НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ**	_____ **Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Програмне забезпечення для обліку робочого часу та розрахунку заробітної плати операторів логістичної компанії

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ (підпис) **Ганна ВАЙГАНГ**

Керівник бакалаврської кваліфікаційної роботи
к.ф.-м.н., доцент _____ (підпис) **Віктор КИРИЧЕНКО**

Виконав _____ (підпис) **Ілля БУРЕНКОВ**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ
Буренков Ілля Дмитрович

(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення для обліку робочого часу та розрахунку заробітної плати операторів логістичної компанії»

затверджена наказом від «19» грудня 2025 р. № 3204 «С»
Термін подання завершеної роботи на кафедру «20» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): матеріали щодо обліку робочого часу та розрахунку заробітної плати операторів; вимоги до мобільного застосунку; правила нарахування оплати; середовище Xcode; мова Swift; технології SwiftUI та SwiftData.
Перелік питань, що підлягають дослідженню:
1. Аналіз предметної області.
2. Проєктування мобільного застосунку.
3. Розробка модулів обліку змін, задач і зарплати.
4. Тестування та аналіз роботи системи.

Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Віктор КИРИЧЕНКО**

Завдання взяв до виконання _____ (підпис) **Ілля БУРЕНКОВ**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - програмний інтерфейс для взаємодії між програмними компонентами.

ER - модель сутність-зв’язок.

PDF - формат електронного документа.

PIN - персональний цифровий код доступу.

UI - користувацький інтерфейс.

UML - уніфікована мова моделювання.

iOS - операційна система мобільних пристроїв Apple.

Swift - мова програмування для розроблення застосунків Apple.

SwiftData - засіб локального збереження та роботи з моделями даних у застосунку.

ВСТУП

У сучасних умовах діяльність логістичних компаній значною мірою залежить від швидкості обробки інформації, точності внутрішнього обліку та своєчасного контролю виконаних операцій. Оператори логістичної компанії щоденно працюють зі змінами, завданнями, зверненнями, документами та іншими робочими діями, які безпосередньо впливають на розмір їхньої заробітної плати. При цьому оплата праці може залежати не лише від кількості відпрацьованих годин, а й від типу зміни, кількості виконаних завдань, чинних ставок, внутрішніх правил нарахування, додаткових компенсацій та можливих утримань. Через це процес розрахунку заробітної плати стає складним, особливо якщо облік ведеться вручну або за допомогою розрізнених таблиць.

Ручний облік робочого часу та виконаних завдань часто призводить до зайвих витрат часу, дублювання інформації та помилок у розрахунках. Оператору або відповідальній особі потрібно окремо фіксувати дату зміни, її тип, початок і завершення роботи, тривалість зміни, кількість виконаних завдань, а також перевіряти, які з них можуть бути включені до оплати. Якщо таких даних багато, навіть незначна помилка в одному полі може вплинути на підсумкову суму заробітної плати. Особливо це актуально для компаній, де оператори працюють у різні періоди доби, а правила оплати для ранкових, основних і нічних змін можуть відрізнятися.

Актуальність теми бакалаврської кваліфікаційної роботи полягає у необхідності створення зручного програмного засобу, який дає змогу автоматизувати облік робочих змін, внесення виконаних завдань та розрахунок заробітної плати операторів логістичної компанії. Такий застосунок має допомогти зменшити кількість ручних дій, зробити процес нарахування більш прозорим, забезпечити швидкий перегляд статистики за вибраний період і надати користувачу зрозумілий інструмент для контролю власного доходу.

У межах роботи розглядається мобільне програмне забезпечення, орієнтоване на використання на пристроях з операційною системою iOS. Вибір

мобільного формату є доцільним, оскільки оператор або відповідальна особа може швидко внести інформацію про зміну, додати виконані завдання, переглянути суму заробітку та сформувати загальну картину за потрібний період без використання окремого комп’ютера. Мобільний застосунок також дає змогу зробити інтерфейс простим, компактним і доступним для щоденного використання.

Об’єктом дослідження є процес обліку робочого часу та розрахунку заробітної плати операторів логістичної компанії. Цей процес охоплює фіксацію робочих змін, визначення їхньої тривалості, облік виконаних завдань, застосування правил тарифікації, підрахунок загальної суми нарахувань та відображення результатів у зручному для користувача вигляді.

Предметом дослідження є програмні засоби та алгоритми автоматизації обліку змін, внесення виконаних завдань і визначення суми оплати з урахуванням типу зміни, тривалості роботи та правил нарахування. Основна увага приділяється побудові зрозумілої моделі даних, реалізації логіки розрахунку та створенню інтерфейсу, який дає змогу користувачу швидко виконувати основні дії без зайвого навантаження.

Метою бакалаврської кваліфікаційної роботи є розроблення мобільного програмного забезпечення для обліку робочих змін, внесення виконаних завдань та автоматичного розрахунку заробітної плати операторів логістичної компанії. Досягнення цієї мети передбачає аналіз предметної області, визначення функціональних вимог, проєктування структури даних, реалізацію основних модулів застосунку, створення зручного інтерфейсу користувача та перевірку працездатності розробленого рішення.

Для досягнення поставленої мети необхідно проаналізувати особливості обліку робочого часу операторів логістичної компанії, визначити основні сутності предметної області, сформувати вимоги до програмного забезпечення, розробити логічну модель даних, реалізувати модулі обліку працівників, змін і завдань, створити механізм розрахунку заробітної плати, передбачити перегляд

статистики за вибраний період, додати можливість формування звіту та виконати тестування основних сценаріїв роботи застосунку.

Розроблене програмне забезпечення передбачає роботу з працівниками, робочими змінами, виконаними завданнями, налаштуваннями ставок, валютою відображення та статистикою. Користувач може створювати працівників, фіксувати зміну, обирати її тип, вносити виконані завдання, переглядати підсумкову суму заробітку за зміну або за певний період. Для зручності передбачено головний екран зі статистикою, де показники можуть відображатися як загально по всіх працівниках, так і за конкретним вибраним працівником. На цьому екрані відображаються загальна сума заробітку, кількість змін, відпрацьовані години, загальна кількість завдань, кількість оплачуваних завдань та середній заробіток за зміну.

Особливістю розрахунку є те, що заробітна плата формується з кількох складових. До них належать погодинна оплата, оплата за завдання, додаткові бонуси, компенсації та можливі відрахування. У межах реалізованого застосунку основну увагу приділено погодинній оплаті, обліку виконаних завдань і підрахунку заробітку за зміну та період. Перші п’ятнадцять завдань у межах зміни вважаються включеними до базової частини оплати, тому окремо не тарифікуються. Оплата за завдання нараховується лише для тих завдань, які перевищують цей базовий ліміт. Такий підхід дає змогу наблизити розрахунок до реальних внутрішніх правил компанії та уникнути необґрунтованого завищення виплат.

Для підвищення зручності використання в застосунку передбачено автоматичне формування номера завдання в межах конкретної зміни. Користувачу не потрібно самостійно вводити порядковий номер, оскільки система визначає його автоматично. Це зменшує ризик помилок і забезпечує послідовність обліку. Також передбачено введення тривалості завдання, після чого система може використати ці дані для визначення вартості завдання відповідно до заданих правил.

Важливою частиною роботи є забезпечення зрозумілого та цілісного інтерфейсу користувача. Оскільки застосунок призначений для регулярного використання, інтерфейс має бути компактним, логічним і не перевантаженим зайвою інформацією. Основні екрани повинні надавати користувачу швидкий доступ до головних функцій, а деталізована інформація має відображатися лише там, де вона справді потрібна. Такий підхід дозволяє зменшити візуальне навантаження та зробити роботу із застосунком більш комфортною.

Окрему увагу приділено захисту даних. Оскільки застосунок містить інформацію про працівників, зміни, виконані завдання та суми заробітку, доступ до нього має бути обмеженим. Для цього реалізовано механізм блокування за допомогою PIN-коду та можливість використання біометричної автентифікації Face ID або Touch ID, якщо пристрій підтримує таку функцію. Це підвищує рівень безпеки та запобігає несанкціонованому доступу до службової інформації.

Практичне значення розробленого програмного забезпечення полягає у можливості його використання для спрощення щоденного обліку роботи операторів логістичної компанії. Застосунок може бути корисним як для самого оператора, який хоче контролювати свій дохід, так і для відповідальної особи, яка аналізує зміни, задачі та підсумкові нарахування. Автоматизація таких дій допомагає зменшити кількість помилок, пришвидшити підрахунки та зробити процес оплати більш прозорим.

Структура пояснювальної записки відповідає логіці розроблення програмного забезпечення. У першому розділі розглядається предметна область, аналізуються вимоги до системи та описуються основні процеси, які мають бути автоматизовані. У другому розділі подано проєктування інформаційного та програмного забезпечення, зокрема модель даних і діаграми, що відображають структуру майбутньої системи. У третьому розділі описується розроблення застосунку, вибір інструментів, реалізація програмних модулів, алгоритмів і користувацького інтерфейсу. У четвертому розділі наведено рекомендації щодо впровадження та експлуатації системи, результати тестування, вимоги до

апаратного і програмного забезпечення та аналіз отриманих результатів.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область бакалаврської кваліфікаційної роботи пов’язана з організацією обліку робочого часу та розрахунку заробітної плати операторів логістичної компанії. У такій компанії оператори виконують щоденні робочі дії, пов’язані з обробкою інформації, супроводом логістичних процесів, внесенням даних, перевіркою задач, комунікацією з іншими учасниками робочого процесу та виконанням операцій, які мають бути враховані під час визначення оплати праці. Через це процес розрахунку заробітної плати не обмежується простим множенням кількості годин на ставку. Він залежить від типу зміни, тривалості роботи, кількості виконаних завдань, правил тарифікації, додаткових нарахувань та окремих умов, які можуть впливати на кінцеву суму.

У логістичній компанії робота операторів часто організована за змінами. Зміна має дату, час початку, час завершення, тривалість, тип та відповідального працівника. Тип зміни є важливою характеристикою, оскільки ранкова, основна та нічна зміни можуть мати різні умови оплати. Крім того, нічна зміна може переходити через межу календарної доби, тому система обліку повинна коректно визначати її фактичну тривалість. Якщо такі розрахунки виконуються вручну, виникає ризик помилок, особливо у випадках, коли зміна починається в один день, а завершується вже наступного дня.

Окремою складовою предметної області є облік виконаних завдань. Для оператора важливо не лише відпрацювати певну кількість годин, а й виконати робочі задачі, які можуть впливати на суму заробітку. Завдання має порядковий номер у межах зміни, тривалість, статус та ознаку належності до конкретної зміни. У межах розроблюваної системи основна увага приділяється тим завданням, які вносяться користувачем вручну та вважаються виконаними. Такий підхід спрощує облік і дозволяє зосередитися на головній задачі системи,

а саме на правильному підрахунку кількості завдань і визначенні тієї частини, яка підлягає оплаті.

Важливою особливістю розрахунку є те, що не всі виконані завдання оплачуються окремо. Перші п’ятнадцять завдань у межах зміни вважаються включеними до базової частини оплати, тому вони не тарифікуються окремо. Додаткова оплата нараховується лише за завдання, які перевищують цей базовий ліміт. Наприклад, якщо оператор виконав дванадцять завдань, окрема оплата за завдання не нараховується. Якщо ж оператор виконав двадцять завдань, то до окремої оплати потрапляють лише п’ять завдань, починаючи з шістнадцятого. Такий підхід дає змогу зробити модель розрахунку ближчою до реальних правил компанії та уникнути необґрунтованого завищення суми виплат.

Розрахунок заробітної плати оператора формується з кількох частин. Основою є погодинна оплата, яка залежить від тривалості зміни та встановленої ставки. Додатково може враховуватися оплата за виконані завдання, бонуси, компенсації, аванси та інші складові. У межах програмного забезпечення основними об’єктами обліку є працівник, робоча зміна, виконане завдання, ставка, валюта відображення та підсумковий розрахунок. Кожен з цих об’єктів має власні властивості та пов’язаний з іншими об’єктами предметної області. Наприклад, працівник може мати багато змін, зміна може містити багато завдань, а підсумковий розрахунок формується на основі даних зміни, ставки та кількості оплачуваних завдань.

У процесі роботи відповідальна особа або сам оператор повинен мати можливість швидко внести нову зміну, обрати її тип, зазначити працівника, зафіксувати тривалість роботи та додати виконані завдання. Після цього система має автоматично визначити кількість годин, кількість усіх завдань, кількість оплачуваних завдань та суму заробітку. Такий підхід зменшує кількість ручних дій і дає змогу уникнути ситуацій, коли користувач самостійно виконує складні підрахунки або переносить дані між кількома таблицями.

Ручний спосіб обліку має кілька суттєвих недоліків. По-перше, він потребує значної кількості часу, оскільки користувачу доводиться окремо фіксувати кожну зміну та кожне завдання. По-друге, зростає ризик арифметичних помилок під час підрахунку годин, задач і підсумкової суми. По-третє, у разі використання таблиць або нотаток складніше швидко отримати загальну статистику за тиждень, місяць, квартал або рік. По-четверте, відсутність єдиного інтерфейсу ускладнює контроль правильності внесених даних. Саме тому доцільним є створення програмного засобу, який об’єднує ці дії в одному середовищі.

Застосунок, що розробляється в межах роботи, має забезпечити централізований облік даних про працівників, зміни та завдання. Користувач повинен мати змогу переглядати список працівників, додавати нові зміни, бачити коротку інформацію про кожну зміну, відкривати деталі, додавати або видаляти завдання, переглядати суму заробітку та аналізувати статистику за вибраний період. Для зручності передбачено фільтрацію статистики та списку змін за конкретним працівником або перегляд загальних даних по всіх працівниках. При цьому важливо, щоб інформація на основних екранах не була перевантаженою. У списках доцільно відображати лише ключові дані, а докладну інформацію переносити на окремі екрани деталей.

Предметна область також передбачає роботу з валютами. Основні розрахунки можуть виконуватися в одній базовій валюті, а користувач може переглядати суму заробітку в іншій валюті для зручності. Це особливо актуально у випадках, коли внутрішні ставки визначаються в доларах США, а користувачу потрібно бачити підсумкову суму в гривні або євро. При цьому важливо зберігати логіку розрахунку сталою: валюта відображення не повинна змінювати саму формулу нарахування, а лише впливати на спосіб показу підсумкових сум.

Ще одним важливим аспектом є формування звітності. Для практичного використання недостатньо лише показувати суму на екрані. Користувач може потребувати окремий звіт за вибраний період, який можна зберегти, надіслати

або використати для перевірки розрахунків. Такий звіт має містити загальну інформацію про період, кількість змін, відпрацьовані години, кількість завдань, оплачувані завдання та підсумкову суму. Наявність звіту робить програмне забезпечення більш корисним для реального застосування, оскільки результати розрахунку можна передати іншій особі або зберегти як документ.

З огляду на те, що система працює з даними про працівників і розмір заробітку, важливим є питання захисту доступу. Дані про робочі зміни, задачі та оплату мають службовий характер, тому застосунок повинен обмежувати доступ до них. Для цього доцільно використовувати механізм авторизації через PIN-код та біометричну перевірку. Такий підхід не ускладнює щоденну роботу користувача, але забезпечує базовий рівень захисту інформації від стороннього перегляду.

Таким чином, предметна область характеризується наявністю кількох взаємопов’язаних процесів: обліку працівників, фіксації змін, внесення виконаних завдань, застосування правил оплати, перегляду статистики, формування звітів і захисту доступу до даних. Для ефективної автоматизації цих процесів необхідно розробити програмне забезпечення, яке поєднує простий інтерфейс, зрозумілу модель даних і коректну логіку розрахунку заробітної плати. Саме ці вимоги визначають подальший напрям проєктування та реалізації системи.

1.2 Огляд інформаційних джерел та існуючих рішень

Для визначення вимог до програмного забезпечення було розглянуто підходи до обліку робочого часу, контролю виконаних завдань та формування розрахунків заробітної плати. У більшості організацій такі процеси можуть виконуватися за допомогою електронних таблиць, спеціалізованих вебсервісів, мобільних застосунків або внутрішніх корпоративних систем. Кожен із цих підходів має власні переваги та обмеження, тому під час аналізу важливо оцінити не лише наявність окремих функцій, а й відповідність рішення конкретним правилам предметної області.

Найпростішим способом обліку є використання електронних таблиць. У такому випадку користувач самостійно створює таблицю зі змінами, годинами, задачами, ставками та підсумковими формулами. Перевагою такого підходу є гнучкість, оскільки таблицю можна швидко змінити під конкретні потреби. Однак цей спосіб має суттєві недоліки. Дані часто вводяться вручну, формули можуть випадково пошкоджуватися, а контроль правильності розрахунків залежить від уважності користувача. Крім того, у таблицях складніше реалізувати зручний мобільний інтерфейс, захист доступу, автоматичну нумерацію задач, формування PDF-звітів та швидкий перегляд статистики за вибраний період.

Серед спеціалізованих рішень для обліку робочого часу можна виділити Clockify. Сервіс орієнтований на трекінг продуктивності, відвідуваності, billable hours, роботу з таймером, timesheet та звітами. На офіційній сторінці Clockify зазначено можливості відстеження робочого часу, аналізу звітів, керування командами та контролю робочих записів. Таке рішення добре підходить для загального обліку часу, однак воно не враховує специфічну логіку розрахунку заробітної плати операторів логістичної компанії, де важливими є тип зміни, кількість задач у межах конкретної зміни, правило базових п’ятнадцяти задач та окрема тарифікація задач за тривалістю.

Іншим прикладом є Toggl Track. Цей сервіс позиціонується як інструмент для відстеження часу, формування звітів, аналізу прибутковості, payroll-сценаріїв і роботи з різними робочими процесами. Його перевагою є простий підхід до time tracking, наявність звітності та можливість використання в різних командах. Водночас така система залишається універсальною. Вона не орієнтована саме на локальні правила логістичної компанії, де заробіток оператора формується не тільки з часу, а й із задач, які мають власну тривалість, порядок у межах зміни та окремі правила потрапляння до оплати.

Також було розглянуто QuickBooks Time. Це рішення призначене для обліку часу працівників, роботи з timesheets, payroll-звітами, мобільним

доступом і контролем робочих годин. На офіційних сторінках Intuit зазначено, що QuickBooks Time дає змогу відстежувати час, працювати з payroll reports, itemized total time reports, wage reports та іншими видами звітів. Такі можливості є корисними для підприємств, які використовують типові процеси payroll. Проте для розроблюваної системи потрібна більш вузька логіка, пов’язана з внутрішніми правилами нарахування, ручним внесенням задач, розрахунком вартості задач за тривалістю та відображенням результатів у компактному мобільному застосунку.

Проведений огляд показує, що існуючі рішення переважно орієнтовані на загальний облік часу, командну роботу, клієнтські проєкти, GPS-контроль, timesheets або інтеграцію з бухгалтерськими сервісами. Для логістичної компанії, де оператор працює зі змінами та задачами, цього недостатньо. У такій предметній області важливо не лише знати, скільки годин відпрацював працівник, а й правильно визначити, які задачі враховуються в оплаті, які задачі входять до базової частини, як змінюється вартість задачі залежно від тривалості та типу зміни, а також як швидко сформувати зрозумілий звіт за вибраний період.

З огляду на це доцільним є розроблення власного мобільного програмного забезпечення. Воно має бути не універсальним time tracker, а спеціалізованим інструментом для конкретної логіки розрахунку заробітної плати операторів логістичної компанії. Такий підхід дає змогу уникнути зайвих функцій, зробити інтерфейс простішим, адаптувати формули до реальних правил нарахування та забезпечити швидку роботу користувача з основними даними. Для реалізації локальної моделі даних у застосунку доцільно використовувати сучасні інструменти iOS-розробки. Зокрема, SwiftData призначений для опису моделі даних застосунку та роботи з об’єктним графом у Swift, а також може інтегруватися зі SwiftUI.

1.3 Аналіз вимог до програмного забезпечення

Після аналізу предметної області та існуючих рішень було визначено основні вимоги до програмного забезпечення. Розроблювана система має

забезпечувати зручний облік працівників, робочих змін, виконаних задач, ставок, статистики та підсумкових нарахувань. Основним користувачем системи є оператор або відповідальна особа, яка вносить дані про зміни та задачі. Додатковим користувачем може бути бухгалтер або менеджер, який переглядає підсумкові суми, аналізує дані та використовує звіти для перевірки нарахувань.

Програмне забезпечення повинно давати можливість створювати працівників і зберігати про них основну інформацію. До такої інформації належать ім’я працівника, посада, extension, статус активності та пов’язані з ним робочі зміни. Замість складної кадрової інформації система має зберігати лише ті дані, які потрібні для роботи застосунку. Це дозволяє не перевантажувати інтерфейс і зосередитися на головній меті системи, а саме на обліку змін та розрахунку заробітку.

Однією з головних вимог є можливість створення робочої зміни. Під час створення зміни користувач має обрати працівника, дату, тип зміни, час початку, час завершення та погодинну ставку. Система повинна автоматично визначати тривалість зміни, зокрема у випадку нічної зміни, яка може завершуватися наступного календарного дня. Для зручності користувача доцільно передбачити типові шаблони часу для ранкової, основної та нічної зміни, але залишити можливість ручного редагування часу.

Важливою вимогою є облік виконаних задач у межах зміни. Користувач повинен мати змогу додавати задачі вручну, зазначаючи їхню тривалість. Номер задачі має формуватися автоматично, щоб уникнути пропусків і дублювання. Якщо задача видаляється, нумерація в межах зміни повинна залишатися послідовною. Такий підхід зменшує кількість помилок і робить список задач зрозумілим для перевірки.

Розрахунок заробітної плати повинен виконуватися автоматично. Система має враховувати тривалість зміни, погодинну ставку, кількість виконаних задач, кількість оплачуваних задач та правила тарифікації. Перші п’ятнадцять задач у межах зміни не повинні оплачуватися окремо, оскільки вони входять до базової

частини оплати. Якщо кількість задач перевищує п’ятнадцять, до окремої оплати потрапляють лише задачі понад цей ліміт. Завдяки цьому система відображає не просто загальну кількість задач, а саме ту частину, яка впливає на додаткову оплату.

Окремою вимогою є відображення статистики. На головному екрані користувач має бачити заробіток за вибраний період, кількість відпрацьованих годин, кількість змін, загальну кількість задач, кількість оплачуваних задач та середній заробіток за зміну. Статистика має відображатися як по всіх працівниках, так і за конкретним вибраним працівником. Період має обиратися користувачем, наприклад сім днів, місяць, три місяці або рік, а повторне натискання на активний період повинно знімати фільтр і показувати дані за весь час. Такий підхід дозволяє швидко оцінити результати роботи без відкриття кожної зміни окремо.

Програмне забезпечення має підтримувати налаштування. До них належать вибір мови інтерфейсу, вибір валюти відображення, перегляд ставок, правила розрахунку та параметри безпеки. Основні розрахунки можуть виконуватися у базовій валюті, а підсумкові суми мають відображатися у вибраній користувачем валюті. При цьому зміна валюти не повинна змінювати самі дані розрахунку, вона впливає лише на спосіб показу підсумкових сум.

Важливою нефункціональною вимогою є зручність інтерфейсу. Застосунок повинен бути придатним для щоденного використання, тому екрани мають бути компактними, зрозумілими та не перевантаженими зайвою інформацією. У списках працівників і змін доцільно показувати лише ключові дані, а докладну інформацію розміщувати на екранах деталей. Це дозволяє зберегти чисту структуру інтерфейсу та зменшити кількість візуального шуму.

Ще однією нефункціональною вимогою є безпека. Оскільки система містить інформацію про працівників, зміни та суми заробітку, доступ до застосунку має бути захищений. Для цього передбачено блокування за допомогою PIN-коду та підтримку Face ID або Touch ID. Такий механізм є

зручним для користувача і водночас зменшує ризик несанкціонованого доступу до службової інформації.

Також система повинна мати можливість формування PDF-звіту. Користувач має обрати період, після чого застосунок формує документ із підсумковими даними. PDF-звіт повинен мати світлий стиль оформлення, білий фон, чорний або темно-сірий текст, таблицю змін, загальні показники та короткий підсумок. Це дає змогу зберегти результат, надіслати його іншій особі або використати для перевірки розрахунків.

1.4 Моделювання предметної області

Для кращого розуміння структури програмного забезпечення та взаємодії користувача із системою було виконано моделювання предметної області. Моделювання дозволяє показати, які функції має виконувати система, як проходять основні сценарії роботи та які об’єкти беруть участь у розрахунку заробітної плати. У межах цього підрозділу доцільно використати діаграму прецедентів, діаграму послідовності та діаграму діяльності.

Діаграма прецедентів відображає основні можливості системи з погляду користувача. Головним актором є користувач застосунку, який може керувати працівниками, створювати зміни, додавати задачі, переглядати статистику за всіма працівниками або за конкретним працівником, змінювати налаштування, формувати PDF-звіт і виконувати розблокування застосунку. Додатково можна виділити бухгалтера або менеджера, який використовує систему для перегляду підсумкових нарахувань і звітів. Усі ці функції утворюють загальну модель використання програмного забезпечення.

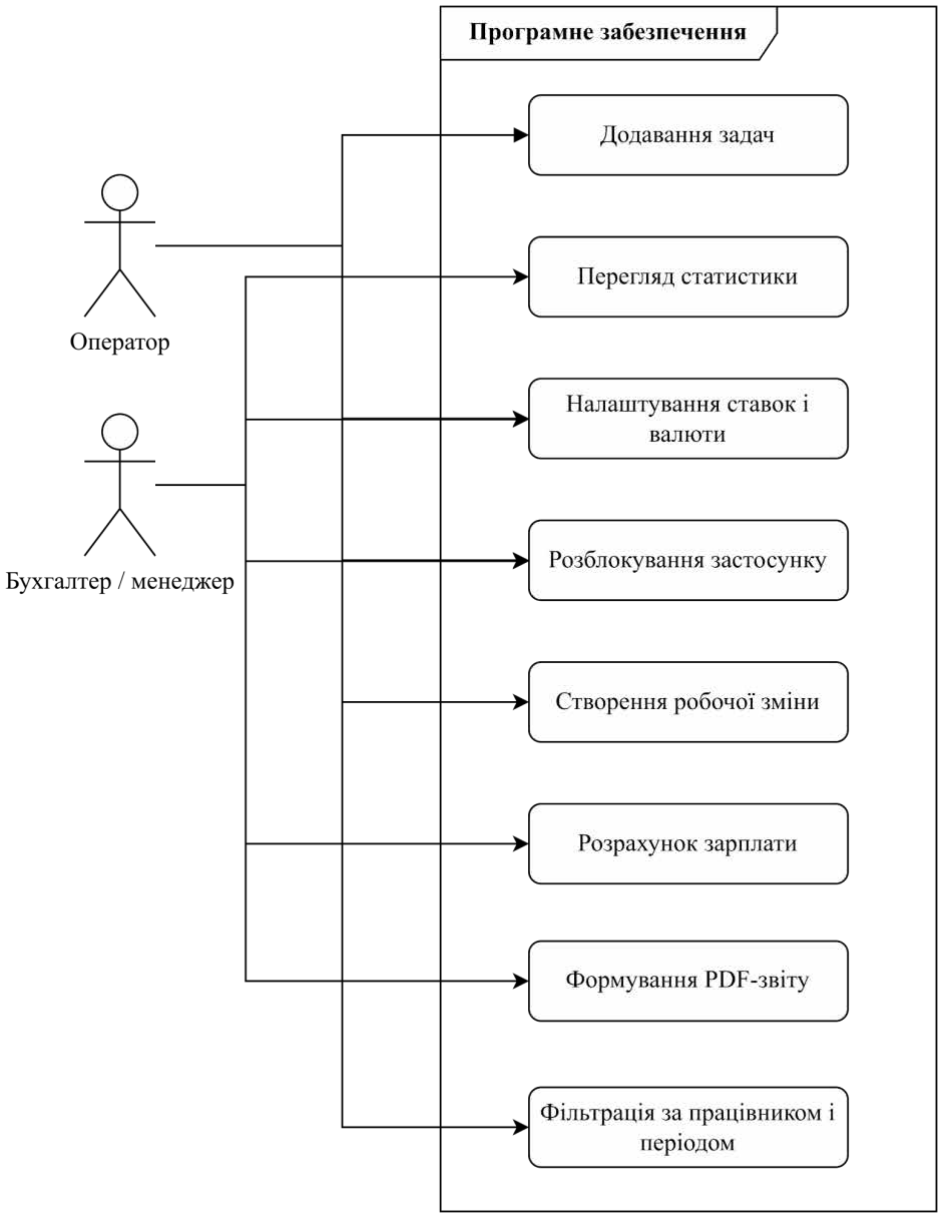


Рисунок 1.1. Діаграма прецедентів програмного забезпечення

На діаграмі показано, що користувач має доступ до основних функцій застосунку. Він може розблокувати застосунок, створити робочу зміну, додати задачі, виконати розрахунок заробітної плати, переглянути статистику, налаштувати ставки і валюту, сформувати PDF-звіт, а також застосувати фільтрацію за працівником і періодом. Такий набір прецедентів відповідає практичному призначенню системи та відображає головні дії, які користувач виконує під час роботи із застосунком.

Діаграма прецедентів демонструє загальні можливості програмного забезпечення без деталізації внутрішньої реалізації. Вона показує, які функції доступні користувачу та як вони пов’язані з основною метою системи, а саме з обліком робочого часу, внесенням виконаних задач, переглядом статистики і розрахунком заробітної плати.

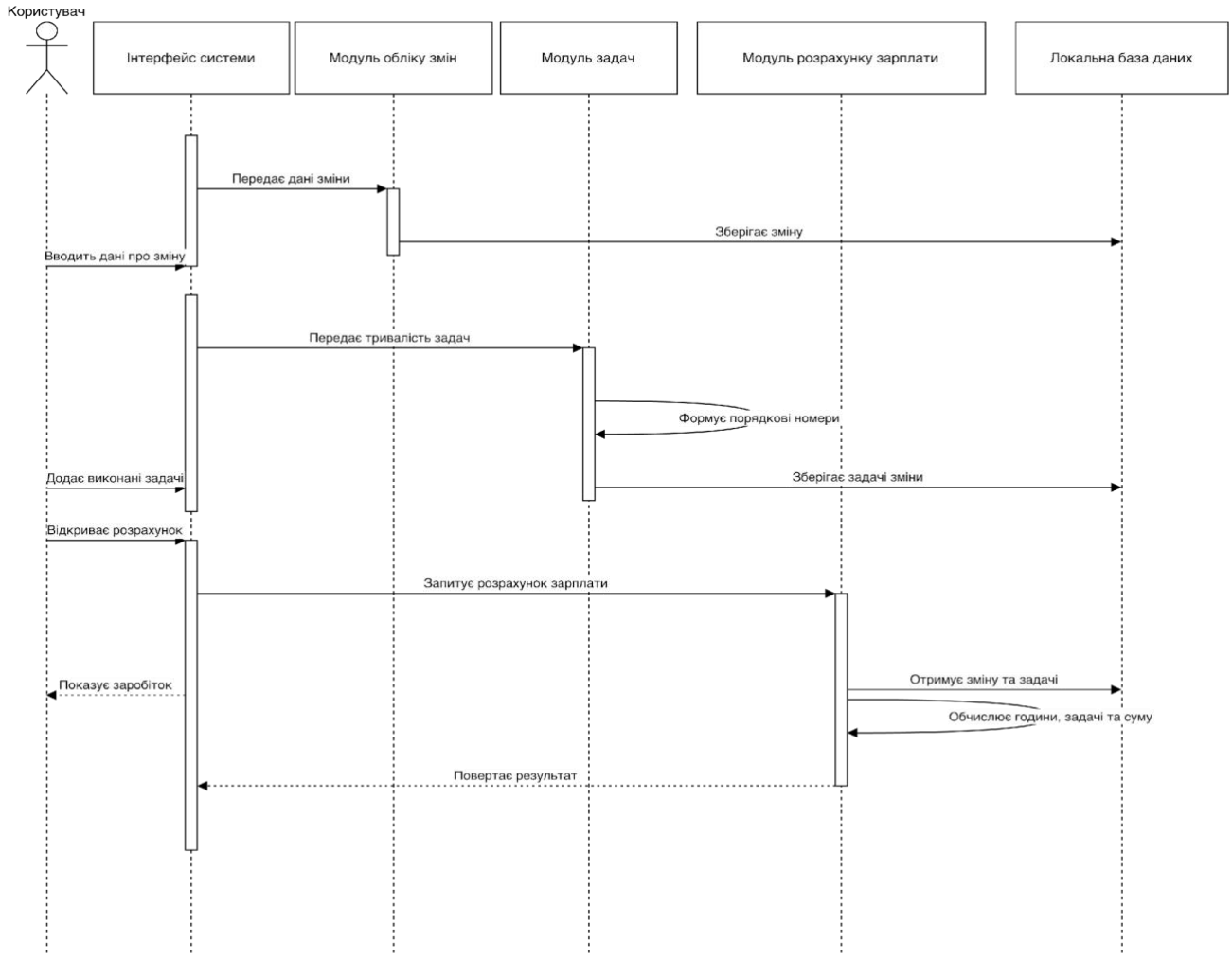


Рисунок 1.2. Діаграма послідовності розрахунку заробітної плати

Діаграма послідовності показує типовий сценарій роботи користувача із системою під час створення зміни, додавання задач і перегляду розрахунку заробітної плати. Спочатку користувач вводить дані про зміну через інтерфейс системи. Після цього інтерфейс передає ці дані до модуля обліку змін, а модуль зберігає зміну в локальній базі даних. Після створення зміни користувач додає виконані задачі. Інтерфейс передає тривалість задач до модуля задач, який

формує порядкові номери задач і зберігає їх у локальній базі даних. Далі користувач відкриває розрахунок, а інтерфейс надсилає запит до модуля розрахунку заробітної плати. Модуль отримує дані зміни та задач, обчислює години, кількість задач і підсумкову суму, після чого повертає результат в інтерфейс. У результаті користувач бачить суму заробітку за зміну.

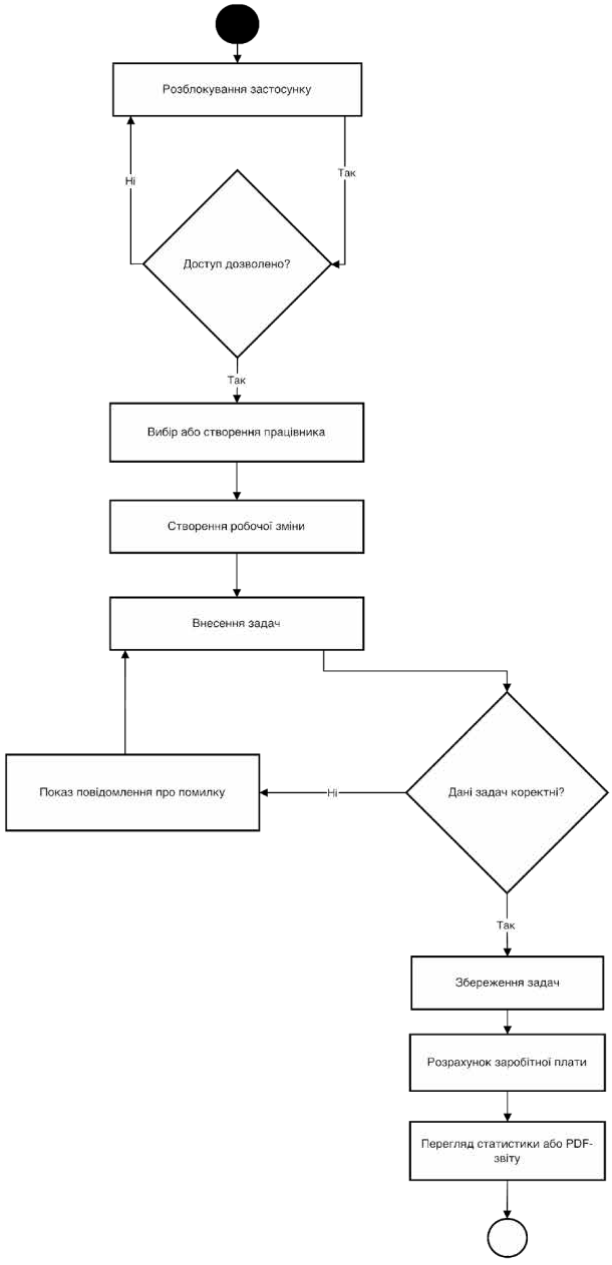


Рисунок 1.3. Діаграма діяльності основного сценарію роботи

Подана діаграма діяльності відображає логіку роботи на рівні предметної області. Вона не деталізує всі технічні дії, але показує основний шлях користувача від входу в застосунок до отримання результату. Така модель є

корисною для подальшого проєктування, оскільки дозволяє визначити, які екрани, модулі та перевірки повинні бути реалізовані.

1.5 Постановка завдання

На основі аналізу предметної області, огляду існуючих рішень і визначених вимог було сформульовано завдання бакалаврської кваліфікаційної роботи. Необхідно розробити мобільне програмне забезпечення для обліку робочого часу та розрахунку заробітної плати операторів логістичної компанії. Застосунок повинен забезпечувати ведення працівників, створення робочих змін, внесення виконаних задач, автоматичний розрахунок заробітку, перегляд статистики, налаштування параметрів і формування PDF-звіту.

Розроблювана система має враховувати специфіку роботи операторів логістичної компанії. Основними даними для розрахунку є працівник, тип зміни, тривалість роботи, погодинна ставка, кількість задач і тривалість кожної задачі. Система повинна автоматично визначати, скільки задач входить до базової частини, а скільки задач підлягає окремій оплаті. Також необхідно передбачити коректну роботу з нічними змінами, які можуть переходити через межу календарної доби.

Програмне забезпечення має мати зручний інтерфейс, орієнтований на щоденне використання. Головний екран повинен показувати підсумкові показники за вибраний період і підтримувати вибір конкретного працівника або режим загальної статистики. Списки працівників і змін мають бути компактними та зрозумілими, а список змін повинен підтримувати фільтрацію за працівником. Детальна інформація повинна відкриватися на окремих екранах, щоб не перевантажувати основні списки. Також необхідно реалізувати налаштування мови, валюти, ставок і безпеки.

Під час розроблення потрібно забезпечити локальне збереження даних, коректну роботу розрахункових алгоритмів і захист доступу до застосунку. Для захисту даних необхідно реалізувати PIN-код і підтримку біометричної автентифікації. Для підвищення практичної цінності системи слід додати

можливість формування PDF-звіту за вибраний період. Звіт має містити загальну статистику, таблицю змін і підсумкову суму нарахувань.

У результаті виконання роботи має бути отриманий мобільний застосунок, який дозволяє скоротити кількість ручних дій під час обліку змін і задач, зменшити ризик помилок у розрахунках і зробити процес визначення заробітної плати більш прозорим для користувача.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Проєктування інформаційного забезпечення починається з визначення основних сутностей предметної області та зв’язків між ними. Для розроблюваного програмного забезпечення ключовими сутностями є працівник, робоча зміна, задача, налаштування застосунку та правила тарифікації задач. Саме ці сутності зберігають дані, необхідні для обліку робочого часу та розрахунку заробітної плати.

Сутність «Працівник» описує оператора або іншу особу, для якої ведеться облік змін. Вона містить ім’я працівника, посаду, extension, статус активності та дату створення запису. Один працівник може мати багато робочих змін, але кожна зміна належить тільки одному працівнику. Такий зв’язок дозволяє переглядати історію роботи конкретного оператора та формувати статистику за його змінами.

Сутність «Робоча зміна» описує окремий період роботи працівника. Вона містить дату, час початку, час завершення, тип зміни, погодинну ставку, примітки та посилання на працівника. Робоча зміна також пов’язана із задачами, які були виконані в її межах. Одна зміна може містити багато задач, а кожна задача належить тільки одній зміні. Завдяки цьому система може правильно рахувати загальну кількість задач і визначати, які з них підлягають окремій оплаті.

Сутність «Задача» містить дані про виконану робочу дію в межах зміни. До її атрибутів належать ідентифікатор, порядковий номер або зовнішній номер задачі, дата створення, дата задачі, тип задачі, статус, тривалість у секундах та посилання на робочу зміну. У розроблюваному застосунку задачі вносяться вручну, а їхній порядковий номер формується автоматично в межах конкретної зміни. Це дозволяє уникнути помилок при введенні та забезпечує послідовність обліку.

Сутність «Налаштування застосунку» містить параметри, які впливають на розрахунок та відображення підсумкових сум. До таких параметрів належать погодинні ставки для ранкової, основної та нічної зміни, вибрана валюта відображення та кількість задач, що входять до базової частини оплати. Параметри мови, безпеки та біометричного доступу в реалізації застосунку обробляються окремими службовими компонентами, оскільки вони не є частиною розрахункової моделі заробітної плати. Окремо можна виділити сутність «Правило тарифікації», яка описує межі вартості задачі залежно від типу зміни. Такі правила потрібні для того, щоб система могла визначати суму оплати за задачу з урахуванням її тривалості.

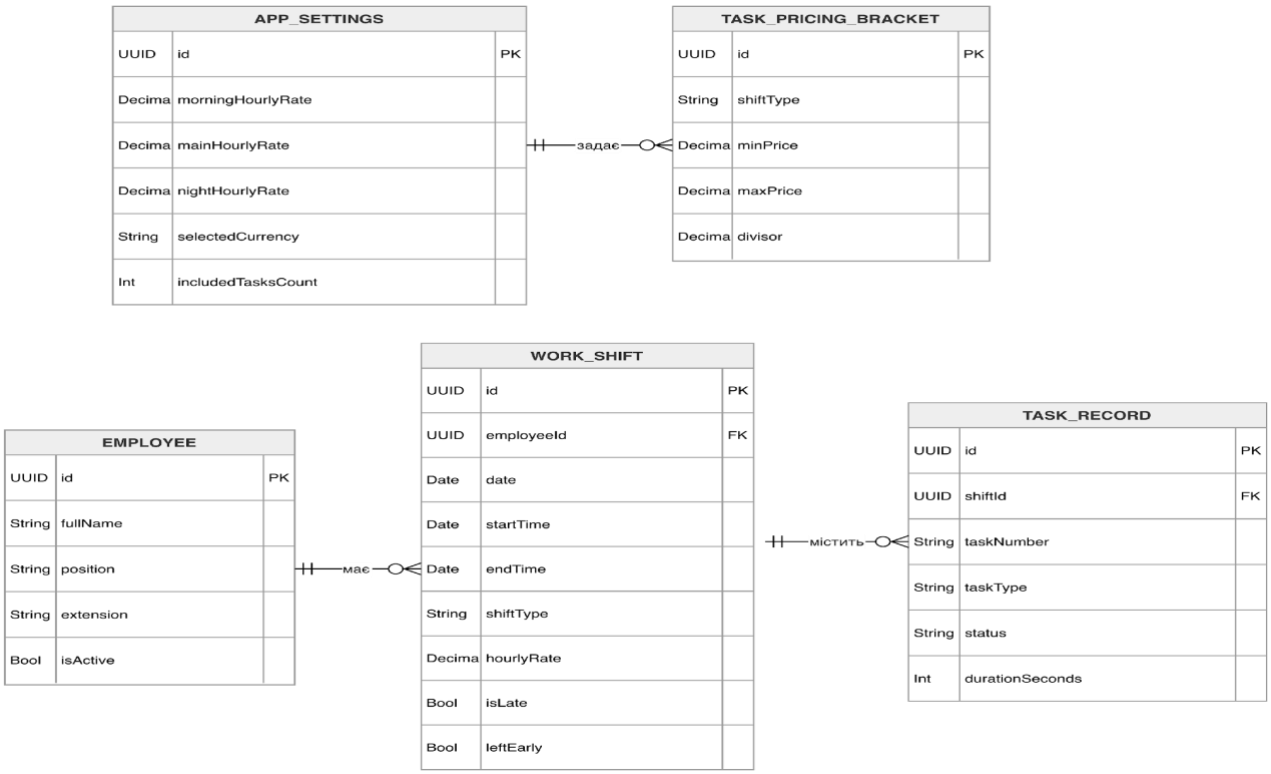


Рисунок 2.1. ER-діаграма програмного забезпечення

ER-діаграма показує основні сутності інформаційної бази програмного забезпечення та зв’язки між ними. Центральними сутностями є EMPLOYEE, WORK_SHIFT і TASK_RECORD. Працівник має робочі зміни, а кожна зміна належить конкретному працівнику. Робоча зміна містить задачі, які були виконані в її межах, тому TASK_RECORD не існує окремо від WORK_SHIFT. Така структура дає змогу коректно зберігати дані про працівників, зміни, задачі та використовувати їх під час розрахунку заробітної плати. Окремо на діаграмі показано APP_SETTINGS і TASK_PRICING_BRACKET. Налаштування застосунку містять ставки, вибрану валюту та кількість задач, що входять до базової частини оплати. Правила тарифікації задач задають мінімальну й максимальну вартість задачі залежно від типу зміни. Первинні ключі всіх сутностей представлені атрибутом id. Зовнішній ключ employeeId у сутності WORK_SHIFT забезпечує зв’язок зі сутністю EMPLOYEE, а shiftId у сутності TASK_RECORD забезпечує зв’язок із робочою зміною. Це дозволяє зберігати дані цілісно та уникати ситуацій, коли задача не має пов’язаної зміни.

2.2 Діаграма класів та кооперацій

Після побудови логічної моделі даних доцільно перейти до проєктування програмної структури. Діаграма класів відображає основні моделі, сервіси та зв’язки між ними. У застосунку можна виділити класи моделей, які відповідають сутностям предметної області, та службові класи, які реалізують бізнес-логіку. До моделей належать Employee, WorkShift, TaskRecord, AppSettings і TaskPricingBracket. До сервісів належать PayrollEngine, PricingEngine, DashboardOverviewService, PayrollReportService, PayrollPDFRenderer, SecurityManager і LanguageManager.

Клас Employee відповідає за збереження даних працівника. Він пов’язаний із класом WorkShift, оскільки один працівник може мати багато змін. Клас WorkShift містить інформацію про дату, час, тип зміни, ставку та пов’язані

задачі. Клас TaskRecord описує окрему задачу, її тривалість, статус і належність до зміни. Саме ці три класи є основою інформаційної моделі застосунку.

Клас PayrollEngine відповідає за розрахунок заробітної плати. Він використовує дані зміни та задач, визначає кількість відпрацьованих годин, погодинну оплату, кількість оплачуваних задач і підсумкову суму. Клас PricingEngine використовується для визначення вартості окремої задачі за її тривалістю та типом зміни. DashboardOverviewService формує агреговані показники для головного екрана. PayrollReportService збирає дані для звіту, а PayrollPDFRenderer відповідає за створення PDF-документа.

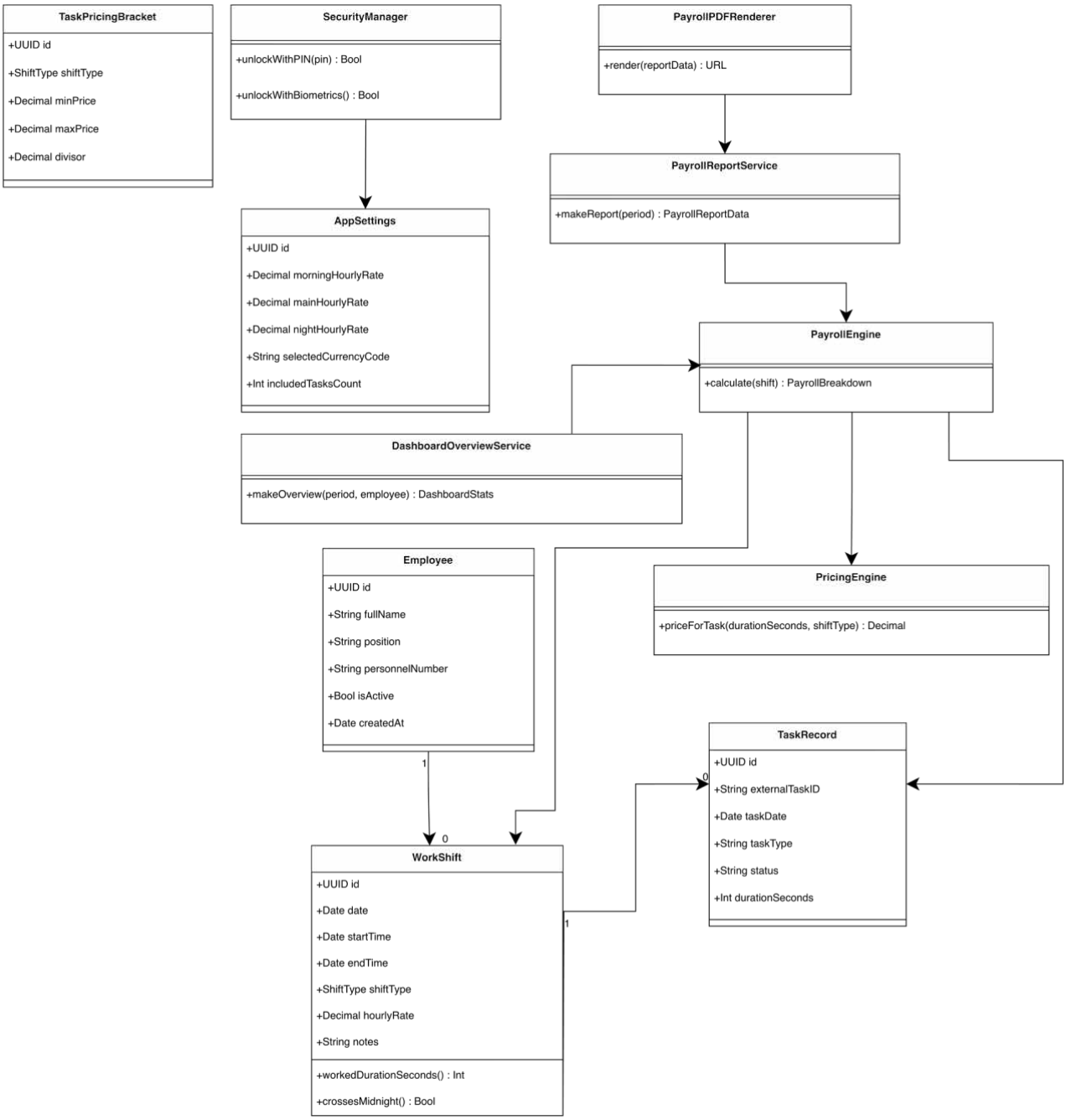


Рисунок 2.2. Діаграма класів програмного забезпечення

Діаграма класів показує, що розрахункова логіка відокремлена від інтерфейсу користувача. Це важливо для підтримки застосунку, оскільки зміна правил розрахунку не потребує повного переписування екранів. PayrollEngine виконує основні обчислення, PricingEngine відповідає за ціну задач, а сервіси огляду та звітності використовують готові результати для відображення статистики або формування PDF-звіту.

Кооперація класів проявляється під час виконання основного сценарію роботи. Користувач створює зміну на екрані інтерфейсу, після чого створюється об’єкт WorkShift. Далі користувач додає задачі, і система формує об’єкти TaskRecord. Коли потрібно показати заробіток, PayrollEngine отримує зміну та задачі, звертається до PricingEngine і повертає результат у вигляді структури розрахунку. Якщо користувач формує звіт, PayrollReportService збирає дані за період і передає їх до PayrollPDFRenderer.

2.3 Діаграма пакетів програмного забезпечення

Для впорядкування структури програмного забезпечення застосунок поділено на логічні пакети. Такий поділ спрощує підтримку проєкту, дозволяє швидше знаходити потрібні файли та зменшує залежність між окремими частинами системи. У межах застосунку можна виділити пакети App, Views, Models, Services, Components, Localization і Utilities.

Пакет App містить точку входу до застосунку, налаштування контейнера моделей і підключення загального середовища. Пакет Views містить екрани користувацького інтерфейсу, зокрема головний екран, список працівників, список змін, деталі зміни, налаштування, екран безпеки та екран звітів. Пакет Models містить основні моделі даних, які відповідають сутностям предметної області. Пакет Services містить бізнес-логіку, зокрема розрахунок заробітної плати, обробку задач, формування статистики, створення звітів і роботу з курсами валют. Пакет Components містить повторно використовувані UI-компоненти, а пакет Localization відповідає за підтримку кількох мов. Пакет Utilities містить допоміжні засоби, форматери та налаштування збереження даних.

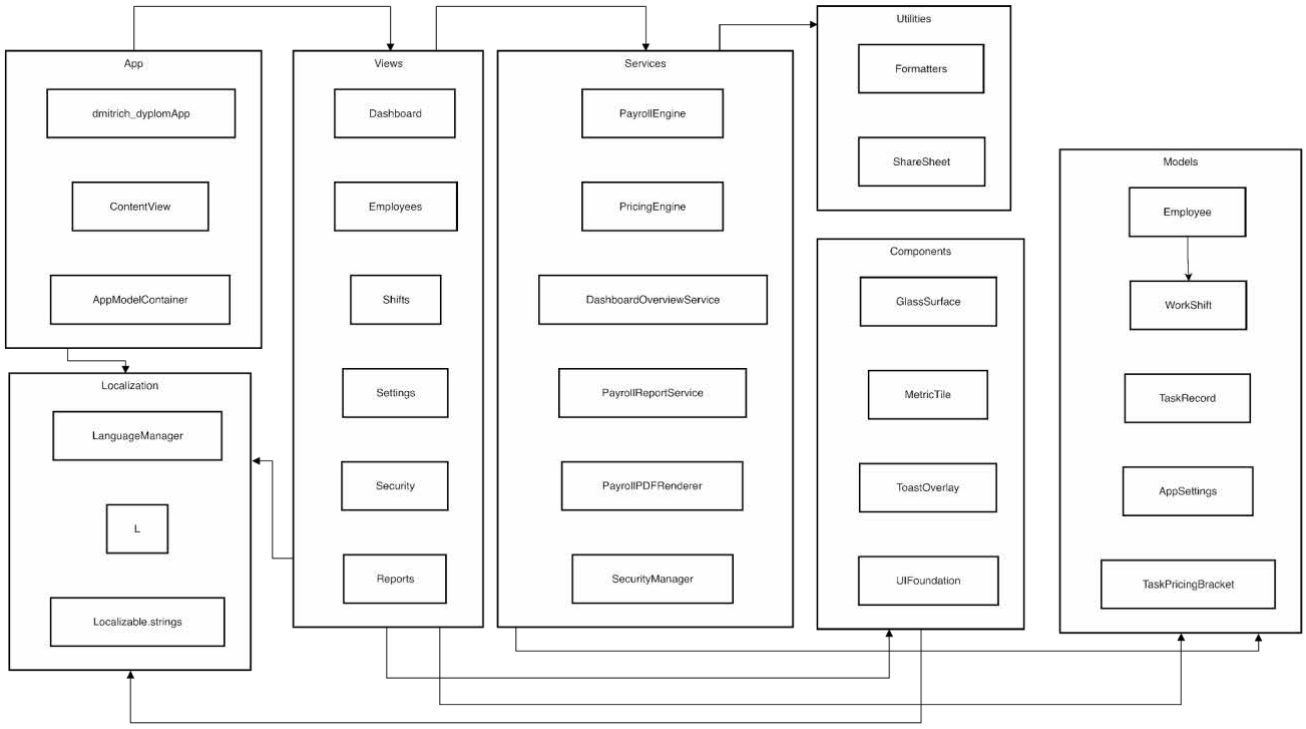


Рисунок 2.3. Діаграма пакетів програмного забезпечення

Діаграма пакетів показує, що найбільша кількість залежностей спрямована від екранів до моделей, сервісів і компонентів. Це природно для мобільного застосунку, оскільки інтерфейс користувача відображає дані та викликає службові модулі. Водночас моделі не повинні залежати від екранів, а сервіси не повинні залежати від конкретного зовнішнього вигляду інтерфейсу. Такий підхід полегшує тестування та подальший розвиток застосунку.

Пакет Components має важливе значення для єдності дизайну. Саме в ньому зосереджуються загальні елементи оформлення, наприклад glass-карточки, плитки статистики, toast-повідомлення та кольорова система. Завдяки цьому зовнішній вигляд різних екранів можна підтримувати в одному стилі. Якщо потрібно змінити фон, відступи або вигляд карток, це краще робити через спільні компоненти, а не окремо в кожному екрані.

2.4 Діаграма компонентів програмного забезпечення

Діаграма компонентів відображає програмне забезпечення як набір взаємопов’язаних частин. У розроблюваній системі можна виділити три основні групи компонентів: сторону користувача, основну логіку системи та рівень

даних і зовнішніх служб. Такий поділ дозволяє зрозуміти, які частини відповідають за інтерфейс, які виконують розрахунки, а які забезпечують збереження та обмін даними.

Сторона користувача містить інтерфейс застосунку, екрани, налаштування, локалізацію та елементи візуального оформлення. Через ці компоненти користувач створює працівників, зміни, задачі, переглядає статистику та формує звіти. Основна логіка системи містить модуль розрахунку зарплати, модуль задач, модуль обліку змін, модуль статистики, модуль PDF-звітів, модуль перевірки даних і модуль безпеки. Рівень даних містить локальне сховище, моделі даних, налаштування, курси валют і механізм експорту PDF-файлів.

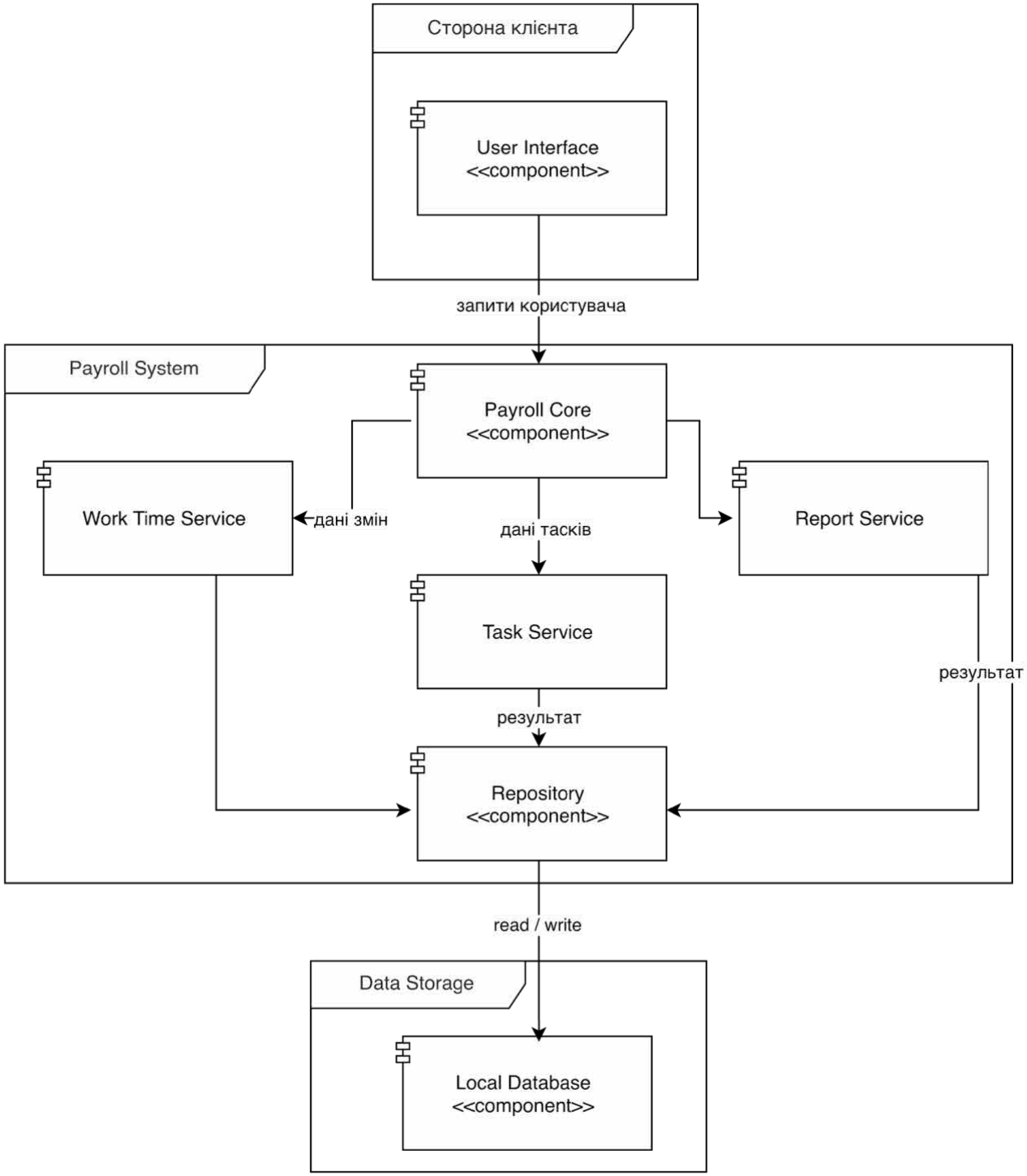


Рисунок 2.4. Діаграма компонентів програмного забезпечення

Діаграма компонентів показує загальну структуру програмного забезпечення та взаємодію його основних частин. На стороні клієнта розміщено компонент User Interface, через який користувач працює із застосунком і надсилає запити до системи. Основна логіка системи представлена компонентом Payroll Core, який координує роботу модулів обліку змін, задач і звітів.

Компонент Work Time Service відповідає за обробку даних робочих змін, Task Service використовується для роботи із задачами, а Report Service формує результати для звітності. Дані, отримані від цих компонентів, передаються до Repository, який забезпечує обмін інформацією з локальним сховищем. Компонент Data Storage містить Local Database, у якій зберігаються працівники, робочі зміни, задачі, налаштування та інші дані застосунку. Така структура показує, що інтерфейс користувача не працює напряму з базою даних, а звертається до неї через службові компоненти системи. Це робить архітектуру більш зрозумілою та зручною для подальшої підтримки.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Розроблення програмного забезпечення для обліку робочого часу та розрахунку заробітної плати операторів логістичної компанії передбачає створення надійної інформаційної бази, у якій зберігаються працівники, робочі зміни, виконані задачі, ставки та налаштування. Для цього застосунку було обрано локальний підхід до збереження даних, оскільки основні операції виконуються без необхідності постійного підключення до сервера. Користувач працює із застосунком на власному пристрої, а дані про зміни та розрахунки мають бути доступними одразу після відкриття програми.

Як основу для управління інформаційною базою використано SwiftData. Цей інструмент є частиною сучасної екосистеми iOS-розробки та дає змогу описувати моделі даних безпосередньо мовою Swift. Для застосунку це зручно, оскільки об’єкти предметної області, зокрема працівник, зміна, задача та налаштування, можуть бути представлені у вигляді моделей, які зберігаються локально та використовуються в інтерфейсі без додаткового перетворення в окремі табличні структури.

У застосунку локальна база даних підключається через контейнер моделей. Такий контейнер визначає, які саме сутності беруть участь у збереженні, і забезпечує доступ до них через контекст даних. Контекст використовується під час створення нових працівників, додавання робочих змін, внесення задач, редагування записів, видалення зміни та оновлення налаштувань. Завдяки цьому робота з даними відбувається централізовано, а різні екрани застосунку не повинні самостійно керувати фізичним збереженням інформації.

До інформаційної бази входять моделі Employee, WorkShift, TaskRecord, AppSettings і TaskPricingBracket. Модель Employee відповідає за дані працівника. Модель WorkShift описує конкретну робочу зміну, її дату, тип, час початку, час завершення, ставку та додаткові ознаки відвідуваності. Модель TaskRecord

використовується для збереження задач, виконаних у межах конкретної зміни. AppSettings містить параметри ставок, вибрану валюту та кількість задач, що включаються до базової частини оплати. TaskPricingBracket використовується для опису правил тарифікації задач залежно від типу зміни.

Важливою властивістю інформаційної бази є збереження зв’язків між об’єктами. Працівник може мати багато змін, а кожна зміна належить певному працівнику. Робоча зміна може містити багато задач, але кожна задача належить тільки одній зміні. Для задач, пов’язаних зі зміною, використовується каскадне видалення. Це означає, що під час видалення робочої зміни пов’язані з нею задачі також видаляються, тому в базі не залишаються записи, які вже не мають власної зміни.

Локальне збереження даних відповідає практичному призначенню застосунку. Програма орієнтована на щоденне внесення змін і задач, швидкий перегляд статистики та формування звіту. Для таких сценаріїв локальна база є достатньою, оскільки забезпечує швидкий доступ до інформації, не потребує окремого серверного середовища та зменшує залежність від стану мережі. У майбутньому цю архітектуру можна розширити хмарною синхронізацією, але для базової реалізації локальний підхід є більш простим і стабільним.

3.2 Розробка інформаційної бази програмного забезпечення

Розробка інформаційної бази починається з визначення моделей, які відповідають об’єктам предметної області. Основним об’єктом є працівник. У застосунку він описується моделлю Employee, яка містить унікальний ідентифікатор, повне ім’я, посаду, extension, ознаку активності та дату створення запису. Окремо зберігається зв’язок зі змінами працівника. Це дає змогу швидко відфільтрувати статистику або список змін за конкретним працівником, а також показувати загальні підсумки по всіх працівниках.

Модель WorkShift є центральною для розрахунку заробітної плати, оскільки саме зміна об’єднує час роботи, тип зміни, ставку, працівника та виконані задачі. Вона містить дату, час початку, час завершення, тип зміни, тип

погодинної активності, погодинну ставку, примітки, ознаки запізнення або раннього завершення, пояснення та дату створення запису. Якщо час завершення менший за час початку, система нормалізує завершення зміни та переносить його на наступну добу. Завдяки цьому нічні зміни, які починаються ввечері й завершуються зранку, обробляються коректно.

У моделі WorkShift також передбачено розрахункові властивості. Вони дозволяють визначати фактичний час завершення, тривалість роботи в секундах, перехід зміни через північ, наявність порушень відвідуваності та коефіцієнт погодинної оплати. Наявність таких властивостей спрощує розрахунок заробітної плати, оскільки модуль розрахунку може отримати вже підготовлені значення, а не виконувати однакові перевірки в кожному місці інтерфейсу.

Модель TaskRecord використовується для опису задачі, яка була виконана в межах зміни. У ній зберігаються ідентифікатор, номер задачі, дата створення, дата виконання, тип задачі, статус, тривалість у секундах, додатковий час продажу, ім’я агента, початковий текстовий блок і зв’язок зі зміною. У фінальній логіці застосунку основний сценарій передбачає ручне внесення задач. Користувач не вводить порядковий номер вручну, оскільки система визначає наступний номер у межах конкретної зміни.

Для задач реалізовано перевірку придатності до розрахунку. Задача повинна мати допустимий тип і статус, що відповідає правилам оплати. Основним оплачуваним типом є LOG EDITING зі статусом Completed. Додатково система враховує, що задача повинна належати до часових меж конкретної зміни. Такий підхід запобігає випадковому включенню сторонніх або некоректних задач до розрахунку зарплати.

Модель AppSettings зберігає параметри, що впливають на розрахунок і відображення результатів. До них належать погодинні ставки для ранкової, основної та нічної зміни, код вибраної валюти, кількість задач, які включаються до базової частини оплати, дата створення та дата оновлення налаштувань. За замовчуванням кількість включених задач дорівнює п’ятнадцяти. Це значення

використовується під час визначення, які задачі не оплачуються окремо, а які потрапляють до додаткової оплати.

Модель TaskPricingBracket призначена для опису меж вартості задач. У ній зберігається тип зміни, мінімальна вартість, максимальна вартість і значення, яке використовується під час розрахунку ціни за тривалістю. На практиці ціна задачі визначається на основі кількості секунд, після чого обмежується мінімальним і максимальним значенням відповідно до типу зміни. Для ранкової та основної зміни використовуються одні межі, для нічної зміни інші.

Окремі параметри застосунку не потребують збереження в основній інформаційній базі. Наприклад, PIN-код не зберігається у відкритому вигляді в SwiftData. Для нього використовується захищене сховище Keychain, де зберігається хеш-значення. Параметри біометричної автентифікації та автоблокування пов’язані з механізмами безпеки застосунку. Обрана мова інтерфейсу керується окремим менеджером локалізації, а курси валют отримуються через сервіс обміну та можуть кешуватися для повторного використання.

Такий поділ даних дозволяє не перевантажувати основну модель налаштувань і зберігати кожен тип інформації у відповідному місці. Дані предметної області зберігаються в SwiftData, захищені дані доступу зберігаються в Keychain, а службові параметри інтерфейсу обробляються відповідними менеджерами. Завдяки цьому структура інформаційної бази залишається зрозумілою, а застосунок легше підтримувати та розвивати.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Для створення прикладного програмного забезпечення було використано інструменти екосистеми Apple, оскільки застосунок орієнтований на роботу на пристроях з операційною системою iOS. Основною мовою програмування обрано Swift. Вона добре підходить для розроблення мобільних застосунків,

підтримує сучасні підходи до роботи з типами даних, асинхронними операціями, локальним збереженням і побудовою інтерфейсу.

Користувацький інтерфейс реалізовано за допомогою SwiftUI. Цей фреймворк дозволяє описувати екрани декларативно, тобто через структуру станів і компонентів. Для застосунку це важливо, оскільки дані працівників, змін, задач і налаштувань постійно змінюються, а інтерфейс має автоматично реагувати на ці зміни. Наприклад, після додавання задачі змінюється розрахунок зміни, після видалення зміни оновлюється список, а після вибору працівника або періоду перераховується статистика.

Розроблення виконувалося в середовищі Xcode 26.5 на MacBook Air M2 з операційною системою macOS Tahoe 26.4.1. Тестування застосунку проводилося на iPhone 15 Pro з iOS 26.4.2. Така конфігурація дозволила перевірити роботу інтерфейсу на реальному розмірі екрана, оцінити зручність використання нижньої навігації, перевірити введення даних, роботу біометричної автентифікації та формування PDF-звіту.

Для локального збереження даних використано SwiftData. Цей інструмент забезпечує роботу з моделями Employee, WorkShift, TaskRecord, AppSettings і TaskPricingBracket. У застосунку SwiftData використовується для створення, читання, оновлення та видалення даних. Завдяки цьому користувач може вести облік працівників, створювати зміни, додавати задачі, видаляти непотрібні записи та отримувати актуальні підсумки без підключення до зовнішнього сервера.

Для захисту доступу до застосунку використано механізми Keychain та LocalAuthentication. Keychain застосовується для безпечного збереження даних, пов’язаних із PIN-кодом. LocalAuthentication дає змогу використовувати Face ID або Touch ID, якщо пристрій підтримує відповідну функцію. Такий підхід поєднує зручність щоденного використання з базовим рівнем захисту службової інформації.

Для отримання курсів валют використовується мережевий запит до API Національного банку України. Застосунок може відображати підсумкові суми в доларах США, гривні або євро. Основні розрахунки залишаються сталими, а вибрана валюта впливає лише на спосіб показу підсумкової суми. Це дозволяє не змінювати логіку нарахування при зміні валюти та водночас робить результати більш зручними для користувача.

Формування PDF-звіту реалізовано через окремий модуль звітності. Він збирає дані за вибраний період, обчислює підсумкові показники, формує таблицю змін і передає їх до засобу генерації PDF. Для експорту сформованого документа використовується стандартне системне вікно iOS, через яке файл можна зберегти, надіслати або відкрити в іншому застосунку. PDF-звіт оформлено у світлому стилі: білий фон, чорний або темно-сірий текст, світло-сірі блоки та таблиця з чіткими межами.

Для підтримки двомовного інтерфейсу використано механізм локалізації. У застосунку передбачено українську та англійську мови. Локалізація охоплює назви екранів, кнопки, повідомлення, підписи полів, типи змін і службові тексти. Це робить застосунок більш гнучким і придатним для використання різними користувачами.

Окрему роль відіграють спільні компоненти інтерфейсу. Для карток, плиток статистики, сповіщень, фону та кольорової системи використовуються загальні елементи. Завдяки цьому різні екрани мають однаковий стиль, а зміни в оформленні можна виконувати централізовано. Такий підхід особливо важливий для підтримки світлої й темної теми, оскільки текст, блоки та фон повинні залишатися читабельними в обох режимах.

3.4 Алгоритмізація та програмування програмних модулів

Алгоритмізація програмних модулів виконувалася з урахуванням основних сценаріїв роботи користувача. Застосунок повинен дозволяти створити працівника, створити зміну, внести задачі, виконати розрахунок, переглянути статистику, сформувати PDF-звіт і захистити доступ до даних. Для кожного

сценарію було визначено послідовність дій, необхідні перевірки та результат, який має отримати користувач.

Першим важливим алгоритмом є створення робочої зміни. Користувач відкриває форму нової зміни, обирає працівника, дату та тип зміни. Після вибору типу зміни система може автоматично підставити типовий час: ранкова зміна з 07:30 до 15:00, основна зміна з 15:00 до 23:30, нічна зміна з 23:30 до 07:30 наступного дня. Користувач може відредагувати ці значення вручну, якщо фактичний час відрізняється від шаблону.

Під час збереження зміни система перевіряє, чи вибрано працівника, чи коректно задано дату, час і ставку. Якщо час завершення менший за час початку, зміна нормалізується як така, що завершується наступного дня. Після успішної перевірки створюється або оновлюється об’єкт WorkShift, який зберігається в локальній базі. Список змін і статистика автоматично оновлюються на основі нових даних.

Рисунок 3.1. Алгоритм створення робочої зміни

Другим важливим алгоритмом є додавання задачі до зміни. У фінальній версії застосунку користувач додає задачі вручну. Система визначає наступний порядковий номер у межах поточної зміни, тому користувачу не потрібно вводити його самостійно. Це зменшує ризик помилок і забезпечує послідовність нумерації задач.

Користувач вводить тривалість задачі у текстовому полі або за допомогою вибору годин, хвилин і секунд. Після застосування тривалості система перевіряє, чи вона більша за нуль і чи може бути збережена як кількість секунд. Якщо дані коректні, створюється запис TaskRecord зі статусом Completed і типом задачі, який використовується в розрахунку зарплати. Після видалення задачі система повинна зберегти послідовність номерів, щоб у списку не залишалося пропусків.

Рисунок 3.2. Алгоритм додавання задачі до зміни

Основним розрахунковим модулем є PayrollEngine. Він отримує робочу зміну, пов’язані з нею задачі та налаштування ставок. Спочатку система визначає фактичну тривалість зміни в секундах і переводить її в години. Після цього обчислюється погодинна частина оплати. Якщо для зміни позначено запізнення або раннє завершення без прийнятного пояснення, погодинна оплата може бути зменшена відповідно до правил застосунку.

Далі система обробляє задачі. До розрахунку потрапляють лише задачі з допустимим типом і статусом, які належать до часових меж зміни. Задачі сортуються, після чого перші п’ятнадцять придатних задач вважаються включеними до базової частини оплати. Вони відображаються в розрахунку, але

мають нульову окрему вартість. Задачі, що перевищують базовий ліміт, передаються до PricingEngine для визначення вартості.

Вартість задачі визначається за її тривалістю. Початкова ціна розраховується як тривалість у секундах, поділена на 240, після чого результат округлюється до двох знаків. Для ранкової та основної зміни ціна обмежується діапазоном від 1,10 до 1,70 USD. Для нічної зміни використовується діапазон від 1,15 до 2,10 USD. Підсумкова сума за зміну складається з погодинної частини та оплати задач понад базовий ліміт.

Рисунок 3.3. Алгоритм розрахунку заробітної плати

Модуль статистики використовує результати розрахунків для формування підсумків за вибраний період. Користувач може обрати сім днів, місяць, три місяці або рік. Якщо активний період натиснути повторно, вибір знімається, а статистика показується за весь період. Додатково користувач може обрати конкретного працівника. Якщо працівник не вибраний, система відображає

загальну статистику по всіх працівниках. Якщо працівник вибраний, усі показники рахуються тільки за його змінами.

Фільтрація за працівником і фільтрація за періодом працюють незалежно одна від одної. Вибір працівника не скидає період, а вибір періоду не скидає працівника. Завдяки цьому користувач може швидко переглянути, наприклад, заробіток конкретного оператора за місяць або загальну статистику по всіх працівниках за весь час. Такий підхід робить головний екран більш практичним, оскільки він не обмежується лише загальними підсумками.

Окремий алгоритм реалізовано для формування PDF-звіту. Користувач відкриває екран звітів, обирає період і натискає кнопку формування документа. Сервіс звітності отримує зміни за вибраний проміжок часу, виконує розрахунок для кожної зміни, підраховує загальні показники та формує структуру даних для документа. Після цього PDF Renderer створює файл зі світлим оформленням, у якому містяться інформаційні блоки, загальна статистика, таблиця змін і підсумковий текст.

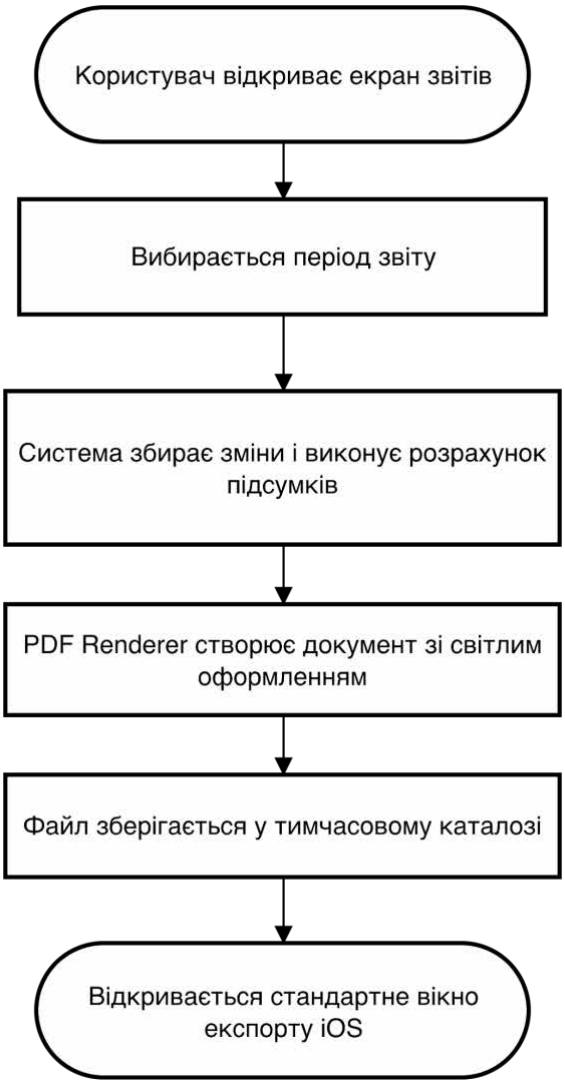


Рисунок 3.4. Алгоритм формування PDF-звіту

Після створення PDF-файлу застосунок відкриває стандартне системне вікно експорту. Через нього користувач може зберегти документ у Files, надіслати його через месенджер, передати поштою або відкрити в іншому застосунку. Це важливо для практичного використання, оскільки підсумки розрахунку можна не лише переглянути в застосунку, а й передати іншій особі або зберегти як окремий документ.

Модуль безпеки відповідає за обмеження доступу до застосунку. Користувач може встановити PIN-код і за потреби ввімкнути біометричне розблокування. PIN-код не зберігається у відкритому вигляді. Під час входу система порівнює введені дані з безпечно збереженим значенням і лише після успішної перевірки відкриває основний інтерфейс. Також передбачено

автоматичне блокування після згортання застосунку, що зменшує ризик доступу сторонньої особи до даних про зарплату.

У програмних модулях використано розділення відповідальності. Екрани відповідають за взаємодію з користувачем, моделі зберігають дані, сервіси виконують розрахунки, а спільні компоненти забезпечують однаковий вигляд інтерфейсу. Завдяки цьому застосунок можна розвивати без повного переписування. Наприклад, зміна формули вартості задачі потребує оновлення розрахункового модуля, а не всіх екранів, де відображається зарплата.

3.5 Опис інтерфейсу користувача

Інтерфейс користувача розроблено з урахуванням щоденного використання застосунку. Основна мета полягала в тому, щоб користувач міг швидко переглянути заробіток, створити працівника, додати зміну, внести задачі та сформувати звіт без зайвих переходів. Для цього використано нижню навігацію з основними розділами, компактні картки, зрозумілі підписи та єдиний візуальний стиль у темній і світлій темі.

Головний екран виконує роль панелі управління. На ньому показується заробіток за вибраний період, кількість відпрацьованих годин, кількість змін, загальна кількість задач, кількість оплачуваних задач і середній заробіток за зміну. У верхній частині екрана розміщено фільтр періоду. Також передбачено вибір працівника: користувач може переглядати загальну статистику або статистику тільки одного оператора.

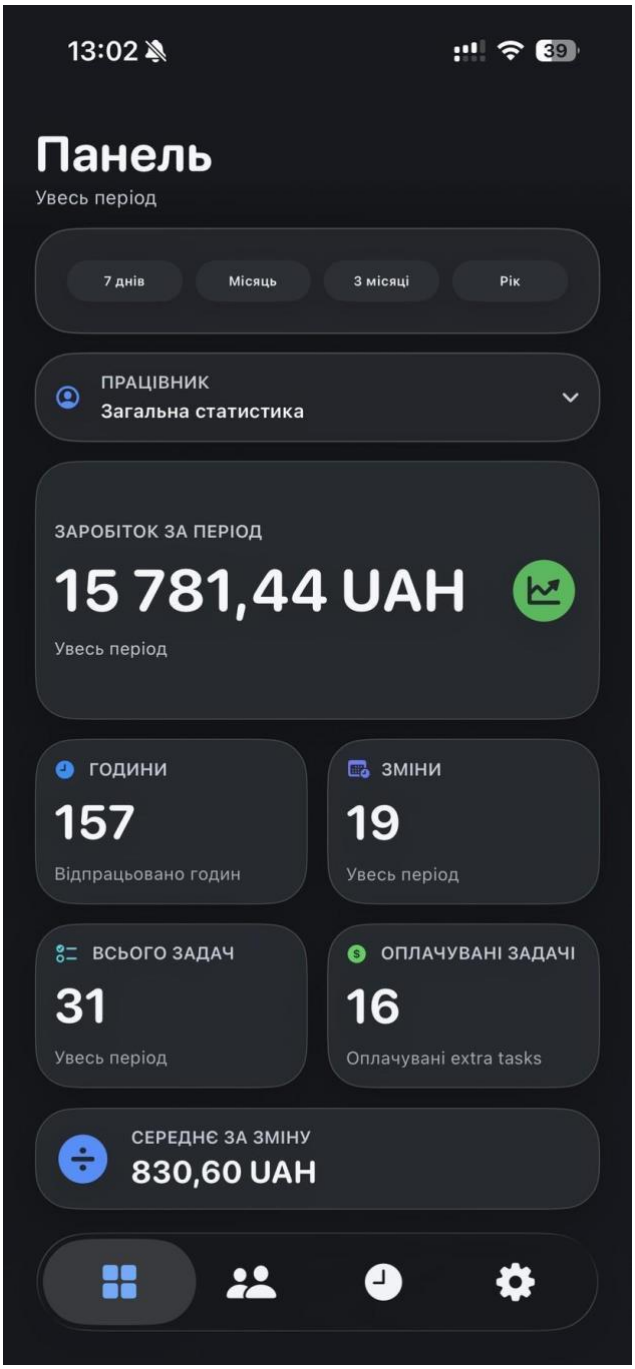


Рисунок 3.5. Головна панель зі статистикою за вибраний період

Екран працівників призначений для перегляду та керування працівниками, для яких ведеться облік змін. У картці працівника відображаються ім’я, посада, extension і статус. Такий набір інформації є достатнім для швидкої ідентифікації оператора та не перевантажує список зайвими полями. Активний працівник може бути використаний під час створення нової зміни або під час фільтрації статистики.

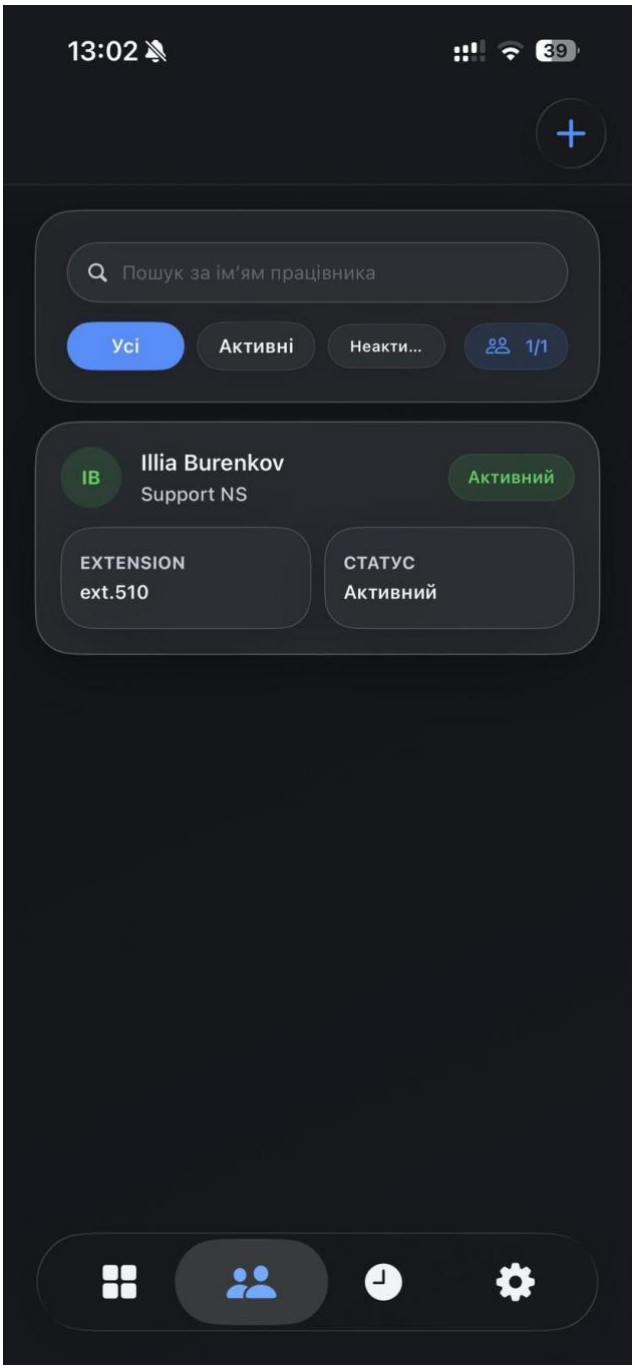


Рисунок 3.6. Екран працівників

Екран змін містить список робочих змін за вибраний період і вибраного працівника. Якщо працівник не вибраний, показуються зміни всіх працівників. Якщо користувач обирає конкретного працівника, список оновлюється та показує лише його зміни. Картка зміни має компактний вигляд і містить головні дані: працівника, дату, тип зміни, кількість задач і суму заробітку за зміну.

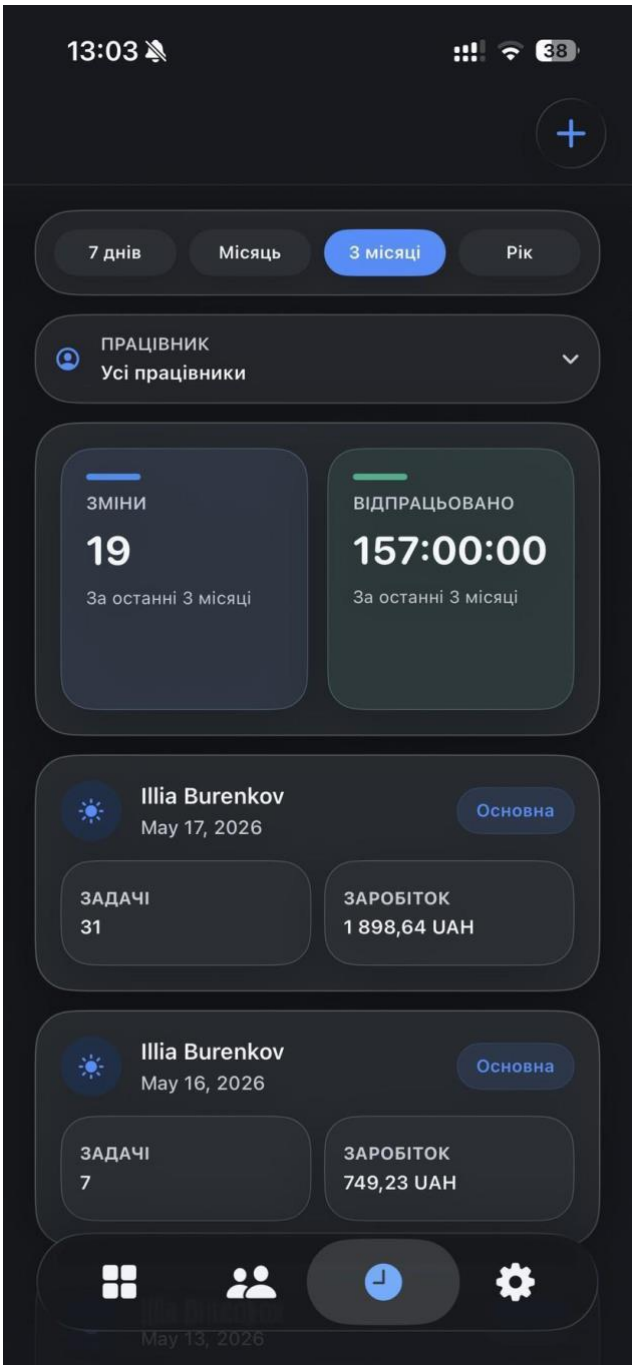


Рисунок 3.7. Екран змін із фільтром періоду та працівника

Після відкриття конкретної зміни користувач бачить детальну інформацію про неї. На екрані деталей відображаються основні параметри зміни, розрахунок оплати, кількість доданих задач, кількість базових задач, оплачувані задачі та підсумкова сума. Саме на цьому екрані користувач може перейти до додавання задач або переглянути вже внесені записи.

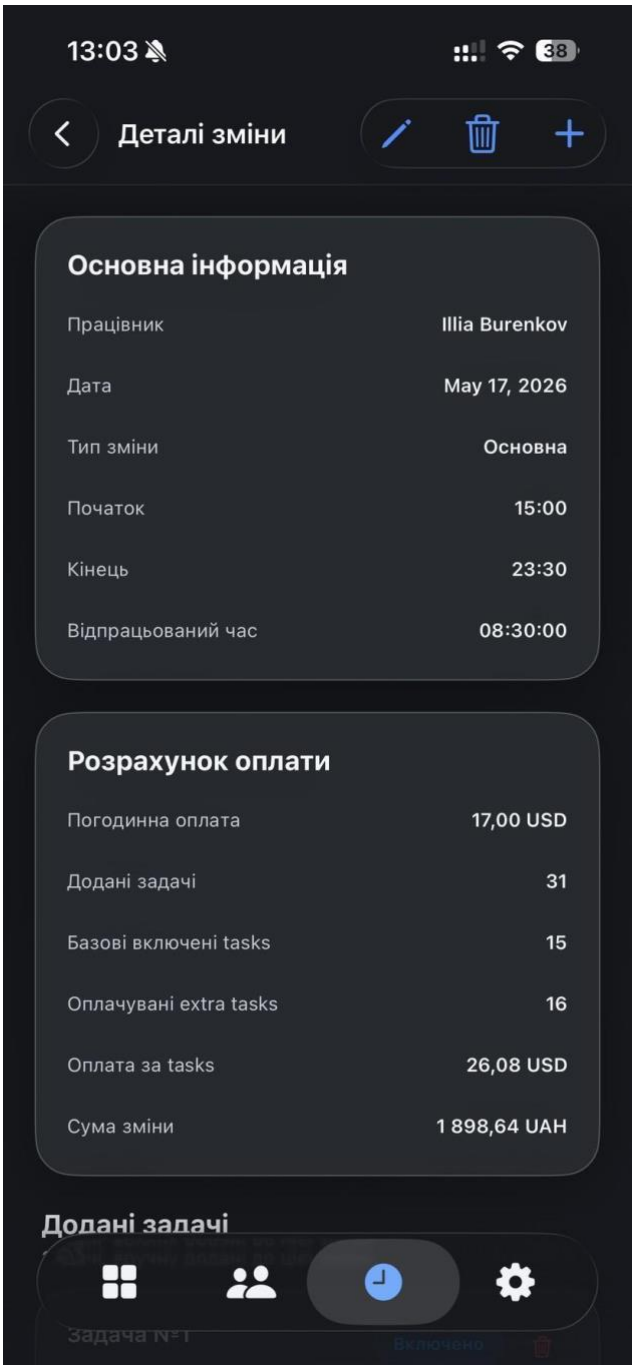


Рисунок 3.8. Екран деталей робочої зміни

Список задач у межах зміни показує кожну внесену задачу, її номер, статус і результат оплати. Якщо задача входить до базових п’ятнадцяти, вона позначається як включена в базову частину. Якщо задача перевищує цей ліміт, для неї відображається окрема вартість. Такий підхід робить розрахунок прозорим і дозволяє користувачу зрозуміти, чому саме сформувалася підсумкова сума.

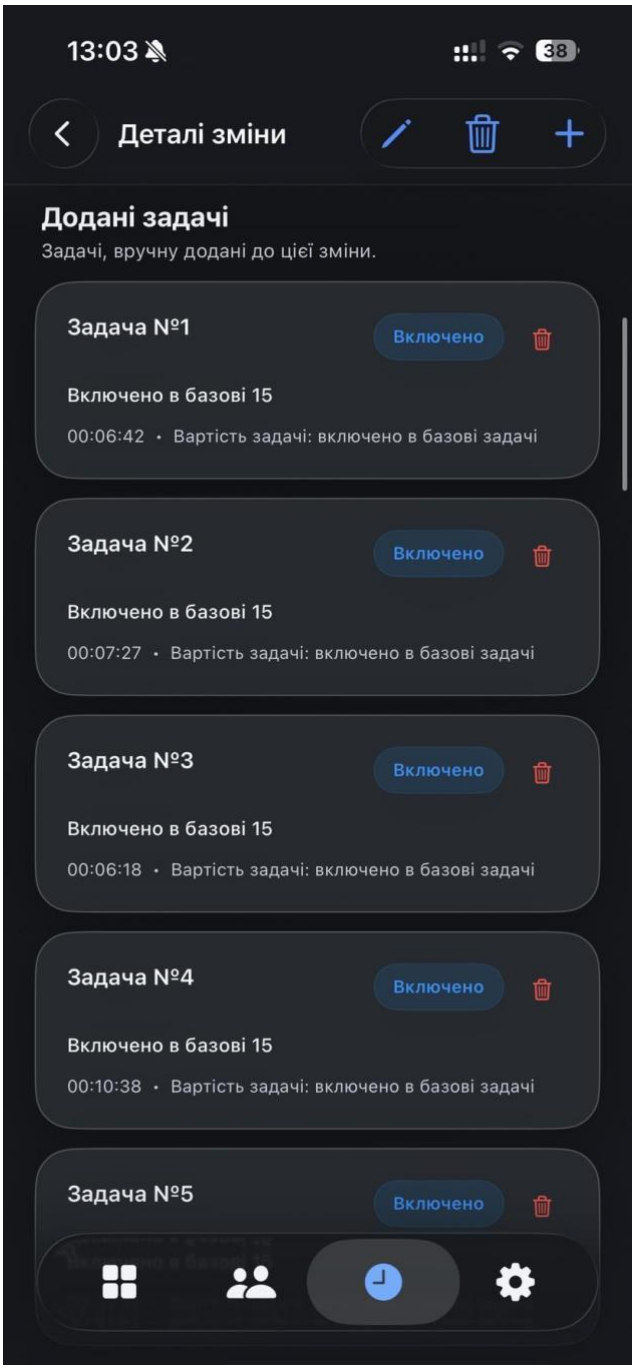


Рисунок 3.9. Список задач у межах зміни

Форма створення зміни містить поля вибору працівника, типу зміни, дати, часу початку, часу завершення, ознак відвідуваності та ставки. Після вибору типу зміни система автоматично підставляє типовий розклад, але користувач може змінити його вручну. Це дозволяє поєднати швидке створення типової зміни з можливістю врахувати фактичні відхилення.

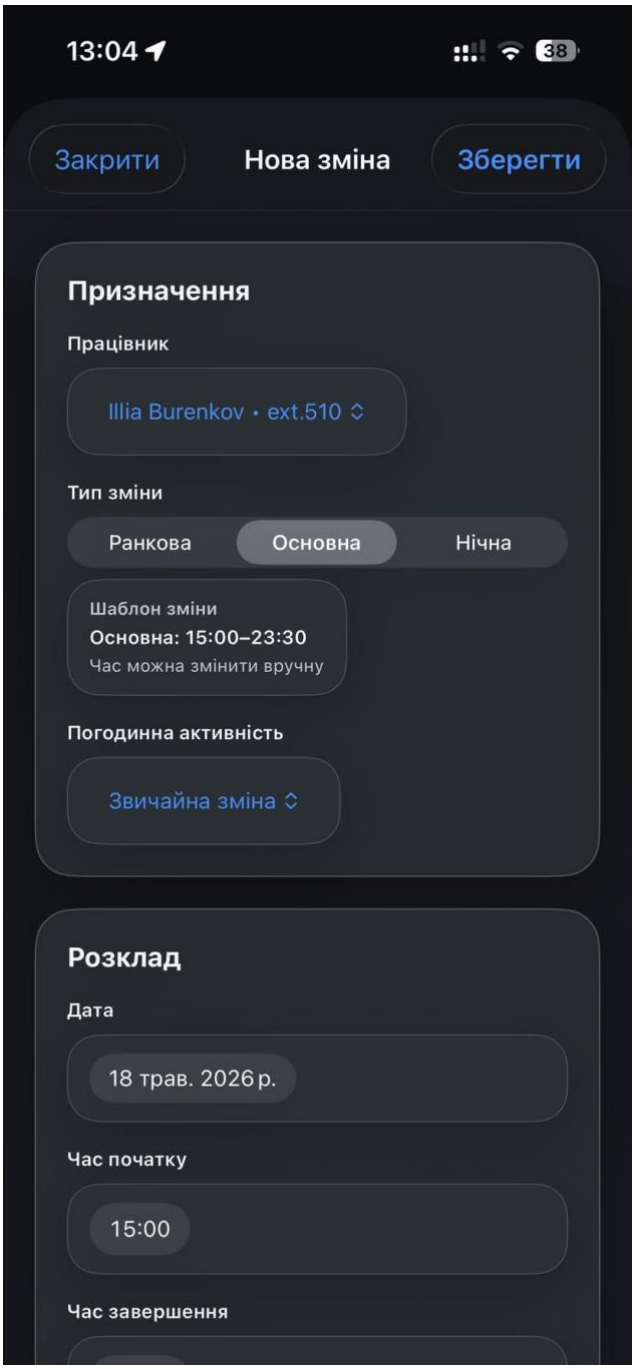


Рисунок 3.10. Форма створення робочої зміни

У нижній частині форми зміни передбачено параметри відвідуваності та оплати. Користувач може позначити запізнення, раннє завершення або наявність прийнятного пояснення. Такі дані впливають на розрахунок погодинної частини оплати та дозволяють врахувати внутрішні правила компанії.

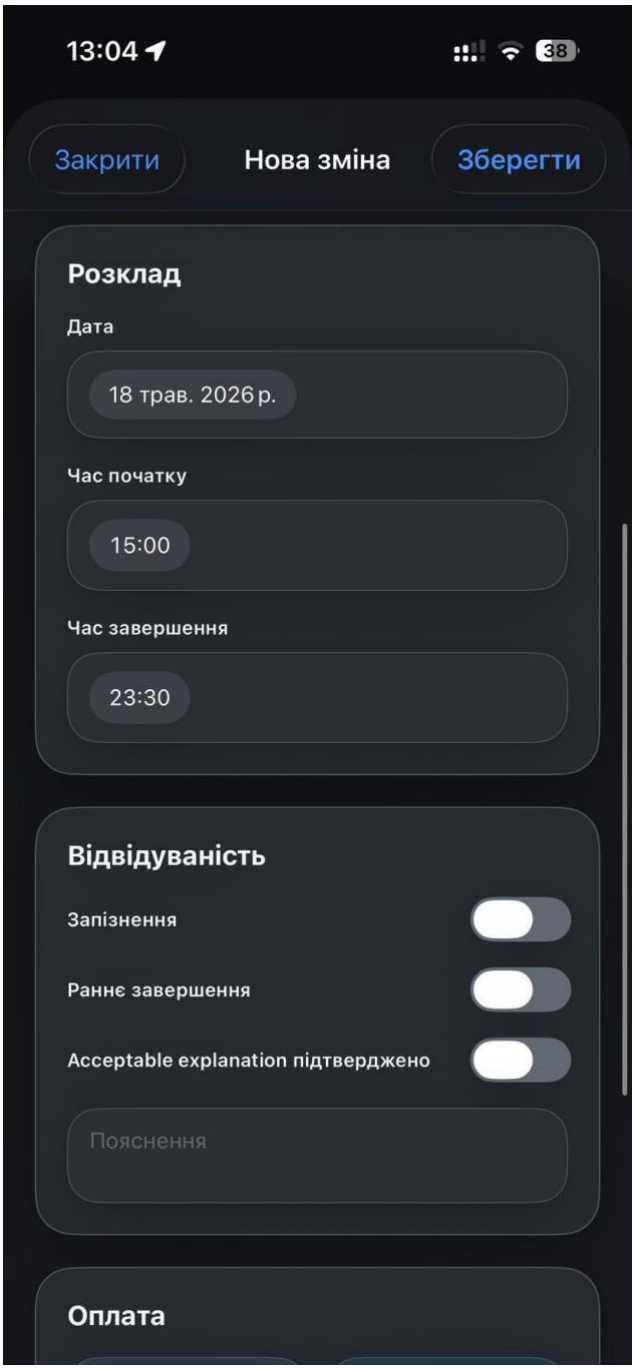


Рисунок 3.11. Додаткові параметри зміни та відвідуваності

Форма додавання задачі дозволяє внести одну або кілька задач за один раз. Номер задачі формується автоматично, а тривалість можна ввести вручну або обрати через поля годин, хвилин і секунд. Після натискання кнопки застосування тривалість перетворюється в секунди, а після збереження задачі додаються до поточної зміни.

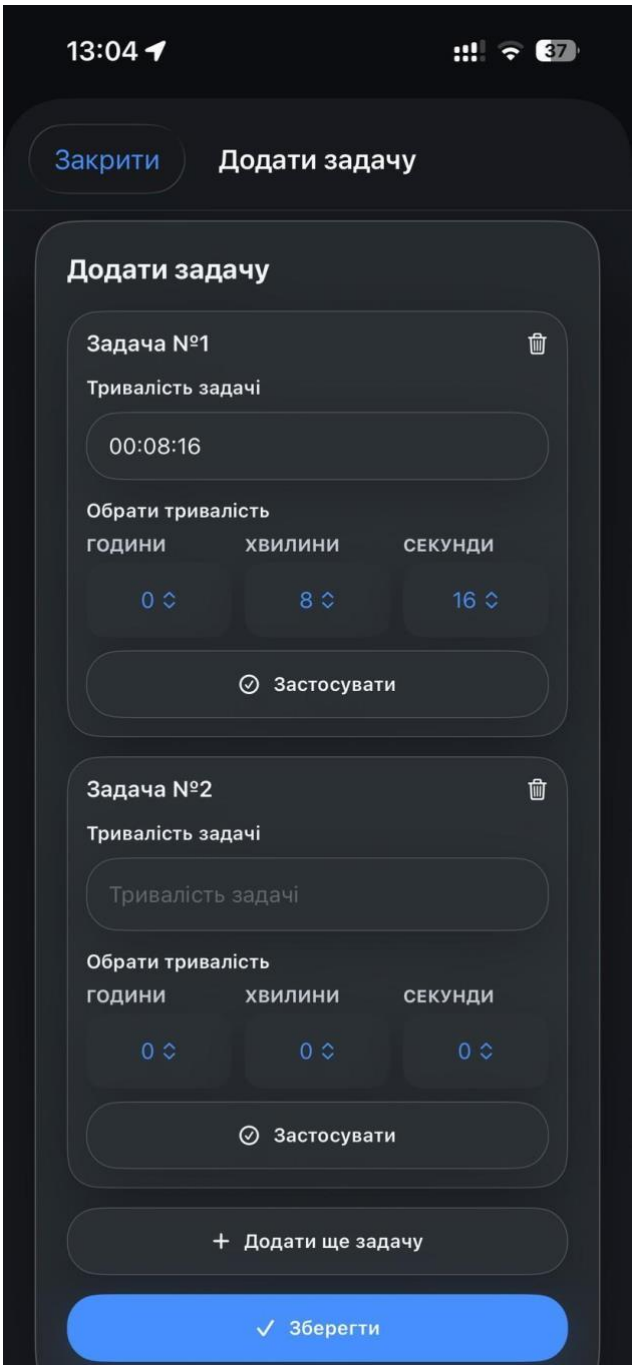


Рисунок 3.12. Форма додавання задач до зміни

У налаштуваннях користувач може змінити мову інтерфейсу. Застосунок підтримує українську та англійську мови, а також може використовувати системну мову пристрою. Наявність локалізації робить застосунок зручнішим для користувачів, які працюють у різних мовних середовищах.

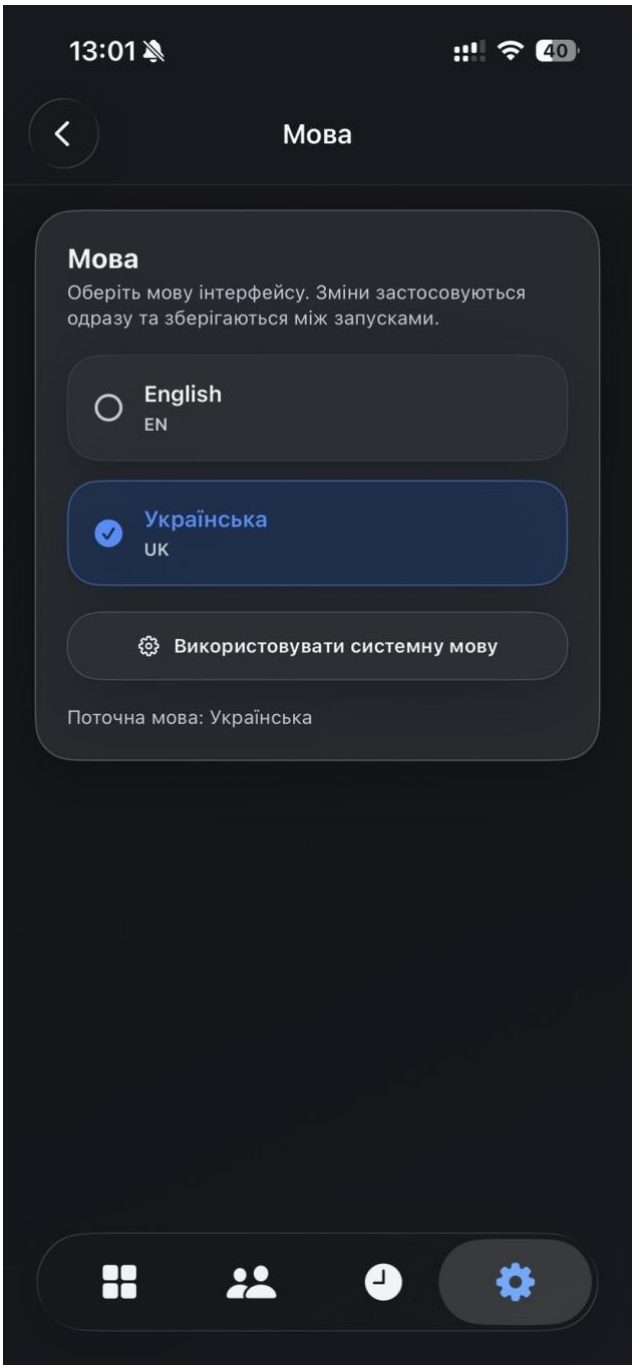


Рисунок 3.13. Екран вибору мови інтерфейсу

Окремий екран налаштувань ставок дає змогу редагувати погодинні ставки для ранкової, основної та нічної зміни. Це важливо, оскільки різні типи змін можуть мати різні правила оплати. Збережені значення використовуються під час створення зміни та під час подальшого розрахунку заробітної плати.

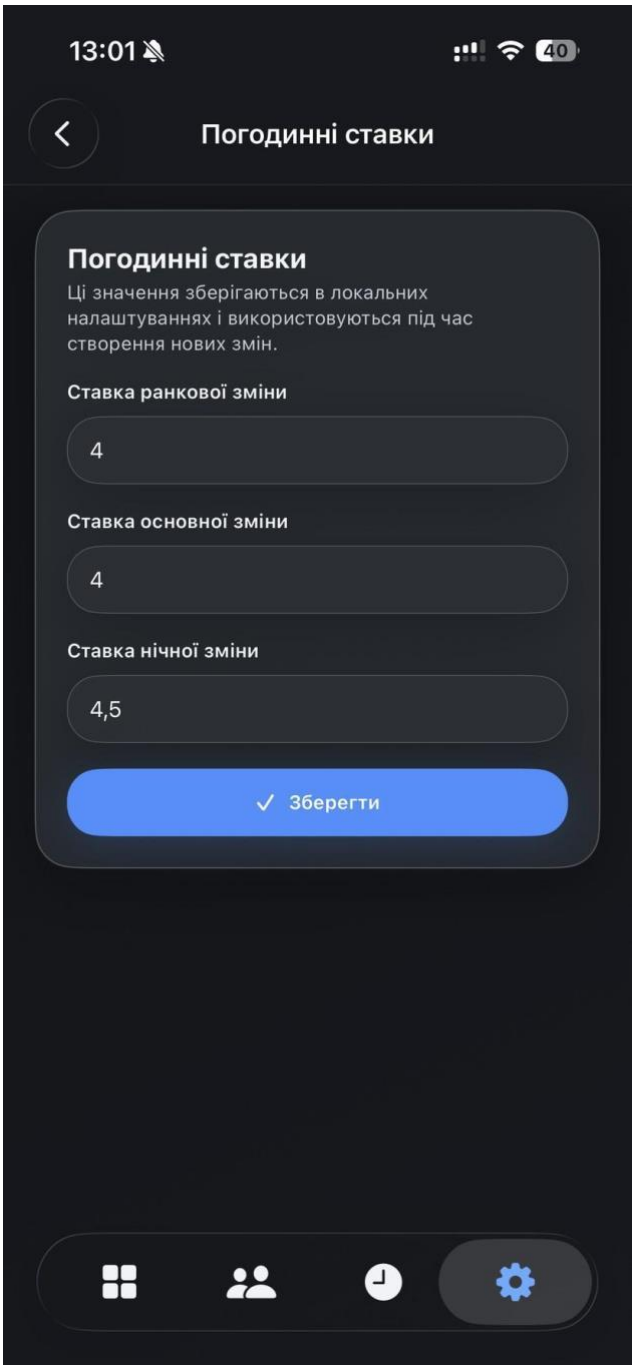


Рисунок 3.14. Екран налаштування погодинних ставок

Екран валюти відповідає за вибір валюти відображення та оновлення курсу. Користувач може вибрати долар США, гривню або євро. Підсумкові суми на головному екрані, у списку змін і в PDF-звіті відображаються в обраній валюті. При цьому базова логіка розрахунку не змінюється, що забезпечує стабільність формул.

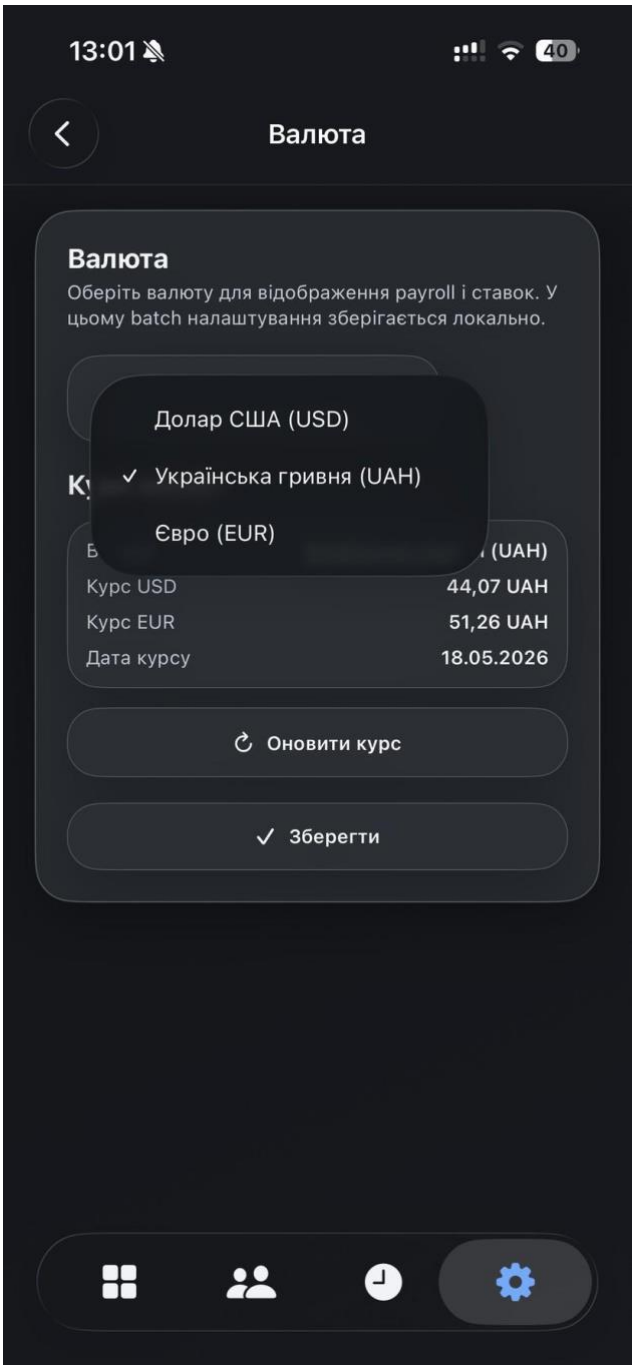


Рисунок 3.15. Екран вибору валюти та оновлення курсу

Екран звітів і експорту призначений для створення PDF-звіту. Користувач вибирає період і натискає кнопку формування документа. Після цього застосунок створює файл і відкриває стандартний механізм експорту. Такий сценарій дозволяє швидко отримати документ із підсумками роботи за потрібний період.

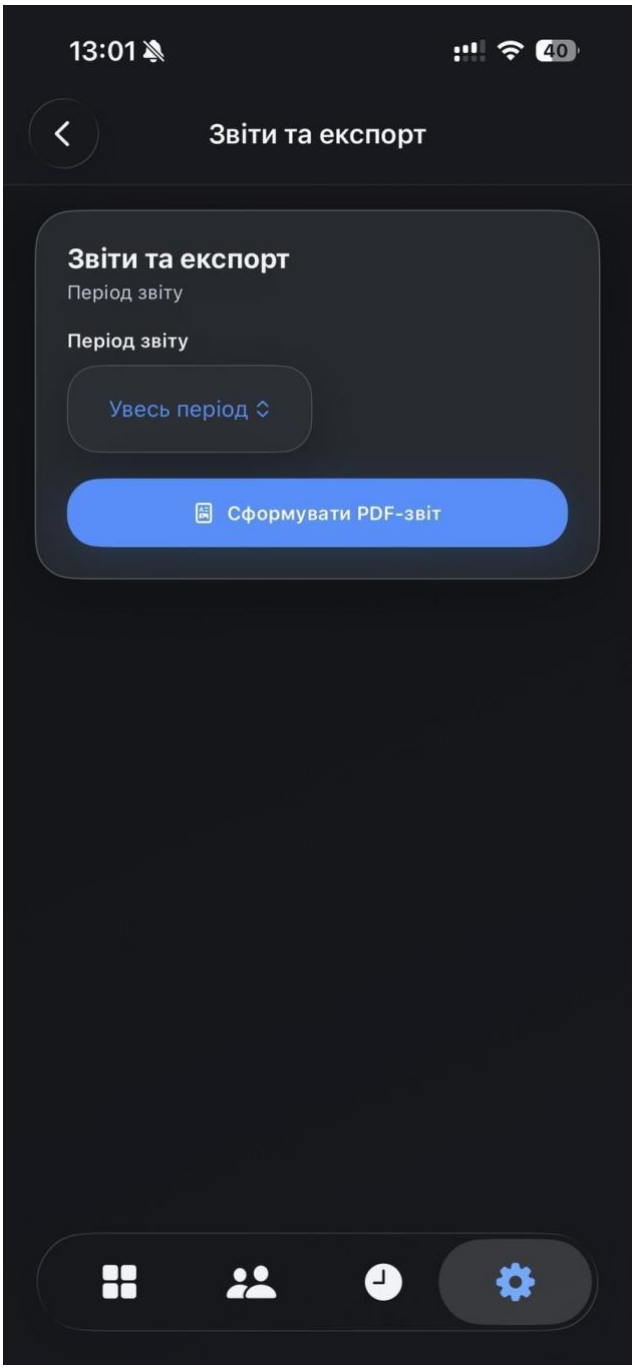


Рисунок 3.16. Екран формування PDF-звіту

Захист доступу налаштовується на окремому екрані безпеки. Користувач може ввімкнути PIN-код, змінити його, активувати Face ID або Touch ID та визначити поведінку автоблокування. Ці можливості важливі, оскільки застосунок містить інформацію про працівників, зміни та суми заробітку.

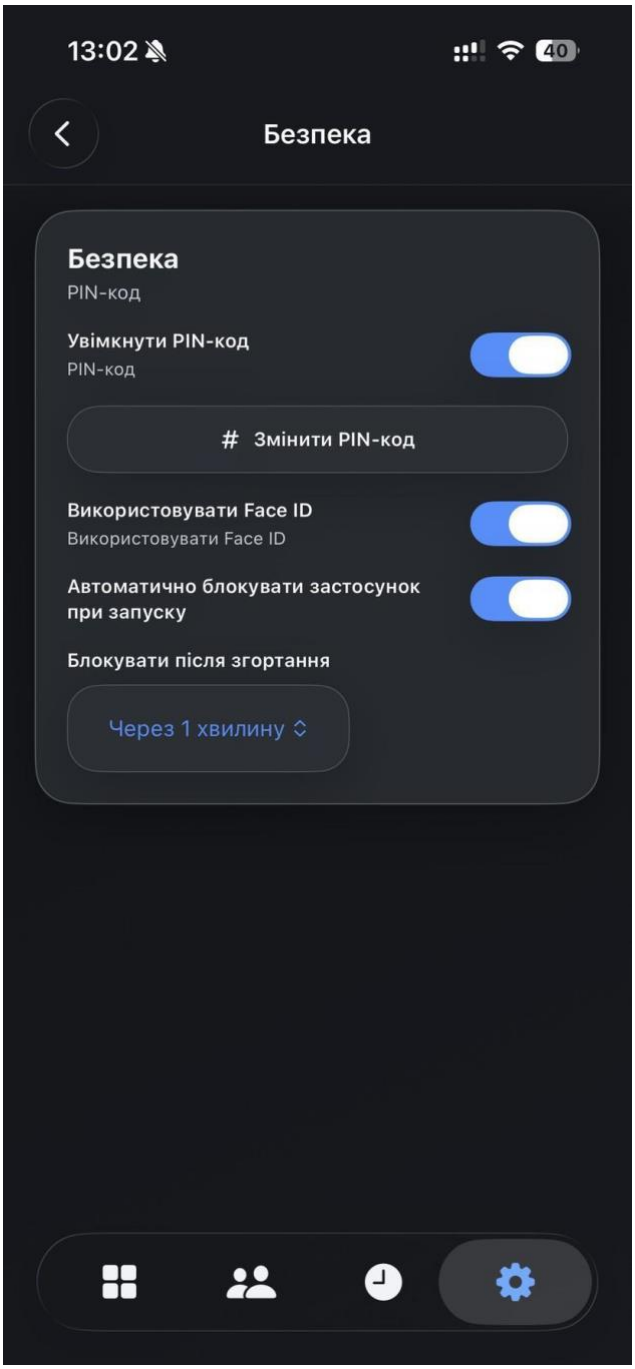
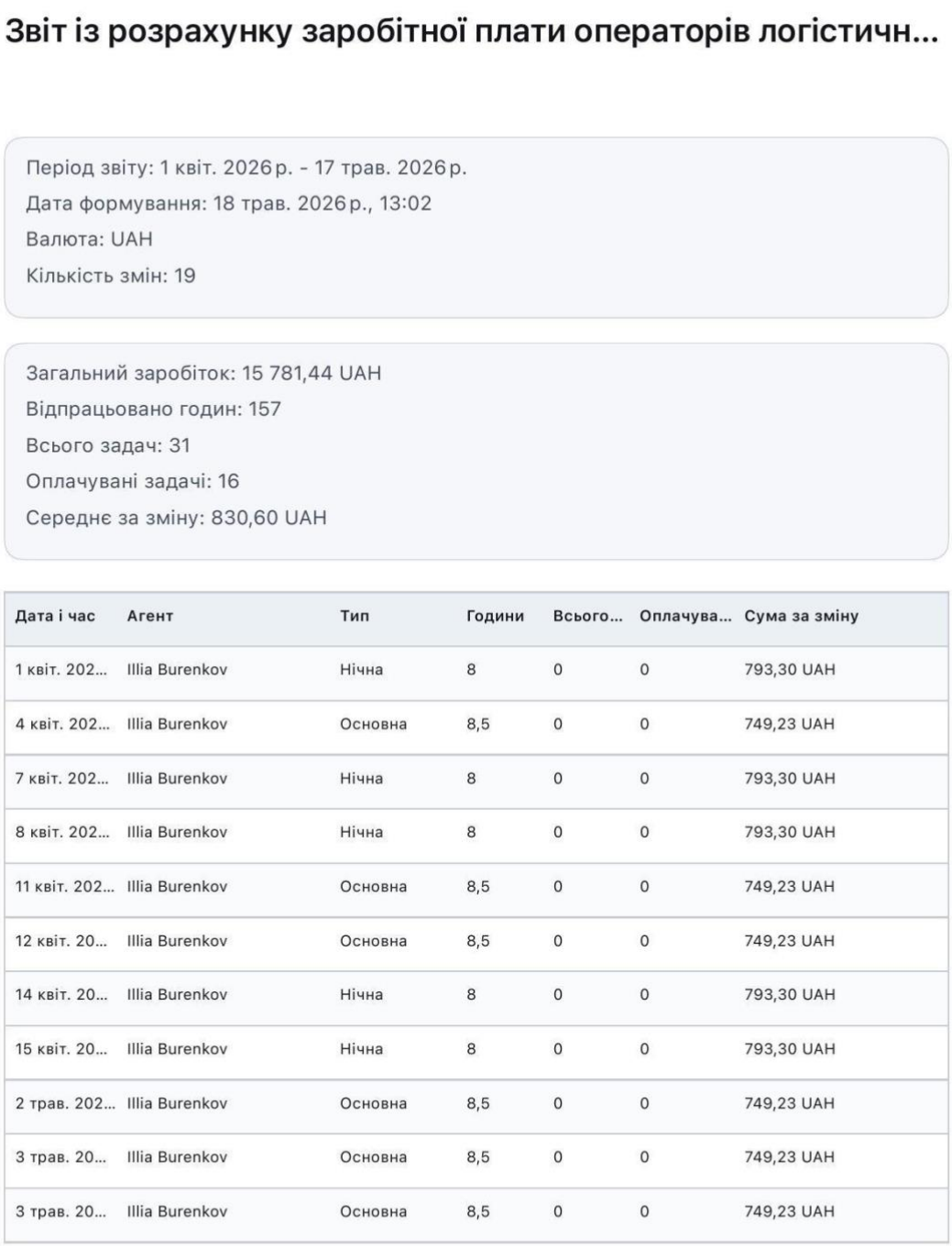


Рисунок 3.17. Екран налаштувань безпеки

Приклад сформованого PDF-звіту показує, що результати розрахунку можуть бути подані у вигляді окремого документа. Звіт містить період, дату формування, валюту, кількість змін, загальний заробіток, відпрацьовані години, кількість задач, оплачувані задачі, середнє значення за зміну та таблицю зі змінами. Документ має світлий стиль оформлення, що робить його придатним для перегляду, друку або надсилання.



Звіт із розрахунку заробітної плати операторів логістичн...

Період звіту: 1 квіт. 2026 р. - 17 трав. 2026 р.
Дата формування: 18 трав. 2026 р., 13:02
Валюта: UAH
Кількість змін: 19

Загальний заробіток: 15 781,44 UAH
Відпрацьовано годин: 157
Всього задач: 31
Оплачувані задачі: 16
Середнє за зміну: 830,60 UAH

Дата і час	Агент	Тип	Години	Всього...	Оплачува...	Сума за зміну
1 квіт. 202...	Illia Burenkov	Нічна	8	0	0	793,30 UAH
4 квіт. 202...	Illia Burenkov	Основна	8,5	0	0	749,23 UAH
7 квіт. 202...	Illia Burenkov	Нічна	8	0	0	793,30 UAH
8 квіт. 202...	Illia Burenkov	Нічна	8	0	0	793,30 UAH
11 квіт. 202...	Illia Burenkov	Основна	8,5	0	0	749,23 UAH
12 квіт. 20...	Illia Burenkov	Основна	8,5	0	0	749,23 UAH
14 квіт. 20...	Illia Burenkov	Нічна	8	0	0	793,30 UAH
15 квіт. 20...	Illia Burenkov	Нічна	8	0	0	793,30 UAH
2 трав. 202...	Illia Burenkov	Основна	8,5	0	0	749,23 UAH
3 трав. 20...	Illia Burenkov	Основна	8,5	0	0	749,23 UAH
3 трав. 20...	Illia Burenkov	Основна	8,5	0	0	749,23 UAH

Рисунок 3.18. Приклад сформованого PDF-звіту

Загалом інтерфейс застосунку побудовано так, щоб основні дії були доступні з нижньої навігації, а детальна інформація відкривалася лише за

потреби. Головний екран дає швидке уявлення про результати роботи, екран працівників дозволяє керувати операторами, екран змін містить історію роботи, налаштування зосереджують службові параметри, а звіти дають змогу отримати підсумковий документ. Такий підхід відповідає поставленій меті та робить програмне забезпечення придатним для практичного використання.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Після завершення основних етапів розроблення було виконано перевірку працездатності програмного забезпечення. Метою тестування було підтвердження правильності роботи основних функцій застосунку, коректності розрахунку заробітної плати, стабільності збереження даних, зручності інтерфейсу та відповідності реалізованих можливостей вимогам, сформованим у попередніх розділах пояснювальної записки.

Тестування виконувалося у середовищі Xcode 26.5 на комп’ютері MacBook Air з процесором M2 та операційною системою macOS Tahoe 26.4.1. Перевірка роботи застосунку проводилася на iPhone 15 Pro з операційною системою iOS 26.4.2. Застосунок встановлювався на пристрій через Xcode у режимі тестового запуску. Такий спосіб є доцільним для етапу розроблення бакалаврської кваліфікаційної роботи, оскільки дозволяє швидко перевіряти зміни в коді, тестувати інтерфейс на реальному пристрої та контролювати поведінку локального сховища даних.

Перевірка охоплювала функції створення працівників, редагування основної інформації, створення робочих змін, додавання і видалення задач, автоматичної нумерації задач, розрахунку заробітної плати, вибору працівника на головній панелі, фільтрації змін за працівником і періодом, вибору валюти, оновлення курсу, формування PDF-звіту, блокування застосунку за допомогою PIN-коду та використання біометричної автентифікації. Окремо перевірялося відображення toast-повідомлень, коректність роботи темної теми та збереження цілісності даних після видалення зміни.

Під час тестування критичних помилок, які б призводили до втрати даних або неправильного розрахунку заробітної плати, не виявлено. У поодиноких випадках спостерігалися короткочасні затримки інтерфейсу під час переходу між екранами або роботи з великим обсягом даних. Такі затримки не впливали на

правильність збереження інформації та результат розрахунків, однак можуть бути враховані як напрям подальшої оптимізації продуктивності. Узагальнені результати тестування наведено в таблиці 4.1.

Таблиця 4.1. Результати тестування основних функцій застосунку

№	Перевірка	Очікуваний результат	Фактичний результат
1	Створення працівника з іменем, посадою, extension і статусом	Працівник зберігається і відображається у списку	Виконано
2	Редагування імені, посади або extension працівника	Оновлені дані зберігаються без створення дубліката	Виконано
3	Створення зміни з вибором працівника, дати, типу і часу	Зміна створюється та прив’язується до вибраного працівника	Виконано
4	Створення нічної зміни з часом 23:30-07:30	Тривалість визначається коректно з переходом через північ	Виконано
5	Додавання задачі з тривалістю у форматі часу	Задача додається до зміни, номер формується автоматично	Виконано
6	Додавання кількох задач у межах однієї зміни	Номери задач формуються послідовно без повторів	Виконано
7	Видалення задачі зі списку задач зміни	Задача видаляється, нумерація інших задач оновлюється	Виконано
8	Розрахунок заробітної плати за зміною, ставкою і задачами	Система визначає погодинну оплату, оплачувані задачі та підсумок	Виконано
9	Перевірка правила перших 15 задач	Окремо оплачуються тільки задачі понад базовий ліміт	Виконано
10	Вибір конкретного працівника на Dashboard	Показується статистика лише вибраного працівника	Виконано
11	Вибір режиму загальної статистики	Показуються дані по всіх працівниках	Виконано
12	Вибір періоду 7 днів, місяць, 3 місяці або рік	Показуються дані тільки за вибраний період	Виконано
13	Повторне натискання активного періоду	Фільтр знімається, відображаються дані за весь період	Виконано
14	Видалення робочої зміни	Зміна та пов’язані задачі видаляються, працівник залишається	Виконано
15	Вибір валюти UAH, USD або EUR	Підсумкові суми відображаються у вибраній валюті	Виконано
16	Оновлення курсу валют через сервіс НБУ	Курс оновлюється або використовується збережене значення	Виконано
17	Формування PDF-звіту за вибраний період	Створюється PDF-документ зі світлим оформленням і таблицею змін	Виконано

18	Експорт PDF-звіту через системне меню iOS	Користувач може зберегти або надіслати документ	Виконано
19	Розблокування застосунку правильним PIN-кодом	Застосунок відкриває доступ до даних	Виконано
20	Біометрична автентифікація за допомогою Face ID	Застосунок розблоковується після успішної перевірки	Виконано
21	Поява toast-повідомлення після дії користувача	Повідомлення з’являється поверх інтерфейсу і зникає автоматично	Виконано

Результати тестування показали, що основні сценарії роботи виконуються відповідно до очікуваної поведінки. Створені працівники, зміни та задачі зберігаються у локальній інформаційній базі, а розрахунок заробітної плати виконується з урахуванням типу зміни, тривалості роботи та кількості оплачуваних задач. Окремо було підтверджено, що фільтр за працівником і фільтр періоду працюють незалежно один від одного. Це дозволяє переглядати як загальну статистику, так і дані конкретного оператора за вибраний або повний період.

Під час перевірки PDF-звіту підтверджено, що документ формується у світлому стилі, має білий фон, темний текст, інформаційні блоки та таблицю змін. Такий формат є придатним для збереження, надсилання або подальшого перегляду поза межами застосунку. Перевірка системного меню поширення показала, що користувач може передати сформований документ через стандартні засоби iOS.

4.2 Вимоги до апаратного та програмного забезпечення

Для роботи розробленого програмного забезпечення потрібен мобільний пристрій Apple з операційною системою iOS. Оскільки застосунок створено для платформи iOS і використовує локальне збереження даних, біометричну автентифікацію, системне меню поширення та засоби формування PDF-документів, його експлуатація передбачає наявність сучасної версії операційної системи та достатнього обсягу пам’яті для збереження локальних даних.

Під час розроблення і тестування використовувався iPhone 15 Pro з iOS 26.4.2. Саме на цьому пристрої перевірялися основні сценарії роботи, зокрема

створення працівників, робочих змін, задач, перегляд статистики, формування PDF-звіту та розблокування застосунку за допомогою Face ID. Розроблення виконувалося на MacBook Air M2 з macOS Tahoe 26.4.1 у середовищі Xcode 26.5.

Для користування застосунком не потрібен окремий сервер або постійне підключення до мережі Інтернет. Основні дані зберігаються локально на пристрої. Інтернет-з’єднання потрібне лише для оновлення курсу валют через зовнішній сервіс. Якщо оновлення курсу недоступне, застосунок може використовувати раніше збережені значення, що дозволяє продовжувати роботу з уже наявними даними.

Зазначені вимоги відповідають умовам, у яких виконувалося розроблення та тестування застосунку. Для подальшого поширення програмного забезпечення може бути використано стандартні механізми підготовки iOS-застосунків, зокрема архівування проєкту в Xcode і передавання збірки через засоби тестового або виробничого розповсюдження. У межах цієї бакалаврської кваліфікаційної роботи достатнім є тестове встановлення на пристрій через Xcode.

4.3 Склад інсталяційного пакету

У межах виконання бакалаврської кваліфікаційної роботи застосунок встановлювався на iPhone через Xcode у режимі тестового запуску. Такий спосіб встановлення дозволяє запускати програмне забезпечення на реальному пристрої, перевіряти роботу інтерфейсу, локального сховища, системних функцій безпеки та формування PDF-звіту. Оскільки застосунок призначений для iOS, підготовка до встановлення виконується засобами Xcode, який формує збірку на основі вихідного коду, ресурсів і налаштувань проєкту.

До складу програмного проєкту входять файли вихідного коду, моделі даних, сервіси бізнес-логіки, компоненти інтерфейсу, ресурси локалізації, налаштування безпеки та допоміжні модулі. Окремими частинами є модуль розрахунку заробітної плати, модуль визначення вартості задач, модуль статистики, модуль формування PDF-звіту, модуль роботи з курсами валют,

модуль блокування застосунку та компоненти для відображення повідомлень користувачу.

Локальна інформаційна база створюється на пристрої автоматично під час роботи застосунку. Користувачу не потрібно вручну створювати окрему базу даних або встановлювати додаткове програмне забезпечення. Після запуску застосунок самостійно працює з моделями працівників, змін, задач, налаштувань і правил тарифікації. Такий підхід спрощує експлуатацію системи та робить її придатною для використання без серверної інфраструктури.

Процес тестового встановлення передбачає відкриття проєкту в Xcode, вибір підключеного iPhone як цільового пристрою, виконання збирання проєкту та запуск застосунку. Після першого запуску користувач може створити власних працівників, внести робочі зміни, додати задачі, переглянути статистику та сформувати звіт. Усі службові дані зберігаються локально на пристрої.

Для фінального розповсюдження застосунку в майбутньому може бути підготовлено архівну збірку, підписану відповідним сертифікатом розробника Apple. Однак у межах цієї роботи головним завданням є демонстрація працездатного прототипу на реальному пристрої, тому тестове встановлення через Xcode є достатнім і відповідає етапу бакалаврського проєктування.

4.4 Аналіз отриманих результатів

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено мобільне програмне забезпечення для обліку робочого часу та розрахунку заробітної плати операторів логістичної компанії. Застосунок забезпечує ведення працівників, створення робочих змін, внесення задач, автоматичний розрахунок заробітку, фільтрацію даних за періодом і працівником, відображення статистики, вибір валюти, формування PDF-звіту та захист доступу до службової інформації.

Реалізована система дозволяє зменшити кількість ручних дій під час обліку змін і задач. Користувачеві не потрібно самостійно виконувати повторювані підрахунки або переносити дані між окремими таблицями. Після створення

зміни та внесення задач застосунок автоматично визначає тривалість роботи, кількість задач, кількість оплачуваних задач і підсумкову суму заробітку. Окреме правило для перших п’ятнадцяти задач дозволяє врахувати специфіку внутрішньої моделі оплати.

Додавання фільтра за працівником підвищило практичну цінність головної панелі та списку змін. Користувач може переглядати як загальну статистику по всіх працівниках, так і окремі показники конкретного оператора. У поєднанні з фільтром періоду це дає змогу швидко аналізувати результати за сім днів, місяць, три місяці, рік або весь період. Така логіка робить застосунок корисним не лише для особистого контролю заробітку, а й для перевірки роботи окремих працівників.

Формування PDF-звіту розширює можливості системи за межі перегляду даних на екрані. Звіт містить період, дату формування, валюту, кількість змін, загальний заробіток, відпрацьовані години, кількість задач, оплачувані задачі, середній заробіток за зміну та таблицю змін. Світле оформлення документа з темним текстом забезпечує його читабельність незалежно від теми пристрою. Наявність системного меню поширення дозволяє зберегти або передати сформований документ стандартними засобами iOS.

Окремою перевагою є реалізація механізму захисту доступу. Інформація про працівників, зміни та заробітну плату має службовий характер, тому використання PIN-коду і Face ID підвищує рівень конфіденційності. Застосунок може блокувати доступ після запуску або повернення до активного стану, що зменшує ризик перегляду даних сторонніми особами.

Під час тестування було підтверджено працездатність основних функцій. Застосунок коректно створює та зберігає працівників, зміни і задачі, виконує розрахунок заробітної плати, оновлює статистику після зміни даних, формує PDF-звіт і забезпечує захищений доступ. Виявлені поодинокі короткочасні фрізи інтерфейсу не впливають на коректність роботи системи, але вказують на можливість подальшої оптимізації. Найімовірнішими напрямами оптимізації є

зменшення кількості повторних розрахунків у списках, кешування окремих агрегованих показників, спрощення важких візуальних ефектів у довгих списках і перенесення частини обчислень із тіла SwiftUI-екранів у сервіси.

Практичне значення отриманого результату полягає в тому, що застосунок може використовуватися як зручний інструмент для щоденного обліку змін і контролю заробітної плати. Він поєднує просту модель введення даних, автоматичні розрахунки, наочну статистику, звітність і базовий захист інформації. Розроблена система відповідає поставленій меті та може бути розширена в майбутньому за рахунок хмарної синхронізації, розширеної аналітики, ролей користувачів або інтеграції з корпоративними сервісами логістичної компанії.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено мобільне програмне забезпечення для обліку робочого часу та розрахунку заробітної плати операторів логістичної компанії. Розроблений застосунок забезпечує ведення працівників, створення робочих змін, внесення задач, автоматичне визначення суми заробітку, перегляд статистики, формування PDF-звіту та захист доступу до службової інформації.

У першому розділі було проаналізовано предметну область, визначено основні проблеми ручного обліку робочих змін і задач, розглянуто існуючі рішення та сформовано вимоги до програмного забезпечення. Було встановлено, що універсальні системи обліку часу не повністю враховують специфіку розрахунку заробітної плати операторів логістичної компанії, тому доцільним є створення спеціалізованого мобільного застосунку.

У другому розділі було виконано проєктування інформаційного та програмного забезпечення. Було визначено основні сутності предметної області, побудовано ER-діаграму, діаграму класів, діаграму пакетів і діаграму компонентів. Запропонована структура дозволяє відокремити інтерфейс користувача, моделі даних, сервіси розрахунку, модулі безпеки, налаштування та формування звітності.

У третьому розділі було описано реалізацію програмного забезпечення. Для розроблення використано Swift, SwiftUI, SwiftData, Xcode, Keychain, LocalAuthentication, мережевий сервіс отримання курсів валют та засоби формування PDF-документів. Основні дані зберігаються локально на пристрої, що дає змогу працювати із застосунком без обов’язкового підключення до сервера.

У четвертому розділі було наведено результати тестування, вимоги до апаратного та програмного забезпечення, склад проєкту для тестового встановлення та аналіз отриманих результатів. Під час перевірки підтверджено працездатність основних функцій: створення працівників, робочих змін і задач,

автоматичної нумерації задач, розрахунку заробітної плати, фільтрації за працівником і періодом, вибору валюти, формування PDF-звіту та захисту доступу.

Практичне значення роботи полягає в тому, що застосунок може використовуватися для спрощення щоденного обліку роботи операторів і контролю їхнього заробітку. Автоматизація розрахунків зменшує ризик помилок, пришвидшує підрахунок суми оплати та робить процес нарахування більш прозорим. У майбутньому систему можна розширити хмарною синхронізацією, розширеною аналітикою, підтримкою ролей користувачів і додатковими видами звітності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Положення про бакалаврську кваліфікаційну роботу у Національному університеті біоресурсів і природокористування України. Київ: НУБіП України, 2021.

2. Методичні рекомендації до виконання бакалаврської кваліфікаційної роботи для здобувачів спеціальності 121 «Інженерія програмного забезпечення». Київ: НУБіП України, 2025.

3. Apple Developer Documentation. Swift.
URL: https://developer.apple.com/swift/

4. Apple Developer Documentation. SwiftUI.
URL: https://developer.apple.com/documentation/swiftui

5. Apple Developer Documentation. SwiftData.
URL: https://developer.apple.com/documentation/swiftdata

6. Apple Developer Documentation. Keychain Services.
URL: https://developer.apple.com/documentation/security/keychain_services

7. Apple Developer Documentation. UIKit. UIActivityViewController.
URL: https://developer.apple.com/documentation/uikit/uiactivityviewcontroller

8. Національний банк України. API для розробників.
URL: https://bank.gov.ua/ua/markets/exchangerates

9. Національний банк України. Офіційний курс гривні щодо іноземних валют. URL: https://bank.gov.ua/ua/markets/exchangerates

10. Object Management Group. Unified Modeling Language Specification. URL: https://www.omg.org/spec/UML/

11. Clockify. Time Tracking Software. URL: https://clockify.me/

12. Toggl Track. Time Tracking Software. URL: https://toggl.com/track/

13. QuickBooks Time. Time Tracking and Timesheets.
URL: https://quickbooks.intuit.com/time-tracking/

14. ISO/IEC/IEEE 29148:2018. Systems and software engineering. Life cycle processes. Requirements engineering. Geneva: ISO, 2018.

ДОДАТКИ

Додаток А

ФРАГМЕНТИ КОДУ МОДЕЛЕЙ ДАНИХ ТА ЛОКАЛЬНОГО СХОВИЩА

Лістинг В.1. Моделі SwiftData та контейнер локальної бази даних

```swift
// Utilities/AppModelContainer.swift

import Foundation
import SwiftData

enum AppModelContainer {
    static func makeContainer() -> ModelContainer {
        makeContainer(isStoredInMemoryOnly: false)
    }

    static func makePreviewContainer() -> ModelContainer {
        makeContainer(isStoredInMemoryOnly: true)
    }

    private static func makeContainer(isStoredInMemoryOnly: Bool) -> ModelContainer {
        let schema = Schema([
            Employee.self,
            WorkShift.self,
            TaskRecord.self,
            AppSettings.self,
            TaskPricingBracket.self
        ])

        let configuration = ModelConfiguration(schema: schema, isStoredInMemoryOnly: isStoredInMemoryOnly)

        do {
            return try ModelContainer(for: schema, configurations: [configuration])
        } catch {
            fatalError("Failed to create model container: \(error.localizedDescription)")
        }
    }
}

// Models/Employee.swift

import Foundation
import SwiftData

@Model
final class Employee {
    var id: UUID
    var fullName: String
    var position: String
    var personnelNumber: String
    var isActive: Bool
    var createdAt: Date

    @Relationship(inverse: \WorkShift.employee)
    var shifts = [WorkShift]()

    init(
        id: UUID = UUID(),
        fullName: String,
        position: String,
        personnelNumber: String,
        isActive: Bool = true,
        createdAt: Date = .now,
        shifts: [WorkShift] = []
    ) {
        self.id = id
        self.fullName = fullName
        self.position = position
        self.personnelNumber = personnelNumber
        self.isActive = isActive
        self.createdAt = createdAt
        self.shifts = shifts
    }

    var hasHistoricalShifts: Bool {
        !shifts.isEmpty
    }
```

```
    var shiftCount: Int {
        shifts.count
    }
}

// Models/WorkShift.swift

import Foundation
import SwiftData

@Model
final class WorkShift {
    var id: UUID
    var date: Date
    var startTime: Date
    var endTime: Date
    var shiftType: ShiftType
    var hourlyActivityTypeCode: String?
    var hourlyRate: Decimal
    var notes: String
    var isLateStorage: Bool?
    var isEarlyFinishedStorage: Bool?
    var hasAcceptableExplanationStorage: Bool?
    var explanationText: String?
    var createdAt: Date

    var employee: Employee?

    @Relationship(deleteRule: .cascade, inverse: \TaskRecord.shift)
    var tasks = [TaskRecord]()

    init(
        id: UUID = UUID(),
        date: Date,
        startTime: Date,
        endTime: Date,
        shiftType: ShiftType,
        hourlyActivityType: HourlyActivityType = .regular,
        hourlyRate: Decimal,
        notes: String = "",
        isLate: Bool = false,
        isEarlyFinished: Bool = false,
        hasAcceptableExplanation: Bool = false,
        explanationText: String? = nil,
        createdAt: Date = .now,
        employee: Employee? = nil,
        tasks: [TaskRecord] = []
    ) {
        self.id = id
        self.date = date
        self.startTime = startTime
        self.endTime = Self.normalizedEndTime(startTime: startTime, endTime: endTime)
        self.shiftType = shiftType
        self.hourlyActivityTypeCode = hourlyActivityType.rawValue
        self.hourlyRate = hourlyRate
        self.notes = notes
        self.isLateStorage = isLate
        self.isEarlyFinishedStorage = isEarlyFinished
        self.hasAcceptableExplanationStorage = hasAcceptableExplanation
        self.explanationText = explanationText
        self.createdAt = createdAt
        self.employee = employee
        self.tasks = tasks
    }

    var effectiveEndTime: Date {
        Self.normalizedEndTime(startTime: startTime, endTime: endTime)
    }

    var workedDurationSeconds: Int {
        max(0, Int(effectiveEndTime.timeIntervalSince(startTime)))
    }

    var crossesMidnight: Bool {
        !Calendar.current.isDate(startTime, inSameDayAs: effectiveEndTime)
    }

    var hasAttendanceIssue: Bool {
        isLate || isEarlyFinished
    }

    var isLate: Bool {
```

```
            get { isLateStorage ?? false }
            set { isLateStorage = newValue }
        }

        var isEarlyFinished: Bool {
            get { isEarlyFinishedStorage ?? false }
            set { isEarlyFinishedStorage = newValue }
        }

        var hasAcceptableExplanation: Bool {
            get { hasAcceptableExplanationStorage ?? false }
            set { hasAcceptableExplanationStorage = newValue }
        }

        var hasApprovedAttendanceExplanation: Bool {
            hasAcceptableExplanation &&
                !(explanationText ?? "").trimmingCharacters(in: .whitespacesAndNewlines).isEmpty
        }

        var hourlyActivityType: HourlyActivityType {
            get { HourlyActivityType(storedValue: hourlyActivityTypeCode) }
            set { hourlyActivityTypeCode = newValue.rawValue }
        }

        var hourlyPayCoefficient: Decimal {
            hourlyActivityType.hourlyPayCoefficient
        }

        func applySchedule(
            date: Date,
            startTime: Date,
            endTime: Date,
            shiftType: ShiftType,
            hourlyActivityType: HourlyActivityType,
            hourlyRate: Decimal,
            notes: String,
            isLate: Bool,
            isEarlyFinished: Bool,
            hasAcceptableExplanation: Bool,
            explanationText: String?,
            employee: Employee?
        ) {
            self.date = date
            self.startTime = startTime
            self.endTime = Self.normalizedEndTime(startTime: startTime, endTime: endTime)
            self.shiftType = shiftType
            self.hourlyActivityType = hourlyActivityType
            self.hourlyRate = hourlyRate
            self.notes = notes
            self.isLate = isLate
            self.isEarlyFinished = isEarlyFinished
            self.hasAcceptableExplanation = hasAcceptableExplanation
            self.explanationText = explanationText
            self.employee = employee
        }

        private static func normalizedEndTime(startTime: Date, endTime: Date) -> Date {
            guard endTime < startTime else {
                return endTime
            }

            return Calendar.current.date(byAdding: .day, value: 1, to: endTime) ?? endTime
        }
    }

    // Models/TaskRecord.swift

    import Foundation
    import SwiftData

    @Model
    final class TaskRecord {
        static let manualEntryRawBlock = "Manual task entry"

        var id: UUID
        var externalTaskID: String
        var createdAt: Date
        var taskDate: Date?
        var taskType: String
        var status: String
        var durationSeconds: Int
        var salesTimeSeconds: Int?
        var agentName: String?
```

```
    var rawBlock: String?

    var shift: WorkShift?

    init(
        id: UUID = UUID(),
        externalTaskID: String,
        createdAt: Date = .now,
        taskDate: Date? = nil,
        taskType: String,
        status: String,
        durationSeconds: Int,
        salesTimeSeconds: Int? = nil,
        agentName: String? = nil,
        rawBlock: String? = nil,
        shift: WorkShift? = nil
    ) {
        self.id = id
        self.externalTaskID = Self.normalizedExternalTaskID(externalTaskID)
        self.createdAt = createdAt
        self.taskDate = taskDate
        self.taskType = Self.normalizedTaskType(taskType)
        self.status = Self.normalizedStatus(status)
        self.durationSeconds = durationSeconds
        self.salesTimeSeconds = salesTimeSeconds
        self.agentName = agentName
        self.rawBlock = rawBlock
        self.shift = shift
    }

    var isEligible: Bool {
        Self.isEligible(taskType: taskType, status: status, salesTimeSeconds: salesTimeSeconds)
    }

    var isManualEntry: Bool {
        rawBlock == Self.manualEntryRawBlock
    }

    static func isEligible(taskType: String, status: String, salesTimeSeconds: Int? = nil) -> Bool {
        guard let knownTaskType = KnownTaskType(storedValue: normalizedTaskType(taskType)) else {
            return false
        }

        guard knownTaskType.isPayrollEligibleType else {
            return false
        }

        switch KnownTaskStatus(importedValue: normalizedStatus(status)) {
        case .completed:
            return true
        case .cancelled:
            guard knownTaskType == .logEditing, let salesTimeSeconds else {
                return false
            }

            return salesTimeSeconds > 1
        default:
            return false
        }
    }

    static func normalizedExternalTaskID(_ value: String) -> String {
        value.trimmingCharacters(in: .whitespacesAndNewlines)
    }

    static func normalizedTaskType(_ value: String) -> String {
        KnownTaskType(importedValue: value)?.rawValue ?? value.trimmingCharacters(in:
.whitespacesAndNewlines)
    }

    static func normalizedStatus(_ value: String) -> String {
        KnownTaskStatus(importedValue: value)?.rawValue ?? value.trimmingCharacters(in:
.whitespacesAndNewlines)
    }
}

enum ManualTaskNumbering {
    static func manualTasks(for shift: WorkShift) -> [TaskRecord] {
        shift.tasks
            .filter(\.isManualEntry)
            .sorted(by: manualTaskSort)
    }
```

```
    static func renumberManualTasks(for shift: WorkShift) {
        for (index, task) in manualTasks(for: shift).enumerated() {
            task.externalTaskID = "\(index + 1)"
        }
    }

    static func renumberManualTasks(for shift: WorkShift, in modelContext: ModelContext) throws {
        renumberManualTasks(for: shift)

        if modelContext.hasChanges {
            try modelContext.save()
        }
    }

    static func nextManualTaskNumber(for shift: WorkShift) -> Int {
        manualTasks(for: shift).count + 1
    }

    nonisolated private static func manualTaskSort(lhs: TaskRecord, rhs: TaskRecord) -> Bool {
        let leftNumber = Int(lhs.externalTaskID)
        let rightNumber = Int(rhs.externalTaskID)

        switch (leftNumber, rightNumber) {
        case let (left?, right?) where left != right:
            return left < right
        case (_?, nil):
            return true
        case (nil, _?):
            return false
        default:
            if lhs.createdAt != rhs.createdAt {
                return lhs.createdAt < rhs.createdAt
            }

            return lhs.externalTaskID < rhs.externalTaskID
        }
    }
}

// Models/AppSettings.swift

import Foundation
import SwiftData

@Model
final class AppSettings {
    var id: UUID
    var morningHourlyRate: Decimal
    var mainHourlyRate: Decimal
    var nightHourlyRate: Decimal
    var selectedCurrencyCode: String
    var includedEligibleTasksPerShift: Int
    var createdAt: Date
    var updatedAt: Date

    init(
        id: UUID = UUID(),
        morningHourlyRate: Decimal,
        mainHourlyRate: Decimal,
        nightHourlyRate: Decimal,
        selectedCurrencyCode: String = AppCurrency.usd.rawValue,
        includedEligibleTasksPerShift: Int = 15,
        createdAt: Date = .now,
        updatedAt: Date = .now
    ) {
        self.id = id
        self.morningHourlyRate = morningHourlyRate
        self.mainHourlyRate = mainHourlyRate
        self.nightHourlyRate = nightHourlyRate
        self.selectedCurrencyCode = selectedCurrencyCode
        self.includedEligibleTasksPerShift = includedEligibleTasksPerShift
        self.createdAt = createdAt
        self.updatedAt = updatedAt
    }

    var selectedCurrency: AppCurrency {
        get { AppCurrency(rawValue: selectedCurrencyCode) ?? .usd }
        set { selectedCurrencyCode = newValue.rawValue }
    }
}

// Models/TaskPricingBracket.swift
```

```
import Foundation
import SwiftData

@Model
final class TaskPricingBracket {
    var id: UUID
    var shiftType: ShiftType
    var minDurationSeconds: Int
    var maxDurationSeconds: Int
    var price: Decimal
    var sortOrder: Int

    init(
        id: UUID = UUID(),
        shiftType: ShiftType,
        minDurationSeconds: Int,
        maxDurationSeconds: Int,
        price: Decimal,
        sortOrder: Int
    ) {
        self.id = id
        self.shiftType = shiftType
        self.minDurationSeconds = minDurationSeconds
        self.maxDurationSeconds = maxDurationSeconds
        self.price = price
        self.sortOrder = sortOrder
    }

    var createdComparisonID: String {
        id.uuidString
    }
}
```

Додаток Б

ФРАГМЕНТ КОДУ СТВОРЕННЯ ТА РЕДАГУВАННЯ РОБОЧОЇ ЗМІНИ

Лістинг Д.1. ViewModel форми робочої зміни

```
import Foundation
import Observation
import SwiftData

@Observable
final class ShiftEditorViewModel {
    let shift: WorkShift?

    var selectedEmployeeID: UUID?
    var shiftType: ShiftType
    var hourlyActivityType: HourlyActivityType
    var shiftDate: Date
    var startTime: Date
    var endTime: Date
    var hourlyRate: Decimal
    var isLate: Bool
    var isEarlyFinished: Bool
    var hasAcceptableExplanation: Bool
    var explanationText: String
    var validationMessageKey: LocalizedTextKey?

    init(shift: WorkShift? = nil) {
        let initialShiftType = shift?.shiftType ?? .main
        let templateTimeRange = Self.templateTimeRange(for: initialShiftType)

        self.shift = shift
        self.selectedEmployeeID = shift?.employee?.id
        self.shiftType = initialShiftType
        self.hourlyActivityType = shift?.hourlyActivityType ?? .regular
        self.shiftDate = shift?.date ?? Calendar.current.startOfDay(for: .now)
        self.startTime = shift?.startTime ?? templateTimeRange.start
        self.endTime = shift?.effectiveEndTime ?? templateTimeRange.end
        self.hourlyRate = shift?.hourlyRate ?? .zero
        self.isLate = shift?.isLate ?? false
        self.isEarlyFinished = shift?.isEarlyFinished ?? false
        self.hasAcceptableExplanation = shift?.hasAcceptableExplanation ?? false
        self.explanationText = shift?.explanationText ?? ""
    }

    var titleKey: LocalizedTextKey {
        shift == nil ? L.shiftsCreateTitle : L.shiftsEditTitle
    }

    var workedDurationSeconds: Int {
        let startDate = combinedDate(date: shiftDate, time: startTime)
        let endDate = normalizedEndDate(
            startDate: startDate,
            endDate: combinedDate(date: shiftDate, time: endTime)
        )

        return max(0, Int(endDate.timeIntervalSince(startDate)))
    }

    var endsNextDay: Bool {
        let startDate = combinedDate(date: shiftDate, time: startTime)
        let endDate = normalizedEndDate(
            startDate: startDate,
            endDate: combinedDate(date: shiftDate, time: endTime)
        )

        return !Calendar.current.isDate(startDate, inSameDayAs: endDate)
    }

    var shouldAutofillRateOnAppear: Bool {
        shift == nil
    }

    func applyHourlyRate(using settings: AppSettings?) {
        hourlyRate = hourlyRateForCurrentShiftType(using: settings)
    }

    func applyShiftTemplate(for shiftType: ShiftType) {
        let timeRange = Self.templateTimeRange(for: shiftType)
        startTime = timeRange.start
```

```
    endTime = timeRange.end
}

var shiftTemplateKey: LocalizedTextKey {
    switch shiftType {
    case .morning:
        return L.shiftsTemplateMorning
    case .main:
        return L.shiftsTemplateMain
    case .night:
        return L.shiftsTemplateNight
    }
}

func save(
    in context: ModelContext,
    selectedEmployee: Employee?,
    settings: AppSettings?
) throws {
    validationMessageKey = nil

    let startDate = combinedDate(date: shiftDate, time: startTime)
    let endDate = combinedDate(date: shiftDate, time: endTime)
    let rate = hourlyRate == .zero ? hourlyRateForCurrentShiftType(using: settings) : hourlyRate

    if let shift {
        shift.applySchedule(
            date: Calendar.current.startOfDay(for: shiftDate),
            startTime: startDate,
            endTime: endDate,
            shiftType: shiftType,
            hourlyActivityType: hourlyActivityType,
            hourlyRate: rate,
            notes: shift.notes,
            isLate: isLate,
            isEarlyFinished: isEarlyFinished,
            hasAcceptableExplanation: hasAcceptableExplanation,
            explanationText: normalizedExplanationText,
            employee: selectedEmployee
        )
    } else {
        context.insert(
            WorkShift(
                date: Calendar.current.startOfDay(for: shiftDate),
                startTime: startDate,
                endTime: endDate,
                shiftType: shiftType,
                hourlyActivityType: hourlyActivityType,
                hourlyRate: rate,
                isLate: isLate,
                isEarlyFinished: isEarlyFinished,
                hasAcceptableExplanation: hasAcceptableExplanation,
                explanationText: normalizedExplanationText,
                employee: selectedEmployee
            )
        )
    }

    do {
        try context.save()
    } catch {
        validationMessageKey = L.shiftsValidationSaveFailed
        throw error
    }
}

private var normalizedExplanationText: String? {
    let trimmedText = explanationText.trimmingCharacters(in: .whitespacesAndNewlines)
    return trimmedText.isEmpty ? nil : trimmedText
}

var hourlyPayCoefficient: Decimal {
    hourlyActivityType.hourlyPayCoefficient
}

private func hourlyRateForCurrentShiftType(using settings: AppSettings?) -> Decimal {
    if let settings {
        switch shiftType {
        case .morning:
            return settings.morningHourlyRate
        case .main:
            return settings.mainHourlyRate
        case .night:
```

```
                return settings.nightHourlyRate
            }
        }

        switch shiftType {
        case .morning:
            return Decimal(string: "14.00") ?? 14
        case .main:
            return Decimal(string: "15.00") ?? 15
        case .night:
            return Decimal(string: "17.00") ?? 17
        }
    }

    private func combinedDate(date: Date, time: Date) -> Date {
        let calendar = Calendar.current
        let dateComponents = calendar.dateComponents([.year, .month, .day], from: date)
        let timeComponents = calendar.dateComponents([.hour, .minute, .second], from: time)

        return calendar.date(
            from: DateComponents(
                year: dateComponents.year,
                month: dateComponents.month,
                day: dateComponents.day,
                hour: timeComponents.hour,
                minute: timeComponents.minute,
                second: timeComponents.second
            )
        ) ?? date
    }

    private func normalizedEndDate(startDate: Date, endDate: Date) -> Date {
        guard endDate < startDate else {
            return endDate
        }

        return Calendar.current.date(byAdding: .day, value: 1, to: endDate) ?? endDate
    }

    private static func defaultTime(hour: Int, minute: Int) -> Date {
        Calendar.current.date(bySettingHour: hour, minute: minute, second: 0, of: .now) ?? .now
    }

    private static func templateTimeRange(for shiftType: ShiftType) -> (start: Date, end: Date) {
        switch shiftType {
        case .morning:
            return (
                start: defaultTime(hour: 7, minute: 30),
                end: defaultTime(hour: 15, minute: 0)
            )
        case .main:
            return (
                start: defaultTime(hour: 15, minute: 0),
                end: defaultTime(hour: 23, minute: 30)
            )
        case .night:
            return (
                start: defaultTime(hour: 23, minute: 30),
                end: defaultTime(hour: 7, minute: 30)
            )
        }
    }
}
```

Додаток В

ФРАГМЕНТИ КОДУ РОЗРАХУНКУ ЗАРОБІТНОЇ ПЛАТИ

Лістинг Ж.1. PayrollEngine та PricingEngine

```
// Services/Payroll/PayrollEngine.swift

import Foundation

enum PayrollEngineError: LocalizedError {
    case missingSettings

    var errorDescription: String? {
        switch self {
        case .missingSettings:
            return L.payrollSettingsMissing.rawValue
        }
    }
}

struct PayrollEngine {
    static let includedEligibleTasksPerShift = 15
    static let defaultHourlyPayCoefficient = Decimal(string: "0.5") ?? Decimal(0.5)

    private let pricingEngine: PricingEngine

    init(pricingEngine: PricingEngine = PricingEngine()) {
        self.pricingEngine = pricingEngine
    }

    func calculateBreakdown(
        for shift: WorkShift,
        settings: AppSettings,
        brackets: [TaskPricingBracket]
    ) throws -> PayrollBreakdown {
        let workedSeconds = shift.workedDurationSeconds
        let workedHours = Decimal(workedSeconds) / 3600
        let baseHourlyRate = shift.hourlyRate
        let hourlyPayCoefficient = Self.defaultHourlyPayCoefficient
        let effectiveHourlyRate = baseHourlyRate * hourlyPayCoefficient
        let isHourlyPaySuppressed = shouldSuppressHourlyPay(for: shift)
        let hourlyPay = isHourlyPaySuppressed ? .zero : workedHours * effectiveHourlyRate

        let sortedTasks = ManualTaskNumbering.manualTasks(for: shift)
        let eligibleTasks = eligibleTasks(in: sortedTasks, for: shift)
        let eligibleTaskIDs = Set(eligibleTasks.map(\.id))
        let ignoredTaskLines = sortedTasks
            .filter { !eligibleTaskIDs.contains($0.id) }
            .map { task in
                PayrollIgnoredTaskLine(
                    id: task.id.uuidString,
                    externalTaskID: task.externalTaskID,
                    taskType: task.taskType,
                    status: task.status,
                    reasonKey: L.payrollIgnoredReason
                )
            }

        let includedTasksCount = min(
            eligibleTasks.count,
            Self.includedEligibleTasksPerShift
        )
        let includedTasks = Array(eligibleTasks.prefix(Self.includedEligibleTasksPerShift))
        let paidTasks = Array(eligibleTasks.dropFirst(Self.includedEligibleTasksPerShift))

        let includedTaskLines = includedTasks.map { task in
            PaidTaskLine(
                id: task.id.uuidString,
                externalTaskID: task.externalTaskID,
                durationSeconds: task.durationSeconds,
                taskType: task.taskType,
                status: task.status,
                taskDate: task.taskDate,
                price: .zero,
                compensationType: .includedInHourlyPay,
                bracketDescription: nil
            )
        }
```

```
    let paidTaskLines = try paidTasks.map { task in
        let pricingMatch = try pricingEngine.pricingMatch(
            for: task.durationSeconds,
            shiftType: shift.shiftType,
            brackets: brackets
        )

        return PaidTaskLine(
            id: task.id.uuidString,
            externalTaskID: task.externalTaskID,
            durationSeconds: task.durationSeconds,
            taskType: task.taskType,
            status: task.status,
            taskDate: task.taskDate,
            price: pricingMatch.price,
            compensationType: .paidByBracket,
            bracketDescription: pricingMatch.bracketDescription
        )
    }

    let paidTasksAmount = paidTaskLines.reduce(Decimal.zero) { partialResult, task in
        partialResult + task.price
    }

    return PayrollBreakdown(
        workedSeconds: workedSeconds,
        workedHours: workedHours,
        baseHourlyRate: baseHourlyRate,
        effectiveHourlyRate: effectiveHourlyRate,
        hourlyRate: baseHourlyRate,
        hourlyPayCoefficient: hourlyPayCoefficient,
        hourlyPay: hourlyPay,
        isHourlyPaySuppressed: isHourlyPaySuppressed,
        hourlySuppressionReasonKey: isHourlyPaySuppressed ? L.payrollHourlyPaySuppressed : nil,
        totalImportedTasks: sortedTasks.count,
        eligibleTasksCount: eligibleTasks.count,
        includedTasksCount: includedTasksCount,
        paidTasksCount: paidTaskLines.count,
        ignoredTasksCount: ignoredTaskLines.count,
        paidTasksAmount: paidTasksAmount,
        totalShiftAmount: hourlyPay + paidTasksAmount,
        includedTaskLines: includedTaskLines,
        paidTaskLines: paidTaskLines,
        ignoredTaskLines: ignoredTaskLines
    )
}

func calculateBreakdown(
    for shift: WorkShift,
    settings: AppSettings?,
    brackets: [TaskPricingBracket]
) throws -> PayrollBreakdown {
    guard let settings else {
        throw PayrollEngineError.missingSettings
    }

    return try calculateBreakdown(
        for: shift,
        settings: settings,
        brackets: brackets
    )
}

func isEligibleTask(_ task: TaskRecord) -> Bool {
    task.isEligible
}

private func shouldSuppressHourlyPay(for shift: WorkShift) -> Bool {
    shift.hasAttendanceIssue && !shift.hasAcceptableExplanation
}

private func eligibleTasks(in tasks: [TaskRecord], for shift: WorkShift) -> [TaskRecord] {
    var seenTaskIDs = Set<String>()

    return tasks.filter { task in
        guard isEligibleTask(task),
              isTaskWithinShift(task, shift: shift) else {
            return false
        }

        let normalizedTaskID = TaskRecord.normalizedExternalTaskID(task.externalTaskID)
        guard !normalizedTaskID.isEmpty, !seenTaskIDs.contains(normalizedTaskID) else {
            return false
```

```
            }

            seenTaskIDs.insert(normalizedTaskID)
            return true
        }
    }

    private func isTaskWithinShift(_ task: TaskRecord, shift: WorkShift) -> Bool {
        guard let taskDate = task.taskDate else {
            return false
        }

        return taskDate >= shift.startTime && taskDate <= shift.effectiveEndTime
    }

}

// Services/Payroll/PricingEngine.swift

import Foundation

enum PricingEngineError: LocalizedError {
    case missingBracket(shiftType: ShiftType, durationSeconds: Int)

    var errorDescription: String? {
        switch self {
        case let .missingBracket(shiftType, durationSeconds):
            return "\(L.pricingMissingBracket.rawValue)|\(shiftType.rawValue)|\(durationSeconds)"
        }
    }
}

struct PricingMatch: Sendable {
    let price: Decimal
    let bracketDescription: String?
}

struct PricingEngine {
    static func priceLimits(for shiftType: ShiftType) -> (minimum: Decimal, maximum: Decimal) {
        switch shiftType {
        case .morning, .main:
            return (decimal("1.10"), decimal("1.70"))
        case .night:
            return (decimal("1.15"), decimal("2.10"))
        }
    }

    func price(
        for durationSeconds: Int,
        shiftType: ShiftType,
        brackets _: [TaskPricingBracket]
    ) throws -> Decimal {
        try pricingMatch(
            for: durationSeconds,
            shiftType: shiftType,
            brackets: []
        ).price
    }

    func pricingMatch(
        for durationSeconds: Int,
        shiftType: ShiftType,
        brackets _: [TaskPricingBracket]
    ) throws -> PricingMatch {
        guard durationSeconds > 0 else {
            return PricingMatch(
                price: .zero,
                bracketDescription: nil
            )
        }

        let rawPrice = Decimal(durationSeconds) / Decimal(240)
        let roundedPrice = rounded(rawPrice, scale: 2)
        let limits = Self.priceLimits(for: shiftType)
        let clampedPrice = min(max(roundedPrice, limits.minimum), limits.maximum)

        return PricingMatch(
            price: clampedPrice,
            bracketDescription: nil
        )
    }

    private func rounded(_ value: Decimal, scale: Int) -> Decimal {
```

```
        var value = value
        var result = Decimal()
        NSDecimalRound(&result, &value, scale, .plain)
        return result
    }

    private static func decimal(_ value: String) -> Decimal {
        Decimal(string: value) ?? .zero
    }
}
```

Додаток Г

ФРАГМЕНТИ КОДУ ФОРМУВАННЯ PDF-ЗВІТУ

Лістинг Л.1. PayrollReportService та фрагменти PayrollPDFRenderer

```swift
// Services/Reports/PayrollReportService.swift

import Foundation

struct PayrollReportPeriod: Sendable {
    let startDate: Date
    let endDate: Date
}

struct PayrollReportSummary: Sendable {
    let totalEarningsUSD: Decimal
    let workedSeconds: Int
    let shiftsCount: Int
    let totalTasks: Int
    let paidExtraTasks: Int
    let averageEarningsUSDPerShift: Decimal
}

struct PayrollReportShiftRow: Identifiable, Sendable {
    let id: UUID
    let date: Date
    let employeeName: String
    let shiftType: ShiftType
    let workedSeconds: Int
    let totalTasks: Int
    let paidExtraTasks: Int
    let totalAmountUSD: Decimal
}

struct PayrollReport: Sendable {
    let period: PayrollReportPeriod
    let generatedAt: Date
    let currency: AppCurrency
    let summary: PayrollReportSummary
    let rows: [PayrollReportShiftRow]
}

enum PayrollReportServiceError: Error {
    case noData
}

struct PayrollReportService {
    private let payrollEngine: PayrollEngine

    init(payrollEngine: PayrollEngine = PayrollEngine()) {
        self.payrollEngine = payrollEngine
    }

    func makeReport(
        period: PayrollReportPeriod,
        shifts: [WorkShift],
        settings: AppSettings?,
        brackets: [TaskPricingBracket]
    ) throws -> PayrollReport {
        let filteredShifts = shifts
            .filter { shift in
                shift.date >= period.startDate && shift.date <= period.endDate
            }
            .sorted { lhs, rhs in
                if lhs.date == rhs.date {
                    return lhs.startTime < rhs.startTime
                }
                return lhs.date < rhs.date
            }

        guard !filteredShifts.isEmpty else {
            throw PayrollReportServiceError.noData
        }

        var totalEarningsUSD: Decimal = .zero
        var workedSeconds = 0
        var totalTasks = 0
        var paidExtraTasks = 0
        var rows: [PayrollReportShiftRow] = []
```

```
        for shift in filteredShifts {
            let breakdown = try? settings.map { currentSettings in
                try payrollEngine.calculateBreakdown(
                    for: shift,
                    settings: currentSettings,
                    brackets: brackets
                )
            }

            let worked = breakdown?.workedSeconds ?? shift.workedDurationSeconds
            let taskCount = breakdown?.addedTasksCount ?? shift.tasks.count
            let paidTasks = breakdown?.paidExtraTasksCount ?? 0
            let amount = breakdown?.totalShiftAmount ?? .zero

            totalEarningsUSD += amount
            workedSeconds += worked
            totalTasks += taskCount
            paidExtraTasks += paidTasks

            rows.append(
                PayrollReportShiftRow(
                    id: shift.id,
                    date: shift.date,
                    employeeName: shift.employee?.fullName ?? "",
                    shiftType: shift.shiftType,
                    workedSeconds: worked,
                    totalTasks: taskCount,
                    paidExtraTasks: paidTasks,
                    totalAmountUSD: amount
                )
            )
        }

        let average = filteredShifts.isEmpty ? Decimal.zero : totalEarningsUSD /
Decimal(filteredShifts.count)
        let summary = PayrollReportSummary(
            totalEarningsUSD: totalEarningsUSD,
            workedSeconds: workedSeconds,
            shiftsCount: filteredShifts.count,
            totalTasks: totalTasks,
            paidExtraTasks: paidExtraTasks,
            averageEarningsUSDPerShift: average
        )

        return PayrollReport(
            period: period,
            generatedAt: .now,
            currency: settings?.selectedCurrency ?? .usd,
            summary: summary,
            rows: rows
        )
    }
}

// Services/Reports/PayrollPDFRenderer.swift, фрагменти

private let pageBounds = CGRect(x: 0, y: 0, width: 595, height: 842)
    private let margin: CGFloat = 36
    private let rowHeight: CGFloat = 30

    private enum PDFColor {
        static let pageBackground = UIColor.white
        static let primaryText = UIColor(red: 0.07, green: 0.08, blue: 0.10, alpha: 1.0)
        static let secondaryText = UIColor(red: 0.27, green: 0.30, blue: 0.35, alpha: 1.0)
        static let blockBackground = UIColor(red: 0.96, green: 0.97, blue: 0.98, alpha: 1.0)
        static let tableHeaderBackground = UIColor(red: 0.92, green: 0.94, blue: 0.96, alpha: 1.0)
        static let alternateRowBackground = UIColor(red: 0.98, green: 0.985, blue: 0.99, alpha: 1.0)
        static let border = UIColor(red: 0.78, green: 0.80, blue: 0.84, alpha: 1.0)
    }



    func renderToTemporaryFile() throws -> URL {
        let fileURL = FileManager.default.temporaryDirectory.appendingPathComponent(fileName)
        let renderer = UIGraphicsPDFRenderer(bounds: pageBounds)

        try renderer.writePDF(to: fileURL) { context in
            context.beginPage()
            drawPageBackground()
            var y = margin

            y = drawTitle(at: y)
```

```
            y = drawInfoBlock(at: y + 12)
            y = drawSummary(at: y + 14)
            y = drawTableHeader(at: y + 18)

            for (index, row) in report.rows.enumerated() {
                if y + rowHeight > pageBounds.height - margin - 72 {
                    context.beginPage()
                    drawPageBackground()
                    y = margin
                    y = drawTableHeader(at: y)
                }

                y = drawRow(row, at: y, isAlternate: index.isMultiple(of: 2))
            }

            if y + 74 > pageBounds.height - margin {
                context.beginPage()
                drawPageBackground()
                y = margin
            }

            _ = drawConclusion(at: y + 18)
        }

        return fileURL
    }

    private func drawInfoBlock(at y: CGFloat) -> CGFloat {
        let lines = [
            "\(languageManager.string(L.reportsPeriod)): \(date(report.period.startDate)) -
                \(date(report.period.endDate))",
            "\(languageManager.string(L.reportsGeneratedAt)): \(dateTime(report.generatedAt))",
            "\(languageManager.string(L.reportsCurrency)): \(report.currency.currencyCode)",
            "\(languageManager.string(L.reportsShiftsCount)): \(report.summary.shiftsCount)"
        ]

        return drawKeyValueBlock(lines, at: y)
    }

    private func drawSummary(at y: CGFloat) -> CGFloat {
        let lines = [
            "\(languageManager.string(L.reportsTotalEarnings)):
\(convertedCurrency(report.summary.totalEarningsUSD))",
            "\(languageManager.string(L.reportsWorkedHours)): \(hours(report.summary.workedSeconds))",
            "\(languageManager.string(L.reportsTotalTasks)): \(report.summary.totalTasks)",
            "\(languageManager.string(L.reportsPaidTasks)): \(report.summary.paidExtraTasks)",
            "\(languageManager.string(L.reportsAveragePerShift)):
                \(convertedCurrency(report.summary.averageEarningsUSDPerShift))"
        ]

        return drawKeyValueBlock(lines, at: y)
    }

    private func drawTableHeader(at y: CGFloat) -> CGFloat {
        let rect = CGRect(x: margin, y: y, width: pageBounds.width - margin * 2, height: rowHeight)
        PDFColor.tableHeaderBackground.setFill()
        UIBezierPath(rect: rect).fill()
        PDFColor.border.setStroke()
        UIBezierPath(rect: rect).stroke()

        drawTableCells(
            [
                languageManager.string(L.commonDateTime),
                languageManager.string(L.commonAgent),
                languageManager.string(L.commonType),
                languageManager.string(L.dashboardHours),
                languageManager.string(L.dashboardTotalTasks),
                languageManager.string(L.dashboardPaidTasks),
                languageManager.string(L.shiftDetailsShiftTotal)
            ],
            at: y,
            font: .boldSystemFont(ofSize: 8.5),
            color: PDFColor.primaryText
        )
        return y + rowHeight
    }

    private func drawRow(_ row: PayrollReportShiftRow, at y: CGFloat, isAlternate: Bool) -> CGFloat {
        let rect = CGRect(x: margin, y: y, width: pageBounds.width - margin * 2, height: rowHeight)
        (isAlternate ? PDFColor.alternateRowBackground : PDFColor.pageBackground).setFill()
        UIBezierPath(rect: rect).fill()
        PDFColor.border.setStroke()
        UIBezierPath(rect: rect).stroke()
```

```
        drawTableCells(
            [
                date(row.date),
                row.employeeName.isEmpty ? languageManager.string(L.commonUnassigned) : row.employeeName,
                languageManager.string(row.shiftType.localizationKey),
                hours(row.workedSeconds),
                "\(row.totalTasks)",
                "\(row.paidExtraTasks)",
                convertedCurrency(row.totalAmountUSD)
            ],
            at: y,
            font: .systemFont(ofSize: 8.5),
            color: PDFColor.primaryText
        )
        return y + rowHeight
    }
```

Додаток Д

ФРАГМЕНТ КОДУ ОНОВЛЕННЯ КУРСІВ ВАЛЮТ

Лістинг М.1. ExchangeRateService

```
import Foundation
import Observation

struct NBUExchangeRate: Decodable {
    let r030: Int
    let txt: String
    let rate: Decimal
    let cc: String
    let exchangedate: String
}

struct ExchangeRates: Codable, Equatable {
    static let fallback = ExchangeRates(
        usdToUahRate: Decimal(string: "40.0") ?? 40,
        eurToUahRate: Decimal(string: "43.5") ?? 43.5,
        exchangeRateDate: nil,
        lastUpdatedAt: nil,
        source: .fallback
    )

    enum Source: String, Codable {
        case api
        case cache
        case fallback
    }

    var usdToUahRate: Decimal
    var eurToUahRate: Decimal
    var exchangeRateDate: String?
    var lastUpdatedAt: Date?
    var source: Source
}

enum ExchangeRateServiceError: Error {
    case missingRates
}

@Observable
final class ExchangeRateService {
    private enum StorageKeys {
        static let exchangeRates = "exchange_rates_cache"
    }

    @ObservationIgnored private let userDefaults: UserDefaults
    @ObservationIgnored private let session: URLSession
    @ObservationIgnored private let cacheFreshInterval: TimeInterval = 4 * 60 * 60
    @ObservationIgnored private let endpoint = URL(string:
        "https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json")

    private(set) var rates: ExchangeRates
    private(set) var isLoading = false
    private(set) var lastError: String?

    init(userDefaults: UserDefaults = .standard, session: URLSession = .shared) {
        self.userDefaults = userDefaults
        self.session = session
        self.rates = Self.loadCachedRates(from: userDefaults) ?? .fallback
    }

    func refreshIfNeeded() async {
        guard shouldRefresh else { return }
        await refresh()
    }

    func refresh() async {
        guard !isLoading else { return }

        isLoading = true
        lastError = nil
        defer { isLoading = false }

        do {
            guard let endpoint else {
                throw ExchangeRateServiceError.missingRates
```

```
            }

            let (data, response) = try await session.data(from: endpoint)
            if let httpResponse = response as? HTTPURLResponse,
               !(200...299).contains(httpResponse.statusCode) {
                throw URLError(.badServerResponse)
            }

            let nbuRates = try JSONDecoder().decode([NBUExchangeRate].self, from: data)
            guard let usd = nbuRates.first(where: { $0.cc.uppercased() == AppCurrency.usd.currencyCode }),
                  let eur = nbuRates.first(where: { $0.cc.uppercased() == AppCurrency.eur.currencyCode })
else {
                throw ExchangeRateServiceError.missingRates
            }

            rates = ExchangeRates(
                usdToUahRate: usd.rate,
                eurToUahRate: eur.rate,
                exchangeRateDate: usd.exchangedate,
                lastUpdatedAt: .now,
                source: .api
            )
            saveRatesToCache()
        } catch {
            lastError = error.localizedDescription
            rates = Self.loadCachedRates(from: userDefaults) ?? .fallback
        }
    }

    func convertedFromUSD(_ amount: Decimal, to currency: AppCurrency) -> Decimal {
        switch currency {
        case .usd:
            return amount
        case .uah:
            return amount * rates.usdToUahRate
        case .eur:
            guard rates.eurToUahRate > .zero else { return amount * (Decimal(string: "0.92") ?? 0.92) }
            return amount * rates.usdToUahRate / rates.eurToUahRate
        }
    }

    private var shouldRefresh: Bool {
        guard !isLoading else { return false }
        guard let lastUpdatedAt = rates.lastUpdatedAt else { return true }
        return Date().timeIntervalSince(lastUpdatedAt) > cacheFreshInterval
    }

    private func saveRatesToCache() {
        guard let data = try? JSONEncoder().encode(rates) else { return }
        userDefaults.set(data, forKey: StorageKeys.exchangeRates)
    }

    private static func loadCachedRates(from userDefaults: UserDefaults) -> ExchangeRates? {
        guard let data = userDefaults.data(forKey: StorageKeys.exchangeRates),
              var cachedRates = try? JSONDecoder().decode(ExchangeRates.self, from: data) else {
            return nil
        }

        cachedRates.source = .cache
        return cachedRates
    }
}
```

Додаток Е

ФРАГМЕНТИ КОДУ ЗАХИСТУ ДОСТУПУ ДО ЗАСТОСУНКУ

Лістинг Н.1. KeychainPINStore та SecurityManager

```
// Services/Security/KeychainPINStore.swift

import Foundation
import Security

enum KeychainPINStoreError: Error {
    case encodingFailed
    case unhandledStatus(OSStatus)
}

struct KeychainPINStore {
    private let service = "dmitrich_dyplom.security.pin"
    private let account = "app_pin"

    nonisolated init() {}

    func savePINHash(_ hash: String) throws {
        guard let data = hash.data(using: .utf8) else {
            throw KeychainPINStoreError.encodingFailed
        }

        let query = baseQuery()
        SecItemDelete(query as CFDictionary)

        var item = query
        item[kSecValueData as String] = data
        item[kSecAttrAccessible as String] = kSecAttrAccessibleWhenUnlockedThisDeviceOnly

        let status = SecItemAdd(item as CFDictionary, nil)
        guard status == errSecSuccess else {
            throw KeychainPINStoreError.unhandledStatus(status)
        }
    }

    func loadPINHash() -> String? {
        var query = baseQuery()
        query[kSecReturnData as String] = true
        query[kSecMatchLimit as String] = kSecMatchLimitOne

        var result: AnyObject?
        let status = SecItemCopyMatching(query as CFDictionary, &result)

        guard status == errSecSuccess,
              let data = result as? Data,
              let hash = String(data: data, encoding: .utf8) else {
            return nil
        }

        return hash
    }

    func deletePINHash() {
        SecItemDelete(baseQuery() as CFDictionary)
    }

    private func baseQuery() -> [String: Any] {
        [
            kSecClass as String: kSecClassGenericPassword,
            kSecAttrService as String: service,
            kSecAttrAccount as String: account
        ]
    }
}

// Services/Security/SecurityManager.swift

import CryptoKit
import Foundation
import LocalAuthentication
import Observation
import UIKit

@Observable
@MainActor
```

```
final class SecurityManager {
    private enum StorageKeys {
        static let isPINEnabled = "security.is_pin_enabled"
        static let isBiometryEnabled = "security.is_biometry_enabled"
        static let locksOnLaunch = "security.locks_on_launch"
        static let lockDelay = "security.lock_delay"
    }

    @ObservationIgnored private let userDefaults: UserDefaults
    @ObservationIgnored private let pinStore: KeychainPINStore
    @ObservationIgnored private var backgroundEnteredAt: Date?

    private(set) var isPINEnabled: Bool
    var isBiometryEnabled: Bool {
        didSet {
            userDefaults.set(isBiometryEnabled, forKey: StorageKeys.isBiometryEnabled)
        }
    }
    var locksOnLaunch: Bool {
        didSet {
            userDefaults.set(locksOnLaunch, forKey: StorageKeys.locksOnLaunch)
        }
    }
    var lockDelay: AppLockDelay {
        didSet {
            userDefaults.set(lockDelay.rawValue, forKey: StorageKeys.lockDelay)
        }
    }
    private(set) var isUnlocked: Bool
    private(set) var biometryType: LABiometryType = .none
    private(set) var isBiometryAvailable = false

    init(userDefaults: UserDefaults = .standard, pinStore: KeychainPINStore = KeychainPINStore()) {
        self.userDefaults = userDefaults
        self.pinStore = pinStore
        let storedPINEnabled = userDefaults.bool(forKey: StorageKeys.isPINEnabled)
        self.isPINEnabled = storedPINEnabled
        self.isBiometryEnabled = userDefaults.bool(forKey: StorageKeys.isBiometryEnabled)

        let storedLocksOnLaunch: Bool
        if userDefaults.object(forKey: StorageKeys.locksOnLaunch) == nil {
            storedLocksOnLaunch = true
        } else {
            storedLocksOnLaunch = userDefaults.bool(forKey: StorageKeys.locksOnLaunch)
        }
        self.locksOnLaunch = storedLocksOnLaunch
        self.lockDelay = AppLockDelay(
            rawValue: userDefaults.string(forKey: StorageKeys.lockDelay) ?? ""
        ) ?? .thirtyMinutes

        self.isUnlocked = !(storedPINEnabled && storedLocksOnLaunch)
        refreshBiometryAvailability()
    }

    var requiresUnlock: Bool {
        isPINEnabled && !isUnlocked
    }

    var canUseBiometryUnlock: Bool {
        isPINEnabled && isBiometryEnabled && isBiometryAvailable
    }

    func lockIfNeededOnLaunch() {
        guard isPINEnabled, locksOnLaunch else {
            isUnlocked = true
            return
        }

        isUnlocked = false
    }

    func sceneDidEnterBackground() {
        backgroundEnteredAt = .now

        if isPINEnabled, locksOnLaunch, lockDelay == .immediately {
            isUnlocked = false
        }
    }

    func sceneDidBecomeActive() {
        guard isPINEnabled, locksOnLaunch, let backgroundEnteredAt else {
            self.backgroundEnteredAt = nil
            return
```

```
        }

        if lockDelay == .immediately || Date().timeIntervalSince(backgroundEnteredAt) >= lockDelay.seconds
{
            isUnlocked = false
        }

        self.backgroundEnteredAt = nil
    }

    func enablePIN(_ pin: String) throws {
        try validatePIN(pin)
        try pinStore.savePINHash(hash(pin))
        isPINEnabled = true
        isUnlocked = true
        userDefaults.set(true, forKey: StorageKeys.isPINEnabled)
    }

    func disablePIN() {
        pinStore.deletePINHash()
        isPINEnabled = false
        isBiometryEnabled = false
        isUnlocked = true
        userDefaults.set(false, forKey: StorageKeys.isPINEnabled)
    }

    func updatePIN(currentPIN: String, newPIN: String) throws -> Bool {
        guard verifyPIN(currentPIN) else { return false }
        try enablePIN(newPIN)
        return true
    }

    func unlock(with pin: String) -> Bool {
        guard verifyPIN(pin) else { return false }
        isUnlocked = true
        return true
    }

    func authenticateWithBiometrics(reason: String) async -> Bool {
        refreshBiometryAvailability()
        guard canUseBiometryUnlock else { return false }

        let context = LAContext()
        var error: NSError?
        guard context.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, error: &error) else {
            isBiometryAvailable = false
            return false
        }

        return await withCheckedContinuation { continuation in
            context.evaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, localizedReason: reason) {
success, _ in
                Task { @MainActor in
                    if success {
                        self.isUnlocked = true
                    }
                    continuation.resume(returning: success)
                }
            }
        }
    }

    func refreshBiometryAvailability() {
        let context = LAContext()
        var error: NSError?
        isBiometryAvailable = context.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, error:
&error)
        biometryType = context.biometryType

        if biometryType == .faceID,
           Bundle.main.object(forInfoDictionaryKey: "NSFaceIDUsageDescription") == nil {
            isBiometryAvailable = false
        }

        if !isBiometryAvailable {
            isBiometryEnabled = false
        }
    }

    private func validatePIN(_ pin: String) throws {
        guard pin.count == 4, pin.allSatisfy(\.isNumber) else {
            throw SecurityPINError.invalidFormat
        }
```

```
    }

    private func verifyPIN(_ pin: String) -> Bool {
        guard pin.count == 4,
              let storedHash = pinStore.loadPINHash() else {
            return false
        }

        return hash(pin) == storedHash
    }

    private func hash(_ pin: String) -> String {
        let data = Data(pin.utf8)
        let digest = SHA256.hash(data: data)
        return digest.map { String(format: "%02x", $0) }.joined()
    }
}

enum SecurityPINError: Error {
    case invalidFormat
}

enum AppLockDelay: String, CaseIterable, Codable, Identifiable {
    case immediately
    case oneMinute
    case fiveMinutes
    case thirtyMinutes

    var id: String { rawValue }

    var seconds: TimeInterval {
        switch self {
        case .immediately:
            return 0
        case .oneMinute:
            return 60
        case .fiveMinutes:
            return 300
        case .thirtyMinutes:
            return 1_800
        }
    }

    var localizedTitleKey: LocalizedTextKey {
        switch self {
        case .immediately:
            return L.securityLockDelayImmediately
        case .oneMinute:
            return L.securityLockDelayOneMinute
        case .fiveMinutes:
            return L.securityLockDelayFiveMinutes
        case .thirtyMinutes:
            return L.securityLockDelayThirtyMinutes
        }
    }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Буренков Ілля Дмитрович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «**Програмне забезпечення для обліку робочого часу та розрахунку заробітної плати операторів логістичної компанії** » є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [ ] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
  - [ ] перекладу іншомовних джерел;
  - [ ] стилістичного редагування та перевірки граматики;
  - [ ] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____2026 р. _____ **Ілля БУРЕНКОВ**
(підпис)