory;
  notifyListeners();
}
```

```
void updateCurrencyDirectory (List<dynamic> currencyDirectory) {
  _currencyDirectory = currencyDirectory;
  notifyListeners();
}

void updateSpendingIncome (List<dynamic> spendingIncome) {
  _spendingIncome = spendingIncome;
  notifyListeners();
}

void updateAllUsers (List<dynamic> allUsers) {
  _allUsers = allUsers;
  notifyListeners();
}

void updateAllRole (List<dynamic> allRole) {
  _allRole = allRole;
  notifyListeners();
}

void updateMediaFileList (List<XFile> mediaFileList) {
  _mediaFileList = mediaFileList;
  notifyListeners();
}

void updateMediaData (List<dynamic> mediaData) {
  _mediaData = mediaData;
  notifyListeners();
}

///новий метод
void updateMediaFileListNew(List<XFile> mediaFiles) {
  _mediaFileList = mediaFiles;
  notifyListeners();
}
///новий метод
void addMediaFiles(List<XFile> newFiles) {
  _mediaFileList?.addAll(newFiles);
  notifyListeners();
}

void updateConsumablesData (List<dynamic> consumablesData) {
  _consumablesData = consumablesData;
  notifyListeners();
}

void setStatConsumablesId(List<String> ids) {
  _statConsumablesId = ids;
  notifyListeners();
}
```

```
  ///підрахунок чи не закінчуються розхідники
  void updateNotificationData(List<dynamic> notification) {
    _notificationData = notification;
    notifyListeners();
  }
}
```

login_page.dart

```
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import 'package:provider/provider.dart';
import 'package:shared_preferences/shared_preferences.dart';
import '../components/api_login_calls_helper.dart';
import '../components/browser_refresh_mixin.dart';
import '../components/load_config.dart';
import '../components/name_routes.dart';
import '../components/name_shared_preferences.dart';
import '../components/phone_input_field.dart';
import '../components/theme.dart';
import '../globalData.dart';
import 'registration_page.dart';
import 'home_page.dart';
import 'package:my_mobile_application/API REST/API.dart';
import 'dart:convert';

class LoginPage extends StatelessWidget {
  const LoginPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          'Вхід',
          style: TextStyle(color: AppColors.textColor),
        ),
        centerTitle: true,
      ),
      body: const LoginForm(),
    );
  }
}

class LoginForm extends StatefulWidget {
  const LoginForm({Key? key}) : super(key: key);

  @override
  _LoginFormState createState() => _LoginFormState();
}

class _LoginFormState extends State<LoginForm> with BrowserRefreshMixin {
```

```
final TextEditingController _emailController = TextEditingController();
final TextEditingController _passwordController = TextEditingController();
bool _obscureText = true; // Початковий стан для скриття тексту паролю
bool _rememberMe = false; // Додали прапорець для зберігання даних
bool isLoggedIn = false;
final RestApiBD _restApiBD = RestApiBD();
final GlobalKey<FormState> _formKey = GlobalKey<FormState>(); // Ключ форми для валідації
final FocusNode _focusMobilephone = FocusNode();



@override
void initState() {
  super.initState();
  /*WidgetsBinding.instance.addPostFrameCallback((_) async {
    await saveCurrentPage();
  });*/
  _emailController.text = _emailController.text.isEmpty ? '380' : _emailController.text;
  _loadLoginData();
}

Future<void> _loadLoginData() async {
  final prefs = await SharedPreferences.getInstance();

  final email = prefs.getString(NameSP.email) ?? '';
  final password = prefs.getString(NameSP.password) ?? '';
  final rememberMe = prefs.getBool(NameSP.rememberMe) ?? false;

  if (rememberMe) {
    setState(() {
      _emailController.text = email;
      _passwordController.text = password;
      _rememberMe = rememberMe;
    });
  }
}

Future<void> _saveLoginData() async {
  final prefs = await SharedPreferences.getInstance();

  await prefs.setBool(NameSP.rememberMe, _rememberMe);
  await prefs.setBool(NameSP.isLoggedIn, true);
  await prefs.setString(NameSP.lastVisitedPage, Routes.home);
  await prefs.setString(NameSP.email, _emailController.text);
  await prefs.setString(NameSP.password, _passwordController.text);
}


@override
void dispose() {
  _emailController.dispose();
  _passwordController.dispose();
```

```
  super.dispose();
 }

 /// Звичайний вхід
 Future<void> _performApiCalls() async {
  if (_formKey.currentState?.validate() ?? false) {

   // Виклик функції авторизації
   Map<String, dynamic>? userData = await loginApi(
    context: context,
    userName: _emailController.text,
    userPass: _passwordController.text
   );

   if (userData != null) {
    await _saveLoginData(); // Збереження логін-даних

    // Виклик функції для виконання решти запитів
    await performPostAuthorizationApiCalls(
     context: context,
     userData: userData,
    );
   } else {
    return;
   }

   Navigator.pushReplacementNamed(context, Routes.home);
   //Navigator.pushNamedAndRemoveUntil(context, Routes.home, (Route<dynamic> route) =>
false);

   /*// Виклик REST API для авторизації
   ResponseData resultAPIAuthorization = await _restApiBD.authorizationRequest(userName,
userPass);
   var parsedData = json.decode(resultAPIAuthorization.responseBody);

   if (parsedData is List && parsedData.isNotEmpty && parsedData[0] is Map<String, dynamic>
&& parsedData.length == 1 && resultAPIAuthorization.statusCode == 200) {
    context.read<DataProvider>().updateDataAuthorizer(parsedData[0]);
    // Зберігаємо дані після успішного логіну
    await _saveLoginData();

   } else if (resultAPIAuthorization.statusCode == 500) {
    // Використовуємо функцію для обробки помилок
    handleServerError(resultAPIAuthorization, context);
    return;
   } else {
    SnackBarHelper.showSnackBarWithTimer(
     context: context,
     text: 'Невірний формат даних від сервера',
    );
    return;
   }
```

```
    // Якщо авторизація пройшла успішно, продовжуємо з іншими запитами
    String updatedConsumablesData = await _restApiBD.selectRecordClientRequest();
    List<dynamic> jsonUpdatedConsumablesData = json.decode(updatedConsumablesData);
    context.read<DataProvider>().updateDataRecordClien(jsonUpdatedConsumablesData);


    await _restApiBD.selectCostSessionDirectory().then((selectCostSessionDirectory) {
      context.read<DataProvider>().updateCostDirectory(json.decode(selectCostSessionDirectory));
    });

    await _restApiBD.selectCurrencyDirectory().then((selectCurrencyDirectory) {
      context.read<DataProvider>().updateCurrencyDirectory(json.decode(selectCurrencyDirectory));
    });

    await _restApiBD.selectSpendingIncome().then((selectSpendingIncome) {
      context.read<DataProvider>().updateSpendingIncome(json.decode(selectSpendingIncome));
    });

    await _restApiBD.selectAllUsers().then((selectAllUsers) {
      context.read<DataProvider>().updateAllUsers(json.decode(selectAllUsers));
    });

    await _restApiBD.selectAllZone().then((resultAPIZone) {
      context.read<DataProvider>().updateDataZone(json.decode(resultAPIZone));
    });

    if (parsedData[0]['user_role'] == 1 || parsedData[0]['user_role'] == 2) {
      await _restApiBD.selectAllRole().then((resultAPIDataRoles){
        context.read<DataProvider>().updateAllRole(json.decode(resultAPIDataRoles));
      });
    }

    await _restApiBD.selectConsumables().then((selectConsumables) {
      context.read<DataProvider>().updateConsumablesData(json.decode(selectConsumables));

      for (var consumable in json.decode(selectConsumables)) {
        int amountConsumable = 0;
        if (statConsumablesId.contains(consumable['id'].toString())) {
          for (var record in jsonUpdatedConsumablesData) {
            if(['2', '3'].contains(record['status_session'].toString()) &&
                DateFormat("yyyy-MM-dd
HH:mm").parse(record['record_date_end']).isAfter(DateFormat("yyyy-MM-dd
HH:mm").parse(consumable['consumable_date'])) &&
                (record['needle_type'].toString() == consumable['id'].toString() ||
record['anesthesia_amount'] != null )) {
              if (consumable['id'].toString() == '1' || consumable['id'].toString() == '2') {
                amountConsumable++;
              }else if (consumable['id'].toString() == '3') {
                amountConsumable += int.tryParse(record['anesthesia_amount'].toString()) ?? 0;
              }
```

```
            }
           }
           if ((int.tryParse(consumable['consumable_amount'].toString())!-amountConsumable) <= 10) {
            notification.add({'title': 'Закінчуються ${consumable['consumable_name'].toString()}',
'subtitle': 'Залишилось ${(int.tryParse(consumable['consumable_amount'].toString())!-
amountConsumable)} одиниць'});
           }
          }
         }
         context.read<DataProvider>().updateNotificationData(notification);
       });*/

       // Переходимо на HomeForm лише після успішної авторизації та виконання всіх запитів
       /*Navigator.pushReplacement(
         context,
         MaterialPageRoute(builder: (context) => const HomeForm()),
       );*/
     }
   }

   /// Швидкий вхід
   Future<void> _fastLoginPerformApiCalls() async {
     List<dynamic> notification = [];
     Map<String, dynamic> dataAuthorizer =
     // '{"user_id": 30, "user_login": "flutter_4", "user_pass": "20527111", "first_name": "flutter_4",
"last_name": "flutter_4", "gender": "F", "mobilephone": "38000000000", "user_role": 4, "role_name":
"user", "role_comment": "користувач"}';
     {"user_id": 30, "user_login": "flutter_4", "user_pass": "20527111", "first_name": "flutter_4",
"last_name": "flutter_4", "gender": "F", "mobilephone": "38000000000", "user_role": 1, "role_name":
"user", "role_comment": "користувач", "cost_min": 10, "empl_percent": 40};
     context.read<DataProvider>().updateDataAuthorizer(dataAuthorizer);



     // Якщо авторизація пройшла успішно, продовжуємо з іншими запитами

     String updatedConsumablesData = await _restApiBD.selectRecordClientRequest();
     List<dynamic> jsonUpdatedConsumablesData = json.decode(updatedConsumablesData);
     context.read<DataProvider>().updateDataRecordClien(jsonUpdatedConsumablesData);
     /*await _restApiBD.selectRecordClientRequest().then((selectRecordClien) {
       context.read<DataProvider>().updateDataRecordClien(json.decode(selectRecordClien));
      });*/

     await _restApiBD.selectCostSessionDirectory().then((selectCostSessionDirectory) {
      context.read<DataProvider>().updateCostDirectory(json.decode(selectCostSessionDirectory));
     });

     await _restApiBD.selectCurrencyDirectory().then((selectCurrencyDirectory) {
      context.read<DataProvider>().updateCurrencyDirectory(json.decode(selectCurrencyDirectory));
     });

     await _restApiBD.selectSpendingIncome().then((selectSpendingIncome) {
```

```
    context.read<DataProvider>().updateSpendingIncome(json.decode(selectSpendingIncome));
   });

   await _restApiBD.selectAllUsers().then((selectAllUsers) {
    context.read<DataProvider>().updateAllUsers(json.decode(selectAllUsers));
   });

   await _restApiBD.selectConsumables().then((selectConsumables) {
    context.read<DataProvider>().updateConsumablesData(json.decode(selectConsumables));

    for (var consumable in json.decode(selectConsumables)) {
     int amountConsumable = 0;
     if (statConsumablesId.contains(consumable['id'].toString())) {
      for (var record in jsonUpdatedConsumablesData) {
       if(['2', '3'].contains(record['status_session'].toString()) &&
          DateFormat("yyyy-MM-dd
HH:mm").parse(record['record_date_end']).isAfter(DateFormat("yyyy-MM-dd
HH:mm").parse(consumable['consumable_date'])) &&
          (record['needle_type'].toString() == consumable['id'].toString() || record['anesthesia_amount']
!= null )) {
         if (consumable['id'].toString() == '1' || consumable['id'].toString() == '2') {
          amountConsumable++;
         }else if (consumable['id'].toString() == '3') {
          amountConsumable += int.tryParse(record['anesthesia_amount'].toString()) ?? 0;
         }
        }
       }
       if ((int.tryParse(consumable['consumable_amount'].toString())!-amountConsumable) <= 10) {
        notification.add({'title': 'Закінчуються ${consumable['consumable_name'].toString()}',
'subtitle': 'Залишилось ${(int.tryParse(consumable['consumable_amount'].toString())!-
amountConsumable)} одиниць'});
       }
      }
     }
    context.read<DataProvider>().updateNotificationData(notification);
   });

   // Переходимо на HomeForm лише після успішної авторизації та виконання всіх запитів
   Navigator.pushReplacement(
    context,
    MaterialPageRoute(builder: (context) => const HomeForm()),
   );
  }


  @override
  Widget build(BuildContext context) {
   return Padding(
    padding: const EdgeInsets.all(16.0),
    child: SingleChildScrollView(
     child: Form(
      key: _formKey, // Прикріплення ключа форми
```

```
child: Column(
  children: [
    const Padding(
      padding: EdgeInsets.only(bottom: 40.0),
    ),
    ClipRRect(
      borderRadius: BorderRadius.circular(80.0),
      child: const Image(
        image: AssetImage('assets/1.PNG'),
        alignment: Alignment.topCenter
      ),
    ),
    const Padding(
      padding: EdgeInsets.only(bottom: 50.0),
    ),

    phoneNumberTextFormField(
      controller: _emailController,
      focusNode: _focusMobilephone,
    ),
    /*TextFormField(
      controller: _emailController,
      validator: (value) {
        if (value?.isEmpty ?? true) {
          return 'Це поле обов\'язкове';
        }
        return null;
      },
      decoration: InputDecoration(
        icon: const Icon(Icons.mail_outline),
        hintText: 'email@gmail.com',
        hintStyle: const TextStyle(
          color: AppColors.text2Color,
        ),
        focusedBorder: UnderlineInputBorder(
          borderSide: const BorderSide(color: AppColors.themColor),
          borderRadius: BorderRadius.circular(8.0),
        ),
      ),
      autofocus: true,
    ),*/

    TextFormField(
      controller: _passwordController,
      validator: (value) {
        if (value?.isEmpty ?? true) {
          return 'Це поле обов\'язкове';
        }
        return null;
      },
      decoration: InputDecoration(
        icon: const Icon(Icons.lock_open_outlined),
```

```
    suffixIcon: IconButton(
      icon: Icon(_obscureText ? Icons.visibility_off : Icons.visibility),
      onPressed: () {
        setState(() {
          _obscureText = !_obscureText;
        });
      },
    ),
    hintText: 'Пароль',
    hintStyle: TextStyle(
      color: AppColors.text2Color,
    ),
    focusedBorder: UnderlineInputBorder(
      borderSide: BorderSide(color: AppColors.themColor),
      borderRadius: BorderRadius.circular(8.0),
    ),
  ),
  obscureText: _obscureText,
),
const SizedBox(height: 20.0),

Row(
  mainAxisAlignment: MainAxisAlignment.center,  // Центрує по горизонталі
  children: [
    Checkbox(
      value: _rememberMe,
      onChanged: (value) {
        setState(() {
          _rememberMe = value!;
        });
      },
    ),
    const Text('Запам\'ятати мене'),
  ],
),
const SizedBox(height: 20.0),

ElevatedButton(
  style: ElevatedButton.styleFrom(
    backgroundColor: them_color,
    minimumSize: const Size(160, 40),
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(10), // Радіус заокруглення кутів
    ),
  ),

  /// ЗВИЧАЙНИЙ ВХІД
  onPressed: _performApiCalls,

  ///ШВИДКИЙ ВХІД
  //onPressed: _fastLoginPerformApiCalls,
  child: Text('Вхід', style: TextStyle(fontSize: 17,  color: AppColors.textColor,),
```

```
                ),
              ),
              const SizedBox(height: 16.0),
              TextButton(
                onPressed: () {
                  // Обробка події натиснення кнопки Реєстрація
                  Navigator.push(
                    context,
                    MaterialPageRoute(builder: (context) => RegisterPage()),
                  );
                },
                child: const Text(
                  'Реєстрація',
                  style: TextStyle(color: text2_color),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Додаток Д – Ілюстративна частина

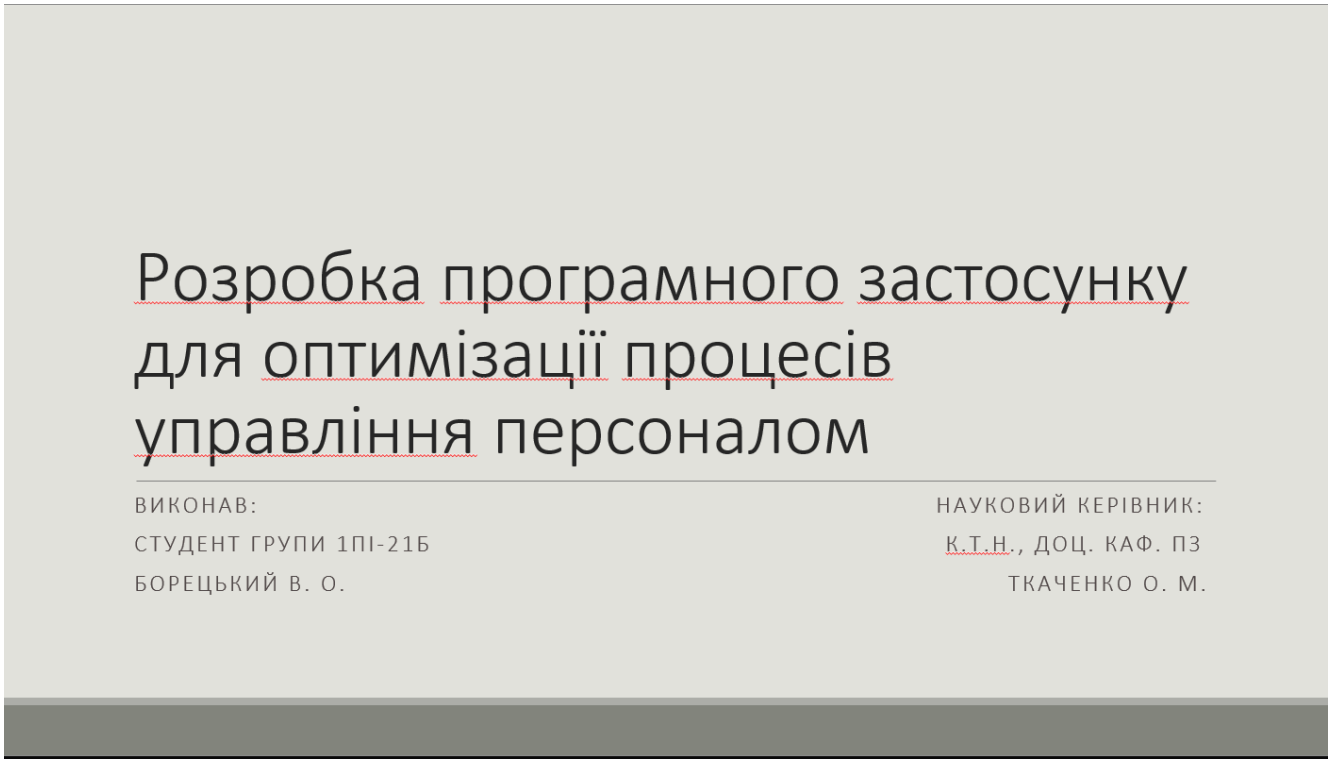


Рисунок Д.1 – Титульний слайд

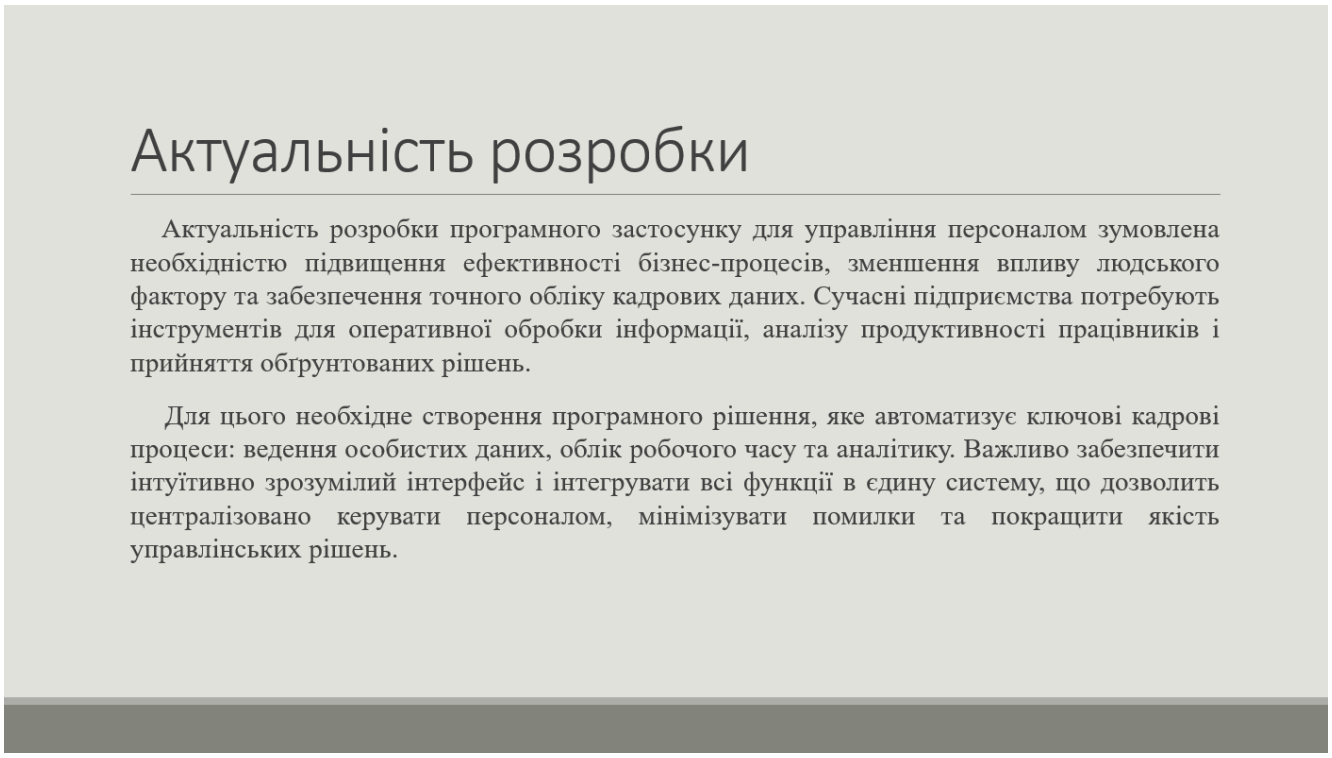


Рисунок Д.2 – Актуальність розробки

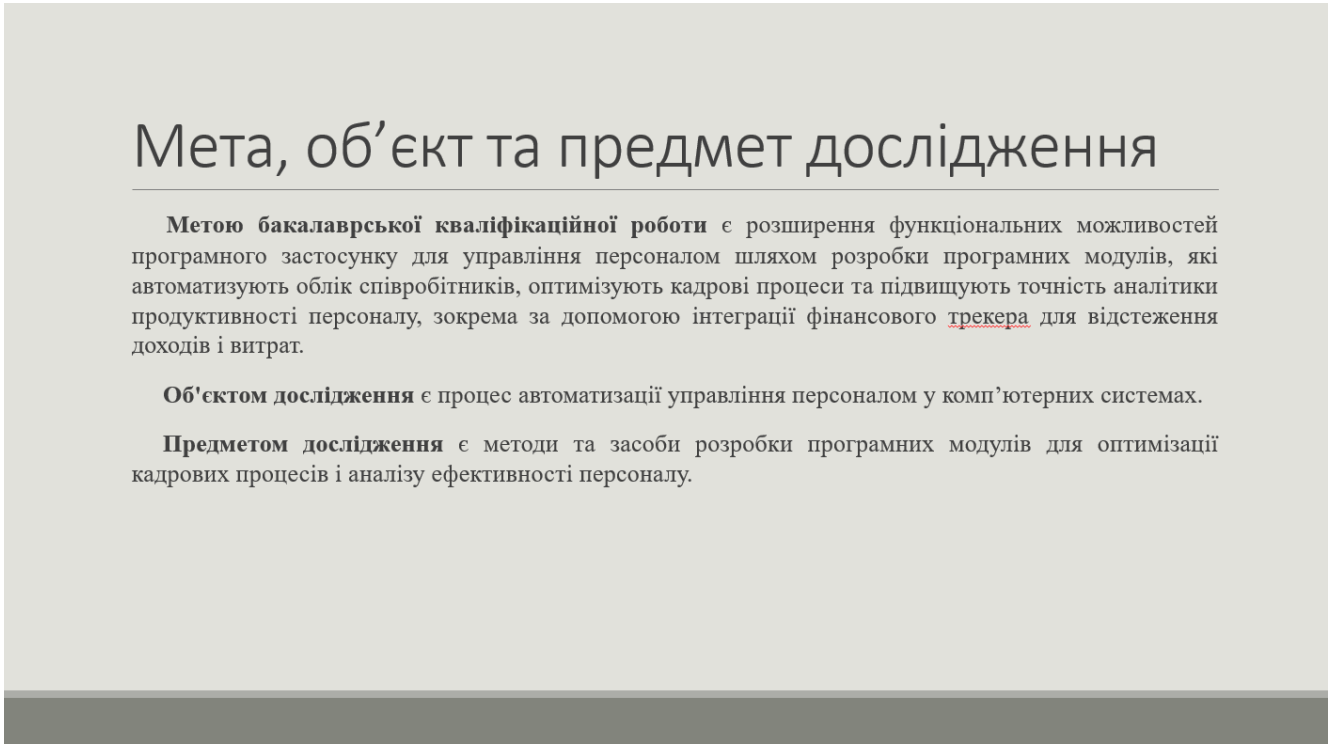


Рисунок Д.3 – Мета, об’єкт та предмет дослідження

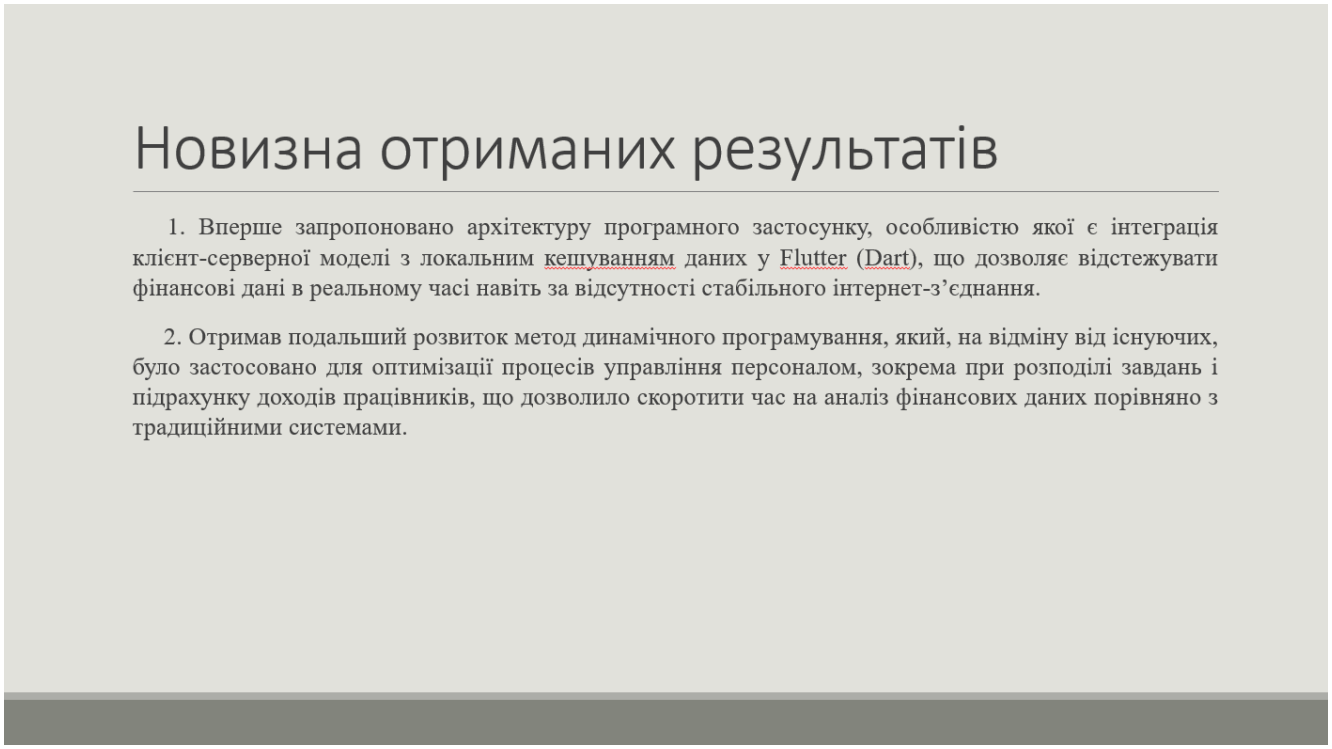


Рисунок Д.4 – Новизна отриманих результатів

Практичне значення отриманих результатів

Практична значення на основі теоретичних положень, викладених у бакалаврській кваліфікаційній роботі, розроблено алгоритми та програмні модулі для автоматизації процесів управління персоналом. Ці засоби забезпечують ефективний облік співробітників, автоматизацію кадрових процесів і аналіз продуктивності, що дозволяє підвищити точність і швидкість прийняття управлінських рішень. Модулі придатні для адаптації в мобільних додатках, що забезпечує гнучкість і зручність використання.

Рисунок Д.5 – Практичне значення отриманих результатів

Аналіз аналогів

На рисунку представлено порівняльний аналіз аналогів системи фінансового трекінгу, включаючи критерії функціональності (наявність аналітики, автоматизація календарного обліку, аналітичні модулі, простота використання, унікальні функції), що демонструє переваги розробленої системи.

Критерії	Squre	CalendaSpots	Mindbody	Власний застосунок
Гнучкість налаштувань	+	+	-	+
Автоматизація кадрового обліку	-	+	+	+
Аналітичні можливості	+	-	+	+
Простота використання	+	-	-	+
Унікальні функції	-	-	-	+
Загальна оцінка	60%	40%	60%	100%

Рисунок Д.6 – Аналіз аналогів

UML-діаграма діяльності

Дана UML-діаграма ілюструє послідовність дій для обробки користувачем програми: спочатку реєструється користувач, потім авторизується, після чого перевіряється правильність введених даних, а в разі успіху виконується перенаправлення до відповідного екранів (головного, екранів управління чи завершення роботи).

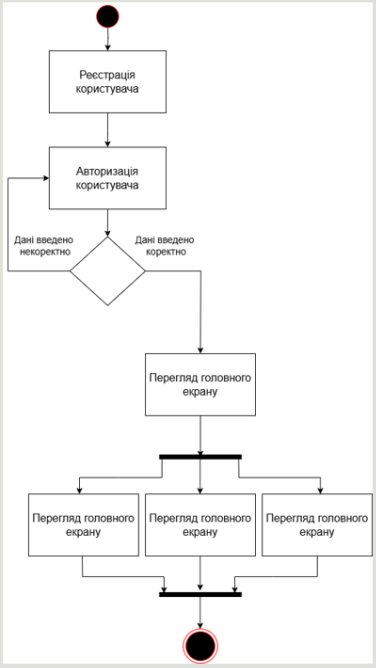


Рисунок Д.7 – UML-діаграма діяльності

Блок-схема алгоритму роботи застосунку

Дана блок-схема ілюструє основний алгоритм роботи програмного застосунку, призначеного для оптимізації процесів управління персоналом, включаючи авторизацію, навігацію між модулями та завершення роботи з збереженням стану.

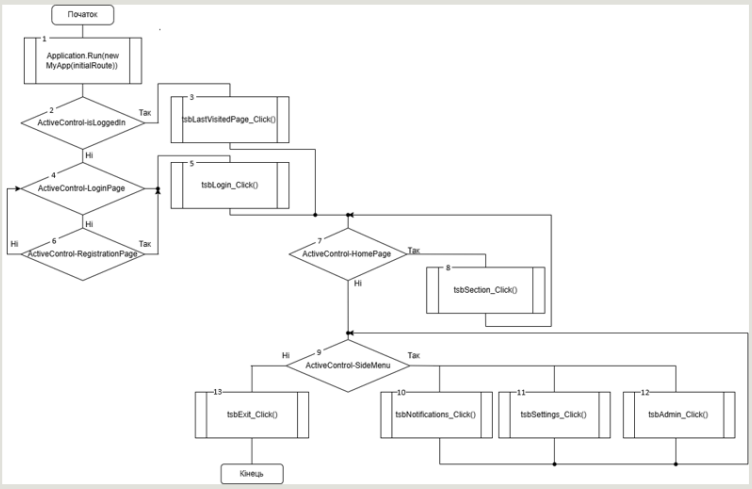


Рисунок Д.8 – Блок-схема алгоритму роботи застосунку

Інтерфейс головного меню

На рисунку представлено інтерфейс користувача програми, який включає бічне меню з основними модулями: головна сторінка, сповіщення, адмін-панель, налаштування та функція виходу, а також відображення статистики, такої як доходи, витрати та кількість заявок за поточний місяць.

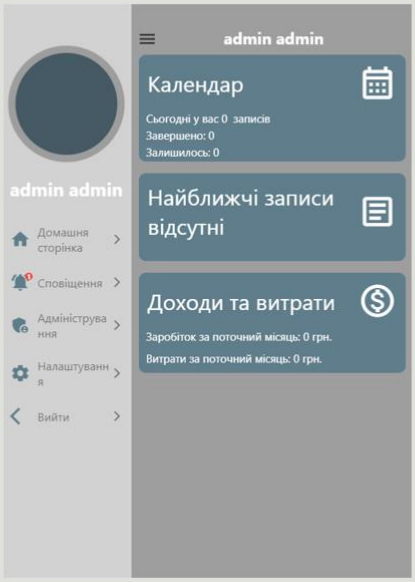


Рисунок Д.9 – Інтерфейс головного меню

Інтерфейс для входу та реєстрації

На екрані представлено два основні вікна мобільного додатку: форма входу та форма реєстрації користувача.
Користувач може авторизуватись за допомогою номера телефону та пароля або створити новий обліковий запис, вказавши необхідну інформацію: прізвище, ім'я, стать та пароль.

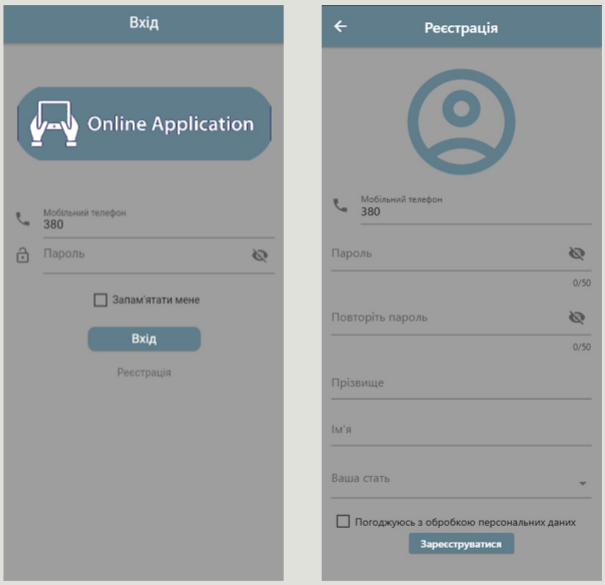


Рисунок Д.10 – Інтерфейс для входу та реєстрації

Інтерфейс та функції екрану адміністрування

Модуль адміністрування надає доступ до ключових функцій керування системою, зокрема зміни ролей користувачів, налаштування працівників, цін, управління розхідниками та додавання нових користувачів. Інтерфейс інтуїтивно зрозумілий, складається з меню з відповідними пунктами, кожен з яких відкриває окрему сторінку для редагування даних. Після додавання користувача з'являється повідомлення про успішне збереження, а дані підтягуються через глобальний DataProvider.

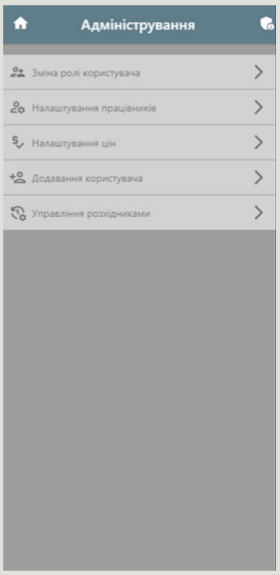


Рисунок Д.11 – Інтерфейс та функції екрану адміністрування

Інтерфейс та функції календаря

Модуль календаря пропонує зручний тижневий вигляд для планування, де відображаються записи та резервації часу з можливістю їх додавання та редагування. Користувачі можуть створювати нові записи, обираючи клієнта, дату, час початку, тривалість і зону, а також резервувати власний час. Інтерфейс включає інтуїтивне вікно для введення даних і кнопку «Додати запис», що забезпечує швидкий доступ до функцій управління розкладом.

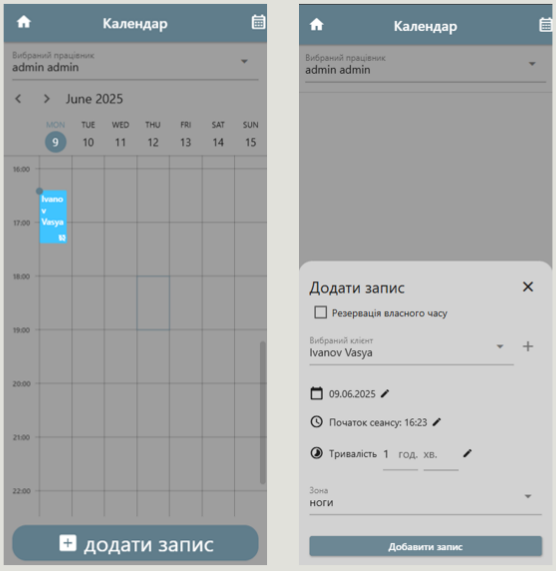


Рисунок Д.12 – Інтерфейс та функції календаря

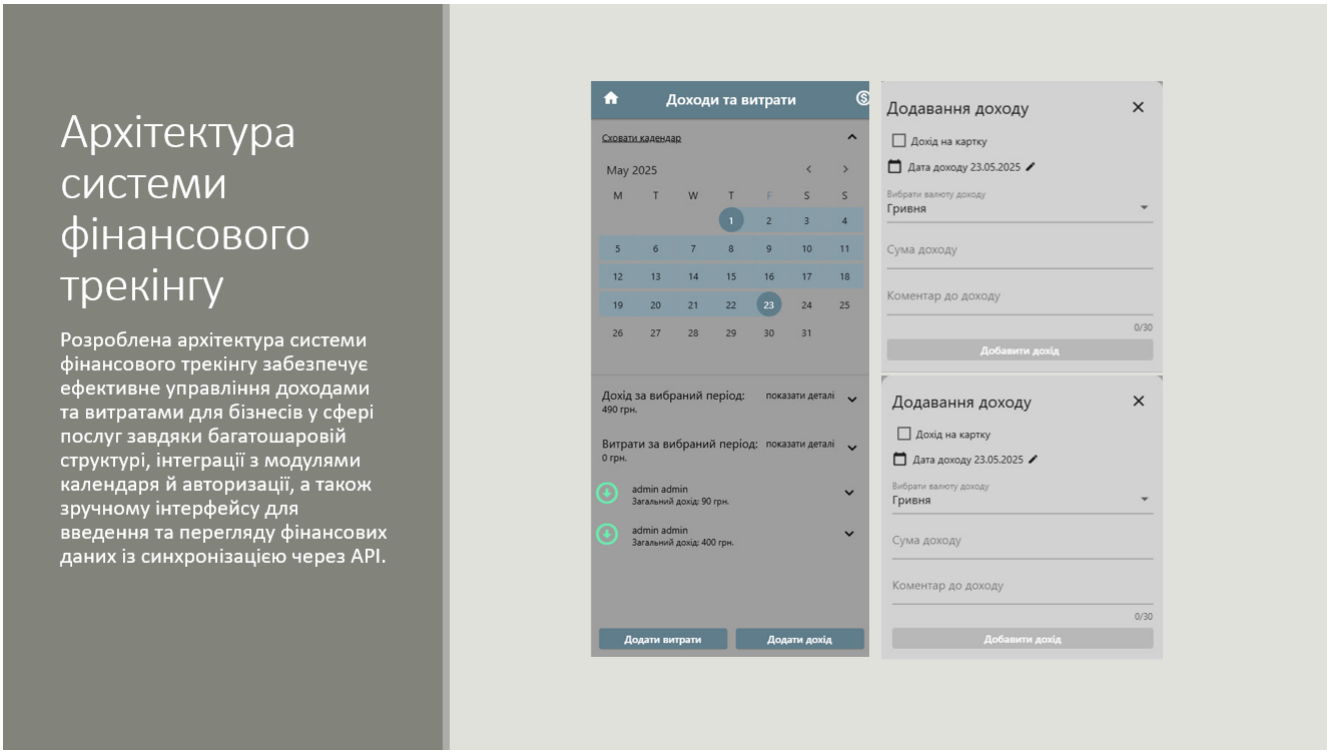


Рисунок Д.13 – Архітектура системи фінансового трекінгу

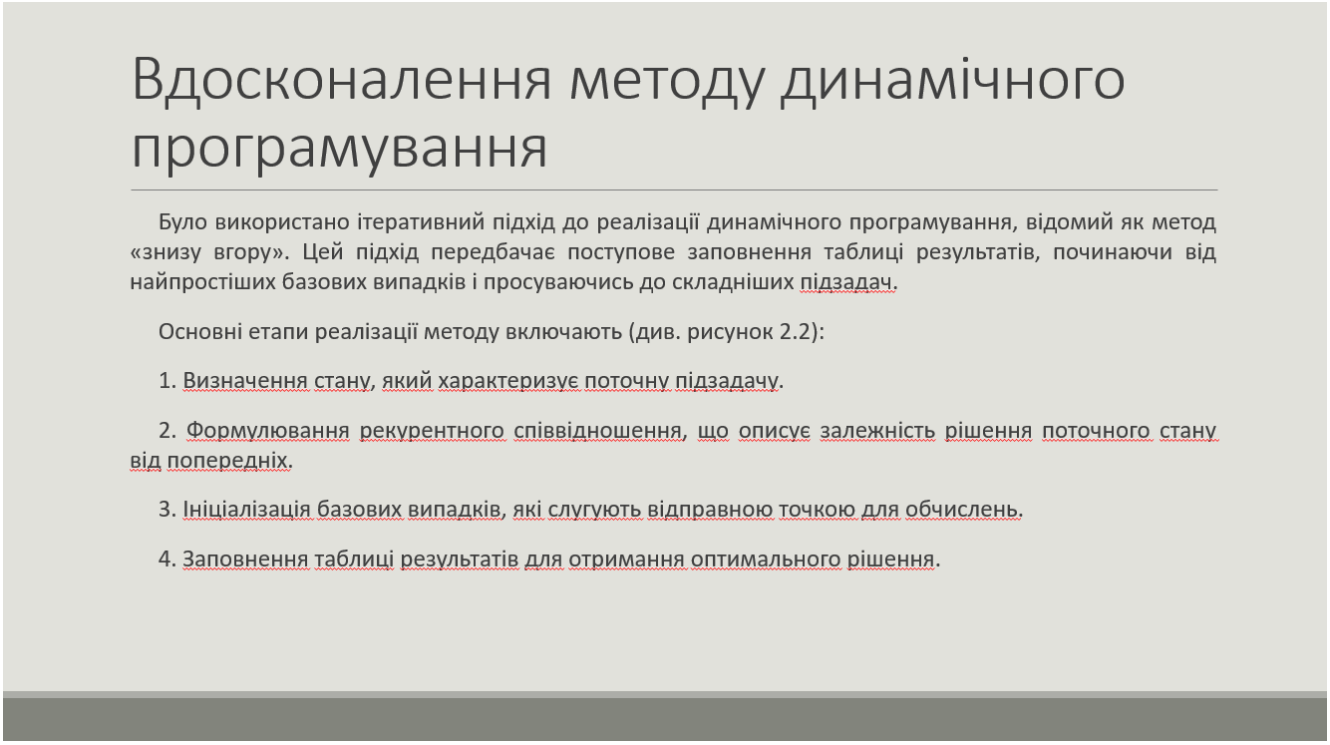


Рисунок Д.14 – Метод динамічного програмування

Апробації та публікації результатів роботи

Основні наукові результати, викладені у бакалаврській кваліфікаційній роботі, доповідалися на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2025) і XIII Міжнародній науково-практичній конференції з інформаційних систем та технологій Infocom Advanced Solutions 2025 (Київ, 2025 р.).

Публікації:

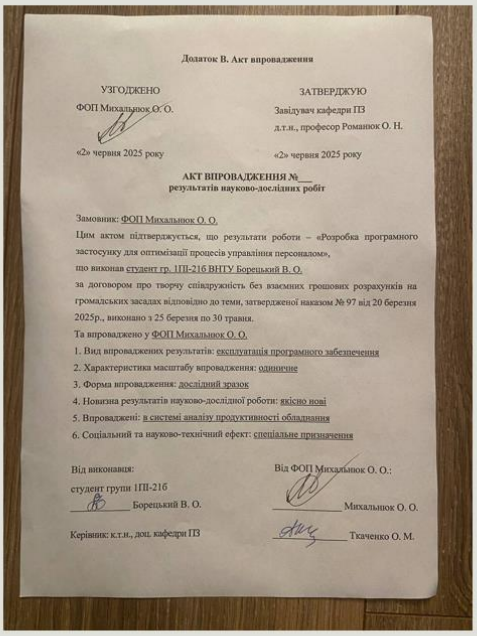
1. Борецький В.О., Програмний застосунок для оптимізації процесів управління персоналом. LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та інженерії Вінниця: ВНТУ, 2025.

2. Борецький В.О., Розробка системи управління персоналом. XIII Міжнародна науково-практична конференція з інформаційних систем та технологій Infocom Advanced Solutions 2025. Київ: КПІ, 2025.

Рисунок Д.15 – Апробації та публікації результатів роботи

Акт впровадження

Програмне забезпечення було використано на підприємстві, що підтверджено актом впровадження.



Додаток В. Акт впровадження

УЗГОДЖЕНО
ФОП Михальнюк О. О.
«2» червня 2025 року

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор Романюк О. Н.
«2» червня 2025 року

АКТ ВПРОВАДЖЕННЯ №___
результатів науково-дослідних робіт

Замовник: ФОП Михальнюк О. О.
Цим актом підтверджується, що результати роботи – «Розробка програмного застосунку для оптимізації процесів управління персоналом»,
що виконав студент гр. 1ПІ-21б ВНТУ Борецький В. О.
за договором про творчу співдружність без взаємних грошових розрахунків на громадських засадах відповідно до теми, затвердженої наказом № 97 від 20 березня 2025р., виконано з 25 березня по 30 травня.
Та впроваджено у ФОП Михальнюк О. О.
1. Вид впроваджених результатів: експлуатація програмного забезпечення
2. Характеристика масштабу впровадження: одиничне
3. Форма впровадження: дослідний зразок
4. Новизна результатів науково-дослідної роботи: якісно нові
5. Впроваджені: в системі аналізу продуктивності обладнання
6. Соціальний та науково-технічний ефект: спеціальне призначення

Від виконавця:
студент групи 1ПІ-21б
_____ Борецький В. О.

Керівник: к.т.н., доц. кафедри ПЗ

Від ФОП Михальнюк О. О.:
_____ Михальнюк О. О.

_____ Ткаченко О. М.

Рисунок Д.16 – Акт впровадження

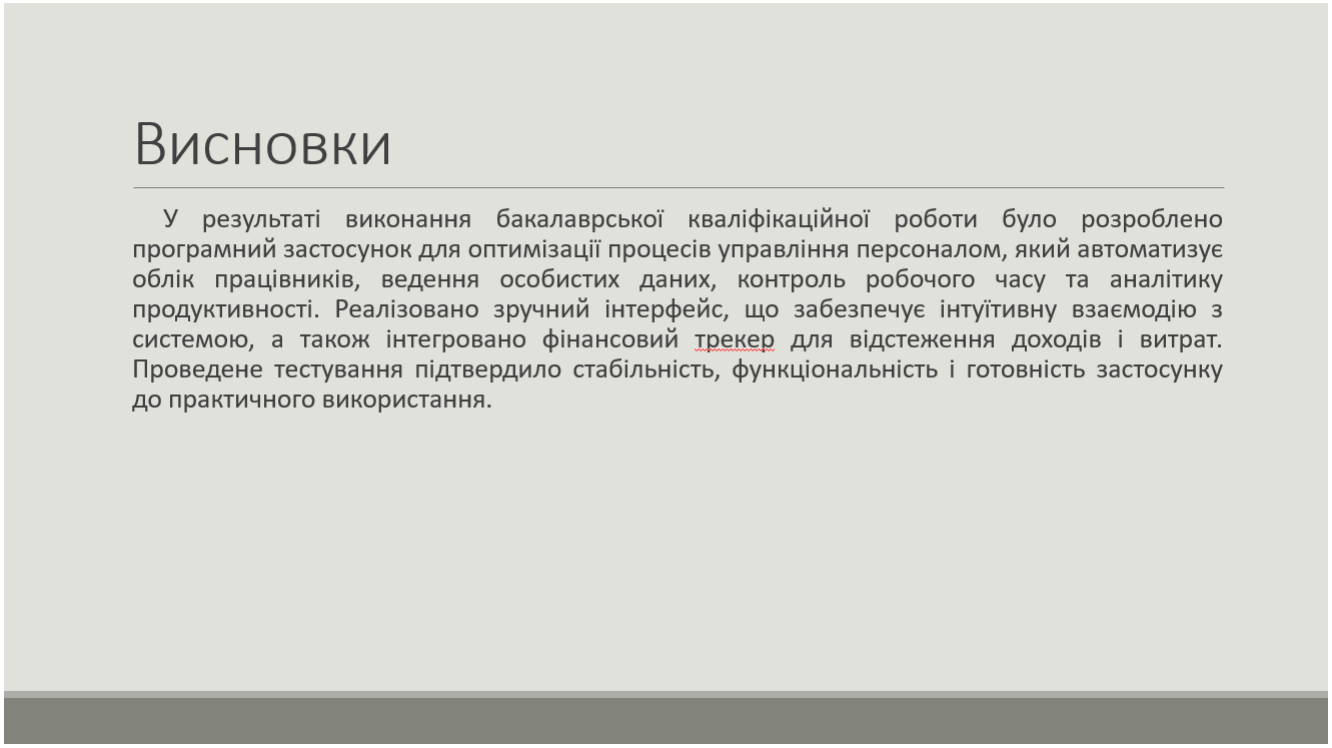


Рисунок Д.17 – Висновки

Рисунок Д.18 – Подяка за увагу