

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
МЕТОДІВ ПЛАНУВАННЯ РЕСУРСІВ ІТ-КОМПАНІЇ
RESEARCH AND SOFTWARE IMPLEMENTATION OF
PLANNING METHODS FOR AN IT COMPANY

Виконав: студент(ка) 2 курсу, групи КНТ-119м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Бессонов О.О.

(прізвище та ініціали)

Керівник Сердюк С.М.

(прізвище та ініціали)

Рецензент Казурова А.Є.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20__ року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Бессонова Олександра Олександровича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів планування ресурсів ІТ-компанії. Research and Software Implementation of Resource Planning Methods for an IT Company

керівник проєкту (роботи) Сердюк Сергій Микитович, к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “ 03 ” листопада 2020 року № 301

2. Строк подання студентом проєкту (роботи) 10 грудня 2020 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали та методи. 3. Розробка архітектури програми. 4. Основні рішення щодо реалізації компонентів програмного комплексу. 5. Експлуатація, тестування та експериментальне дослідження програмного комплексу. 6. Економіко-організаційна частина. 7. Охорона праці та безпека в надзвичайних ситуаціях.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Сердюк С.М., к.т.н., доцент		
Економіко-організаційна частина	Пожуєва Т.О., д.е.н., професор		
Охорона праці та безпека в надзвичайних ситуаціях	Коробко О.В., ст. викладач		
Нормоконтролер	Калініна М.В., асистент		

7. Дата видачі завдання “04” вересня 2020 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	
2	Аналіз предметної області	2-3 тижні	Розділи 1,2 ПЗ
3	Вибір та обґрунтування мов програмування та технологій	4 тиждень	Розділ 2 ПЗ
4	Моделювання ERP-системи	5 тиждень	Розділ 3 ПЗ
5	Розробка програмного комплексу ERP-системи	8-11 тижні	Розділ 4 ПЗ
6	Модернізація та налагодження програмного комплексу ERP-системи	12-13 тижні	Розділ 4 ПЗ
7	Тестування розробленого програмного комплексу ERP-системи	14 тиждень	Розділ 5 ПЗ
8	Завершення оформлення пояснювальної записки	15-16 тижні	Розділи 6,7 ПЗ, додатки
9	Захист магістерської роботи	17 тиждень	

Студент

(підпис)

Бессонов О.О.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Сердюк С.М.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
149 с., 54 рис., 15 табл., 2 додатка, 38 джерел.

ERP-СИСТЕМИ, КОМП'ЮТЕРНА ПРОГРАМА, УПРАВЛІННЯ
ДАНИМИ, СЕМАНТИЧНІ МЕРЕЖІ, ПРОГРАМНИЙ КОМПЛЕКС.

Об'єкт дослідження – процеси балансування, оптимізації та автоматизації
ресурсів підприємства.

Предмет дослідження – методи підвищення якості фінансового аудиту,
підвищення ефективності менеджменту ресурсів та підвищення продуктивності
управлінського персоналу.

Мета роботи – підвищення ефективності менеджменту ресурсів та
оптимізація робочого процесу управлінського персоналу на ІТ підприємстві.

Матеріали, методи та технічні засоби: методи представлення знань; мова
графічного описання та моделювання бізнес-процесів UML версії 2.0;
інструментарій інтегрованого середовища розробки PhpStorm; мови
програмування JavaScript та PHP; операційна система Mac OS Catalina 10.14 або
вище.

Результати. Досліджено поняття ERP-системи та проаналізовано
поширені варіанти архітектури системи; розроблено нову бізнес-модель
системи, на основі якої реалізовано та розгорнуто програмний комплекс ERP-
системи.

Висновки. Використання розробленої ERP-системи дозволяє
оптимізувати та збалансувати управління ресурсами, підвищити якість
фінансового аудиту та оптимізувати процес менеджменту на підприємстві.

Економічна ефективність використання ERP-системи полягає у
підвищенні продуктивності управлінського персоналу шляхом зменшення
кількості інвестицій робочого часу в рутинні завдання менеджменту.

Галузь використання: планування ресурсів ІТ-компанії.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 149 pages, 54 figures, 16 tables, 2 appendices, 38 sources.

ERP-SYSTEMS, COMPUTER PROGRAM, DATA MANAGEMENT, SEMANTIC NETWORKS, SOFTWARE COMPLEX.

The object of study are the processes of balancing, optimization, and automation of enterprise resources.

The subject of research are methods to improve the quality of the financial audit, increase the efficiency of resource management, and increase the productivity of management staff.

The purpose of the work - research, development, software implementation, and deployment in the cloud environment of the software package to improve the quality of the financial audit, improve the efficiency of resource management, and optimize the workflow of management staff in the IT enterprise.

Materials, methods, and tools method of knowledge representation "semantic network"; a language of graphic description and modeling of business processes UML version 2.0; PhpStorm integrated development toolkit; JavaScript and PHP programming language; Mac OS Catalina 10.14 or higher operating system.

Results. The concept of ERP-system is studied and the widespread variants of system architecture are analyzed; a business model of the system was developed, based on which the software complex of the ERP-system was implemented and deployed.

Conclusions. The use of the developed ERP-system allows to optimize and balance the abolition of resource management, improve the quality of the financial audit, and optimize management processes in the enterprise.

The cost-effectiveness of ERP-systems helps to increase the productivity of administrative staff management by reducing the amount of investment of working time in the task management process.

Scope: resource planning of IT companies.

ЗМІСТ

	С.
Перелік скорочень та умовних позначень.....	8
Вступ	9
1 Аналіз предметної області	11
1.1 Основні поняття та класифікація ERP-систем	11
1.2 Аналіз та порівняння методів представлення знань.....	19
1.3 Висновки за розділом 1	25
2 Матеріали та методи	27
2.1 Формалізація знань про предметну область «Робочий процес ІТ підприємства»	27
2.2 Вибір мов програмування, технологій та середовища розробки	30
2.3 Висновки за розділом 2.....	34
3 Розробка архітектури програми	36
3.1 Опис основних вимог до системи.....	36
3.2 Проєктування структури ERP-системи	36
3.3 Моделювання програмного забезпечення ERP-системи.....	38
3.4 Проєктування бази даних	40
3.5 Вимоги до технічних засобів.....	41
3.6 Висновки за розділом 3.....	42
4 Основні рішення щодо реалізації компонентів програмного комплексу	43
4.1 Вибір бібліотек для реалізації функціональних модулів	43
4.2 Основні етапи розробки програмного комплексу.....	44
4.3 Розгортання програмного комплексу	61
4.4 Висновки за розділом 4.....	64
5 Експлуатація, тестування та експериментальне дослідження програмного комплексу	65
5.1 Опис застосування програмного комплексу.....	65
5.2 Інструкція по роботі з системою ERP	66

5.3	Методика тестування програмного комплексу	69
5.4	Практичні завдання для тестування та експериментального дослідження програмного комплексу	70
5.5	Результати тестування та експериментального дослідження програми	71
5.6	Висновки за розділом 5	72
6	Економіко-організаційна частина	73
6.1	Опис ідеї	73
6.2	Аналіз ринкових можливостей запуску проєкту	73
6.3	Планування розробки програмного комплексу	75
6.4	Визначення витрат на оплату праці	78
6.5	Визначення витрат на матеріали	78
6.6	Витрати на технічне обладнання	79
6.7	Розрахунок накладних витрат	82
6.8	Розрахунок економічної ефективності програмного продукту	83
6.9	Висновки за розділом 6	84
7	Охорона праці та безпека в надзвичайних ситуаціях	85
7.1	Аналіз потенційних небезпек	85
7.2	Заходи із забезпечення безпеки	86
7.3	Заходи з забезпечення виробничої санітарії та гігієни праці	87
7.4	Заходи безпеки у надзвичайних ситуаціях	90
	Висновки	98
	Перелік джерел посилання	100
	Додаток А Код програми	104
	Додаток Б Слайди презентації	138

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API	– Application Programming Interface;
CRUD	– Create, Read, Update, Delete;
ERP	– Enterprise Resource Planning;
HTTP	– Hyper Text Transfer Protocol;
IDE	– Integrated Development Environment;
JSON	– JavaScript Object Notation;
NPM	– Node Package Manager;
PHP	– Hypertext Preprocessor;
RBAC	– Role Based Access Control;
REST	– Representational State Transfer;
SQL	– Structured Query Language;
БД	– база даних;
ОС	– операційна система.

ВСТУП

На сьогоднішній день кожне підприємство, від невеликої компанії до величезної корпорації, прагне зайняти на ринку високу позицію, а для цього необхідно правильно і раціонально управляти своїм виробництвом. У сучасних умовах ефективне управління являє собою цінний ресурс організації. Отже, підвищення ефективності управлінської діяльності стає одним з напрямків вдосконалення діяльності підприємства в цілому. Найбільш очевидним способом підвищення ефективності протікання трудового процесу є його автоматизація.

Автоматизація рутинних завдань робочого процесу є ключовим фактором який сприяє загальному підвищенню продуктивності персоналу на фоні якого підприємство має змогу направляти вільні ресурси на виконання інших завдань. Найважче при вирішенні цього завдання – побудувати єдину систему, яка буде відповідати запитам співробітників всіх підрозділів. Поки що не існує ідеального програмного рішення для всіх бізнес-процесів, проте останнім досягненням в еволюції автоматизованих інтегрованих систем управління підприємством є розробка системи планування ресурсів підприємства (ERP).

Робочий процес в ІТ підприємстві також міцно пов'язаний з управлінською діяльністю, що має значний вплив на загальні виробничі показники. Більшість таких підприємств інтегрують в робочий процес декілька інструментів, які у цілому виконують функції повноцінної ERP системи, або використовують об'ємні та дорогі програмні рішення, більшість функцій якого залишаються невикористаними.

Об'єктом дослідження є процеси балансування, оптимізації та автоматизації ресурсів підприємства.

Предметом дослідження є методи підвищення якості фінансового аудиту, підвищення ефективності менеджменту ресурсів та підвищення продуктивності управлінського персоналу.

Мета роботи – підвищення ефективності менеджменту ресурсів та оптимізація робочого процесу управлінського персоналу на ІТ підприємстві.

Для досягнення поставленої мети у роботі вирішувалися такі завдання:

- провести огляд ERP-системи;
- дослідити та проаналізувати основні архітектури ERP-систем;
- розглянути основні методи представлення знань;
- формалізувати знання про предметну область «Робочий процес підприємства»;
- розробити моделі програмного забезпечення ERP-системи;
- зпроєктувати власну архітектуру ERP-системи на основі формалізованих знань про предметну область;
- розробити програмний комплекс ERP-системи на основі зпроєктованих моделей та архітектури;
- розгорнути у хмарному середовищі розроблений програмний комплекс ERP-системи;
- експериментально дослідити та протестувати розроблений програмний комплекс ERP-системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття та класифікація ERP-систем

1.1.1 ERP-системи

ERP (англ. Enterprise Resource Planning, планування ресурсів підприємства) – це організаційна стратегія інтеграції виробництва та операцій, управління трудовими ресурсами, фінансового менеджменту та управління активами, орієнтована на безперервне балансування та оптимізацію ресурсів підприємства за допомогою спеціалізованого інтегрованого пакета прикладного програмного забезпечення, що забезпечує загальну модель даних і процесів для всіх сфер діяльності. ERP-система – конкретний програмний пакет, який реалізує стратегію ERP [1].

Концепція ERP була сформульована в 1990 році аналітиком Gartner як бачення розвитку методик MRP II і CIM, на початку – середині 1990-х років з'явилося кілька успішних тиражованих ERP-систем для великих організацій, найбільш відомі – розробки компаній Baan, Oracle, PeopleSoft, SAP, JD Edwards, сформувався ринок послуг з впровадження ERP-систем за участю компаній великої четвірки, у 2000-і роки відбулася консолідація постачальників, з'явилася значна кількість ERP-систем для малого та середнього бізнесу, найбільш відомими постачальниками яких стали Sage Group і Microsoft [2], [3].

До цього ж часу в зв'язку з швидким зростанням кількості модулів в ERP-системах і їх функціональних можливостей відноситься уявлення про ERP-системах як всеосяжне програмне забезпечення для організацій, принципово замінює всі інші прикладні програми, які змінилися до початку 2000-х років виділенням таких функцій як CRM і PLM в окремі від ERP програмного забезпечення та окресленням рамок ERP як універсальних систем для бек-офісних процесів і управління ресурсами (а CRM-систем – рамками зовнішніх взаємин і фронт-офісу, і PLM – інтелектуальною власністю) [4], [5].

Впровадження ERP-системи вважається фактично необхідною умовою для публічної компанії і, починаючи з кінця 1990-х років, ERP-системи, спочатку

впроваджувалися тільки промисловими підприємствами, зараз експлуатуються більшістю великих організацій незалежно від країни, форми власності, галузі [1].

Використання ERP системи дозволяє використовувати одну інтегровану програму замість декількох розрізнених. Єдина система може керувати обробкою, логістикою, дистрибуцією, запасами, доставкою, виставлянням рахунків, бухгалтерським обліком, податковим обліком, програмою лояльності.

Реалізована в ERP-системах система розмежування доступу до інформації, призначена (в комплексі з іншими заходами інформаційної безпеки підприємства) для протидії як зовнішнім загрозам (наприклад промисловому шпигунству), так і внутрішнім (наприклад, розкраданням). Впроваджені в зв'язці з CRM-системою і системою контролю якості, ERP-системи націлені на максимальне задоволення потреб компаній в засобах управління бізнесом [6].

Недоліки цього типу програмних продуктів є наслідком їх переваг. Єдина база даних і єдина система породжують значну кількість зв'язків, високу складність самої системи і високі вимоги до апаратної (серверної) частини. А тому для організації роботи ERP-системи необхідно потужне обладнання, що тягне за собою відповідні витрати.

Безліч проблем, пов'язаних з ERP, виникають через недостатнє інвестування в навчання персоналу, а також через недосконалість політики занесення і підтримки актуальності даних в ERP [7].

1.1.2 Ключові принципи ERP-систем

В якості характерної особливості ERP-стратегії наголошується принциповий підхід до використання єдиної транзакційної системи для переважної більшості операцій і бізнес-процесів організації, незалежно від функціональної і територіальної роз'єднаності місць їх виникнення і проходження, обов'язковість відомості всіх операцій в єдину базу для подальшої обробки та отримання в реальному часі збалансованих планів.

Тиражовананість, тобто можливість застосувати один і той же програмний пакет для різних організацій (можливо, з різними настройками і розширеннями), фігурує як одна з обов'язкових умов ERP-системи. Однією з причин повсюдного використання тиражованих ERP-систем замість розробки на замовлення вказується можливість впровадження кращих практик за допомогою реінжинірингу бізнес-процесів відповідно до рішень, застосованих в ERP-системі. Однак, зустрічаються і згадки інтегрованих систем, розроблених для окремої організації на замовлення як ERP-систем [1], [8].

Необхідність всеосяжного застосування ERP-системи в територіально-розподілених організаціях вимагає підтримки в єдиній системі безлічі валют і мов. Більш того, необхідність підтримувати кілька організаційних одиниць (кілька юридичних осіб, кілька підприємств), кілька різних планів рахунків, облікових політик, різних схем оподаткування в єдиному екземплярі системи виявляється необхідною умовою для застосування в холдингах, транснаціональних корпораціях [2].

Застосування в різних галузях накладає на ERP-системи, з одного боку, вимоги до універсальності, з іншого боку – підтримку розширюваності галузевою специфікою. Основні великі системи включають готові спеціалізовані модулі і розширення для різних галузей (відомі спеціалізовані рішення в рамках ERP-систем для машинобудівних і обробних виробництв, підприємств добувної промисловості, роздрібної торгівлі, дистрибуції, банків, фінансових організацій і страхових компаній, підприємств електрозв'язку, енергетики, організацій сектора державного управління, сфери освіти, медицини та інших галузей) [7].

1.1.3 Загальна архітектура ERP-системи

Усі ERM-системи, незалежно від того, хто їх розробник, об'єднує загальна архітектура, яку можна описати таким чином (див. рис. 1.1):

а) платформа. Базові можливості і середовище для роботи модулів і компонентів. У код платформи зміни може вносити тільки розробник.

Користувачі і фахівці з впровадженню не мають доступу до цього програмного коду. До складу платформи входять:

1) ядро. Програмне середовище, в якій буде проводитися робота, для якої можна писати якісь надбудови і компоненти;

2) базовий функціонал. Перелік довідників і функцій, без яких не може працювати жодна компанія. Це довідник користувачів з правами доступу, довідник клієнтів, довідник товарів / послуг тощо. Цей функціонал вбудований в платформу, на відміну від модулів він не може відключатися;

б) управління даними. База даних, в тому числі, зберігання і методи обробки (інтерпретації) даних. У цю категорію входять сховище даних на сервері, програмне забезпечення для роботи з базами даних (SQL або будь-яка альтернатива), інструменти для інтерпретації та обробки даних і відправки їх в програмні модулі;

в) модулі. Компоненти, які підключаються до платформи в міру необхідності. Всі вони працюють з єдиною базою даних і застосовують базовий функціонал (в міру необхідності). В іншому модулі працюють незалежно один від одного, можуть «безшовно» підключатися і без проблем відключатися, якщо потреба в них зникла. Така модульна структура - важлива відмінна риса ERP-систем. Модулі діляться, в свою чергу, на кілька типів:

1) модулі внутрішнього використання. Цей рівень – Plug-in, які використовуються співробітниками компанії. Це управління складом, виробництво, бухгалтерія, CRM тощо. Модулі можна підключати, відключати, налаштовувати силами фахівців з впровадження. У стандартний набір зазвичай входять – MRP, HR, CRM, Управління постачанням і закупівлями;

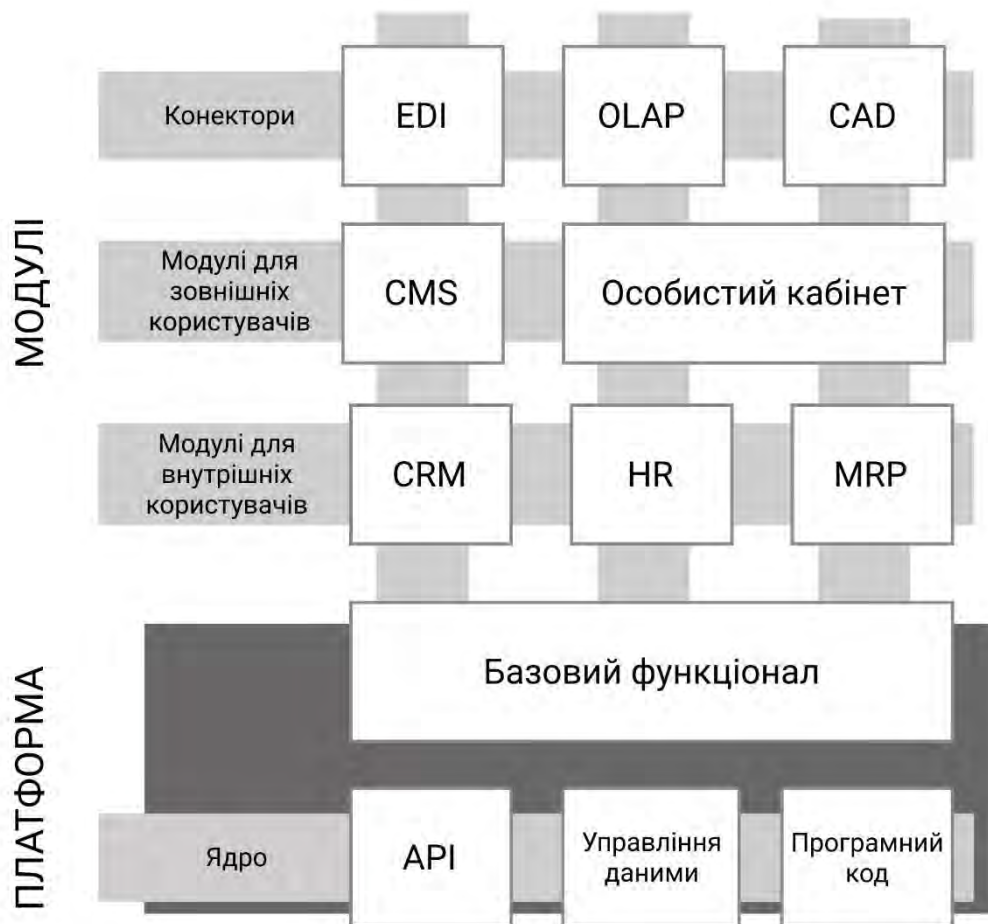


Рисунок 1.1 – Загальна архітектура ERP-системи

2) модулі роботи з зовнішніми користувачами. Цей шар містить в собі модулі, необхідні для взаємодії із зовнішніми користувачами, потенційними і реальними клієнтами компанії, партнерами, користувачами продукції, постачальниками і покупцями. Це може бути інтернет-магазин, особисті кабінети для постачальників і покупців на корпоративному сайті і тому подібні рішення. Деякі ERP-системи містять в собі готові CMS-системи для створення інтернет-магазину або корпоративного сайту з нуля, інші пропонують тільки окремі інструменти «надбудови» до сайту і / або клієнтські програми (для мобільних і планшетів);

3) коннектори – готові рішення для зв'язку зі сторонніми додатками. Найчастіше використовують API з ядра платформи. Дозволяють інтегрувати телефонію, налаштувати обмін даними з сайтом

або будь-якими програмними продуктами і системами. Коннектори призначені тільки для обміну даними і зазвичай використовуються для обміну даними з EDI, CMS, CAD, BI, OLAP тощо. Тобто з тими системами які не входять до ERP, але використовуються в компанії [9], [10].

1.1.4 Класифікація ERP-систем

Система ERP є програмою зі складною архітектурою. Впровадження такого продукту спрощує обмін інформацією між підрозділами компанії і підвищує ефективність роботи підприємства. Причому використовувати систему можна практично в будь-якій галузі бізнесу. Так, BAS ERP, популярна ERP-система в Україні, застосовується для управління аграрними і будівельними компаніями, автосервісами, мережевими магазинами.

Існуючі системи автоматизації розрізняються за кількома параметрами:

- призначення;
- тип розміщення;
- архітектура;
- функціонал;
- тип організації.

Види за призначенням – це найбільш поширений підхід до класифікації програм для автоматизації бізнесу. ERP-системи поділяються на:

- комплексні;
- галузеві;
- для малого бізнесу.

Комплексні системи досить універсальні. Подібні продукти частіше використовуються великими підприємствами і фінансовими організаціями. У таких програмах передбачені різні інструменти і модулі для автоматизації поточних бізнес-процесів. Опрацьовані механізми інтеграції з іншими системами, програмами.

Галузеві ERP-системи орієнтовані на компанії, що працюють в одній конкретній сфері економіки, а також вузькоспеціалізовані підприємства. Наприклад, на компанії, що займаються продажем авіаквитків.

Формально IT-системи для малого бізнесу не можна віднести до класу ERP через мінімального набору функцій. Однак такі програмні продукти повністю покривають потреби компанії. Найчастіше вони використовуються для оптимізації роботи СТО, консалтингових компаній та інших малих підприємств.

Існує ще одна класифікація ERP-систем – по типу розміщення. У цьому випадку бізнес-програми поділяються на десктопні і браузерні. Впровадження перших може викликати певні складнощі. Через різноманіття «заліза», пристроїв, операційних систем продукт може працювати некоректно.

Браузерні системи менш вибагливі, при цьому не поступаються десктопних продуктах по функціональності. Вони доступні з Opera, Chrome або іншого браузера. Отже, використовувати ERP можна практично з будь-якого пристрою. Наприклад, з ПК або планшета.

При підборі системи для оптимізації бізнесу дуже важливу роль відіграє архітектура програмного продукту. Від неї залежать терміни впровадження ERP-системи на підприємстві. За цим параметром можна виділити всього два типи програмних продуктів:

- з єдиною архітектурою;
- модульні.

Бізнес-програми з єдиною архітектурою відрізняються простотою установки і надійністю. Вони являють собою готовий неподільний інструмент, компоненти якого пов'язані між собою. Така архітектура служить своєрідним захистом від помилок і збоїв системи. Але подальша адаптація або розширення програмного продукту може викликати певні складнощі.

Модулі ERP-системи дозволяють оптимізувати різні аспекти роботи компанії. Наприклад, управління персоналом, розподіл ресурсів або нарахування заробітної плати.

Завдяки розвитку технологій, ERP зазнали значних змін. Тепер подібний програмний продукт може використовуватися не тільки з локального комп'ютера одного з користувачів. Все частіше для організації функціональної системи застосовуються хмарні сервіси або орендовані обчислювальні потужності. Такий підхід до розміщення програмного продукту дозволяє знизити витрати компанії.

З огляду на особливості організації, доступні в Україні ERP-системи можна розділити на:

- приватні;
- публічні;
- гібридні.

Приватний ERP повністю ізольовані, в разі потреби можуть доопрацьовуватися. Для їх установки використовуються орендовані сервера і обчислювальні потужності. Подібна турбота про безпеку даних компанії обходиться досить дорого.

На порядок дешевше можна придбати публічну ERP-систему. Вона являє собою хмарний сервіс, доступ до якого відкритий обмеженому колу користувачів.

Гібридні системи ERP – своєрідний симбіоз двох попередніх типів. Такі продукти відрізняються адаптивністю і орієнтовані на швидке розширення.

Існує також класифікація програмних продуктів по типу ліцензії. Виходячи з цього параметра, виділяють наступні види ERP-систем:

- пропрієтарні;
- open-source.

Перші ERP є закритими. Це готові рішення для автоматизації управління бізнесом з набором типових модулів. Підприємці оплачують повну вартість такого програмного продукту, навіть якщо функції ERP-систем частково не використовуються.

У другому випадку замовник отримує програмний продукт з відкритим вихідним кодом. Користуватися ним можна безкоштовно, неодноразово допрацьовує систему під власні потреби. Замовником оплачуються лише

послуги з адаптації програмного продукту. Така ліцензія обходиться на порядок дешевше, особливо якщо в штаті компанії є один або кілька програмістів [10].

1.2 Аналіз та порівняння методів представлення знань

Людина одночасно використовує найрізноманітніші методи представлення знань: мовний опис, графічна інформація, математичні формули тощо. Залежно від специфіки тієї чи іншої області діяльності один або кілька видів опису будуть превалювати над іншими. Наприклад, в математиці формули і графіки будуть переважати над текстовою інформацією, в юриспруденції домінує текстова інформація, в мистецтвознавстві переважає текстова та графічна інформація.

У разі вибору засобів представлення знань в технічних системах такий універсалізм представлення знань неможливий, оскільки зажадає апаратної і програмної реалізації інтелектуальних функцій, недосяжних в даний час.

Методи представлення знань – методи формалізації та структуризації знань для їх подальшого опрацювання в системах штучного інтелекту. Знання якими володіє фахівець в будь-якій області можна розділити на формалізовані та неформалізовані. Формалізовані знання формулюються в книгах, посібниках, документах у вигляді загальних та строгих суджень (законів, формул, моделей, алгоритмів тощо). Неформалізовані знання зазвичай не потрапляють в книги і керівництва в зв'язку з їх конкретністю, суб'єктивністю і приблизністю.

До класичних методів представлення знань відносять: предикати першого порядку, продукційні правила, семантичні мережі, фрейми, нейронні мережі. Кожен з розглянутих методів представлення знань має свої переваги та недоліки. Використання того чи іншого з них у реальних системах обумовлене специфікою предметної галузі, а також наявністю відповідних технічних і програмних засобів [11].

1.2.1 Предикати першого порядку

Логіка предикатів є розвитком алгебри логіки (або логіки висловлювань). У логіці висловлювань для позначення фактів використовуються літери (імена або ідентифікатори), не мають структури, і які приймають значення «1» або «0» («так» або «ні»). У логіці предикатів факти позначаються n -арними логічними функціями – предикатами $F(x_1, x_2, \dots, x_m)$, де F – ім'я предиката та x_i – аргументи предиката. Імена предикатів неподільні, тобто є так званими атомами. Аргументи можуть бути атомами або функціями.

При записі формул (виразів) крім логічних зв'язок "кон'юнкція" (&), "диз'юнкція" ($\vee$), "заперечення" ($\neg$), "проходження" ("імплікація") ($\rightarrow$), запозичених з логіки висловлювань, в логіці предикатів використовуються квантори загальності ($\forall$) та існування ($\exists$).

До недоліків логіки предикатів 1-го порядку як методу представлення знань можна віднести наступне:

- монотонність логічного висновку, тобто неможливість перегляду отриманих проміжних результатів (вони вважаються фактами, а не гіпотезами);
- неможливість застосування як параметрів предикатів інших предикатів, тобто неможливість формулювання знань про знання;
- детермінованість логічного висновку, тобто відсутність можливості оперування з нечіткими знаннями.

Але логіку предикатів 1-го порядку можна використовувати як основу для конструювання більш складних і зручних логічних методів представлення знань.

Логіка предикатів 1-го порядку лягла в основу мов логічного програмування, найпоширенішим з яких є Prolog (різні його діалекти) [12].

1.2.2 Продукційні правила

У продукційній моделі, або моделі, заснованій на правилах, знання зображуються у вигляді сукупності фактів та правил.

Продукційні правила описують процедурні знання і подають їх у наступному вигляді: ЯКЩО (умова), ТО (дія).

Механізм, реалізований як засіб виводу в продукційних системах, називається машиною логічного висновку і виконує функції пошуку в базі правил, послідовного виконання операції над знаннями та отримує висновки. Існує два способи проведення таких висновків – прямі висновки і зворотні висновки. Прямим висновкам (прямому ланцюжку міркувань) відповідає шлях від посилок до слідств. Зворотнім виводам (зворотному ланцюжку міркувань) відповідає шлях від мети (факту, який потрібно встановити) до передумов. Під умовою розуміють пропозицію-зразок, за якою виконується пошук у базі фактів. Умову іноді називають посилкою правила, а дії, що виконуються в разі успішного завершення пошуку за умовою, – дією правила. Дію також називають висновком правила. Дія може бути проміжною або термінальною (цільовою). Проміжні дії – це окремі кроки в ланцюжку можливих (або необхідних) дій. Такою дією може бути, наприклад, ініціювання діалогу з людиною, внесення змін у базу фактів або породження додаткових умов. Термінальні дії завершують роботу системи [13].

Сукупність правил утворює базу правил. База знань у продукційній моделі – це сукупність бази фактів і бази правил.

Перевагами продукційного методу представлення знань є наступні:

- наочність і зрозумілість знань (принаймні, на рівні одного правила);
- можливість реалізації немонотонного логічного висновку та обробки суперечливих фактів;
- можливість введення різних модифікацій в інтерпретацію правил відповідно до особливостей розв'язуваних системою завдань;
- можливість легкого нарощування бази знань шляхом додавання нових правил.

Недоліками цього методу подання є наступні.

- неосяжність великої бази знань і її структури;

- можливість легкого внесення серйозних помилок в базу знань, що призводять до неправильного функціонування системи (якщо в системі немає розвинених засобів перевірки цілісності бази знань);
- орієнтація на послідовну обробку правил [14].

1.2.3 Семантичні мережі

Знаннями можна називати описи зв'язків між абстрактними поняттями і сутностями, що є конкретними об'єктами реального світу. Поняття і зв'язки між ними можна описати мережею, що складається з вузлів і дуг. Вузли в такій мережі зображують сутності й поняття, а дуги відповідають зв'язкам між ними; усі вузли й дуги можуть бути позначені мітками, які вказують, що саме вони описують. Така форма зображення знань називається семантичною мережею.

Перевагою семантичних мереж є їх універсальність, що досягається за рахунок вибору відповідного застосування набору відносин. За допомогою семантичної мережі можна описати яку завгодно складну ситуацію, факт або предметну область.

Недоліком семантичних мереж є їх практична неозорість при опису моделі світу реального рівня складності. При цьому з'являється проблема розміщення семантичної мережі в пам'яті ЕОМ. Якщо її розміщувати в оперативній (віртуальній) пам'яті, на її складність накладаються жорсткі обмеження. Якщо розміщувати у зовнішній пам'яті, з'являється проблема, як довантажувати необхідні для роботи ділянки [15].

1.2.4 Фрейми

Фреймова модель запропонована М.Мінським і є систематизованою психологічною моделлю пам'яті людини і його свідомості.

Фрейм – структура даних для представлення деякого концептуального об'єкта. Це мінімальна структура інформації, необхідна для представлення класу

об'єктів, явищ або процесів. Інформація, що відноситься до фрейму, міститься в складаючих його слотах. Слот – це деякі незаповнені підструктури фрейма, заповнення яких приводить до того, що цей фрейм ставиться у відповідність деякій ситуації, події або об'єкту. Слот може бути термінальним (містити тільки ім'я слота і значення імені) або являти собою фрейм нижнього рівня.

У вигляді фрейму може описуватися деякий об'єкт, ситуація, абстрактної поняття, формула, закон, правило, візуальна сцена тощо.

Особливістю і основним недоліком фреймового представлення знань є відсутність механізму висновку, тобто відповідальність за правильну обробку знань повністю покладається на розробника системи, який повинен представляти предметну область і не тільки детально описати кожну фрейм і слот, але і правильно задати процедури їх обробки. До основних процедур управління висновком у фреймовій моделі звичайно відносять: механізм успадкування, обмін повідомленнями між фреймами, приєднані процедури [14]-[16].

1.2.5 Нейронні мережі

Штучна нейронна мережа – паралельно розподілений процесор, який володіє здатністю до навчання, збереження і представлення знань, набутих на основі досвіду. Якщо розглядати нейронну мережу як спосіб представлення знань, то в ній зберігаються знання про асоціативні зв'язки між стимулами (вхідними векторами) та відгуками (вихідними векторами). Знання зберігаються (формується в процесі навчання) зазвичай у формі ваг зв'язків між нейронами.

В основі нейронних мереж лежить елементарний перетворювач – штучний нейрон, названий так за аналогією з його біологічним прототипом.

Перед використанням нейромережі проводиться її навчання, що є ітераційним процесом налаштування вагових коефіцієнтів. Для навчання використовуються спеціальні алгоритми. Найбільше розповсюдження отримали градієнтні методи – алгоритм зворотного поширення похибки (Back Propagation), зв'язаних градієнтів, RProp тощо. Основна особливість нейронних мереж полягає

в тому, що в процесі навчання вони моделюють складну нелінійну залежність між вхідними і вихідними даними.

Недоліками нейронних мереж як методу представлення знань є:

- труднощі вербалізації результатів роботи нейронної мережі та пояснень, чому вона прийняла те чи інше рішення;
- неможливість гарантувати повторюваність та однозначність отримання результатів.

Переваги нейронних мереж як методу подання знань:

- відсутність необхідності формалізації знань, формалізація замінюється навчанням на прикладах;
- природне уявлення та обробка нечітких знань приблизно так, як це здійснюється в природному інтелектуальній системі – мозку;
- орієнтація на паралельну обробку, що при відповідній апаратній підтримці забезпечує можливість роботи в реальному часі;
- відмовостійкість і живучість при апаратній реалізації нейронної мережі;
- можливість обробки багатовимірних (розмірності більше трьох) даних та знань так само (без збільшення трудомісткості) як і невеликої розмірності (але в цьому випадку ускладнено пояснення результатів, тому що людина насилу сприймає багатовимірність) [14].

1.2.6 Порівняльна характеристика моделей представлення знань

Моделі представлення знань утворюють дві групи – евристичні та логічні моделі. Евристичні підходи до подання знань мають переважно характер мистецтва, в їх використанні превалює інтуїція, досвід і майстерність розробника. До числа евристичних моделей відносяться фреймових моделі подання знань і семантичні мережі.

У основі логічних моделей лежить суворе поняття формальної системи, тобто образотворчі засоби, що лежать в основі цих моделей, мають теоретичне

обґрунтування. Склад логічних моделей представлення знань утворюють логічні і продукційні моделі.

Для порівняння моделей подання знань використовують такі критерії оцінювання моделей подання знань:

- формалізованість. Найнижчий рівень формалізованості мають семантичні мережі, більш високий рівень у продукційних моделях, ще більший рівень формалізованості є у фреймових моделях, найвищий рівень формалізованості має логіка предикатів 1-го порядку;
- можливість поповнення. Найнижчий рівень має логіка предикатів, більш високі рівні займають, відповідно, семантичні мережі, фрейми, продукційні моделі;
- нотаційна адекватність в основному виражається синтаксичною будовою моделей подання знань та має особливе значення для систем, що використовують експертні знання, проектування та підтримка котрих багато у чому залежить від людей, що не є спеціалістами у формалізації знань. Найнижчий рівень нотаційної адекватності є у логіки предикатів, інші моделі мають більш високий відносно нього, але приблизно однаковий у порівнянні між собою;
- відображення семантики предметної області. Найнижчий рівень відображення у логіки предикатів, продукційні моделі та семантичні мережі мають більш високий рівень ніж у логіки предикатів. Найвищий рівень відображення мають фрейми;
- інференційні можливості. У порядку зростання рівня цього критерію слідує: семантичні мережі, фрейми, продукції та логіка предикатів [15].

1.3 Висновки за розділом 1

У даному розділі було розглянуто основне поняття ERP-системи та їх класифікації, а також проаналізовано їх структуру та класифікації.

Також у розділі були розглянуті основні методи представлення знань, з яких для представлення ERP-системи було обрано семантична мережу, яка

підходить для опису об'єктів динамічної природи, оскільки в ній може бути забезпечене включення нових понять і зв'язків, вона може навчатися та зростати.

2 МАТЕРІАЛИ ТА МЕТОДИ

2.1 Формалізація знань про предметну область «Робочий процес ІТ підприємства»

Семантична мережа – мережна модель, що має вигляд орієнтованого графа, вершинами (вузлами) якої є класи (об'єкти предметної), а дугами (ребрами) – спрямованість відносини або зв'язку, що з'єднують ці вузли. У семантичній мережі відображені типи відносин [17].

Як модель предметної області, семантична мережа найбільш часто використовується для представлення декларативних знань, за допомогою якої реалізуються такі властивості системи знань, як інтерпретованість і зв'язність. За рахунок цих властивостей семантична мережа дозволяє знизити обсяг збережених даних, забезпечує висновок умовиводів по асоціативним зв'язкам. Семантична мережа є необхідним ресурсом і способом представлення знань, який відображає семантику предметної області, а також дозволяє проводити візуальний аналіз предметної області, графічно відображаючи її структуру.

Семантичні мережі можуть бути записані практично на будь-якій мові програмування на будь-якій машині. Найпопулярніші в цьому відношенні мови LISP і PROLOG. Однак багато версії були створені і на FORTRANi, PASCALi, C та іншими мовами програмування. Для зберігання всіх вузлів і дуг необхідна велика пам'ять, хоча перші системи були реалізовані в 60-х роках на машинах, які були набагато меншими і повільнішими від сучасних комп'ютерів.

Крім спеціальних мов для семантичних мереж було також розроблено спеціальне апаратне забезпечення. На звичайних комп'ютерах можуть бути успішно виконані операції з мовами синтаксичного аналізу й операції сканування мереж. Однак для великих баз знань знаходження потрібних правил або доступ до передзнання може знадобитися дуже багато часу. Щоб дозволити різним процесам пошуках проходити одночасно Фальман розробив систему NETL, яка представляє собою семантичну мережу, що може використовуватися з паралельним апаратним забезпеченням. Таким чином він хотів створити

модель людського мозку, в якому сигнали можуть рухатися по різних каналах одночасно. Інші вчені розробили паралельне програмне забезпечення для пошуку найбільш вірогідної інтерпретації двозначних фраз природної мови [18].

Як висвітлено у першому розділі, ERP-система має виконувати обов'язки єдиної системи для забезпечення автоматизації більшості операцій та процесів робочого процесу підприємства. Проте без конкретного представлення про предметну область робочого процесу ІТ підприємства, подальше моделювання та розробка можуть не в повній мірі описати архітектуру та бізнес-логіку майбутнього програмного комплексу.

Для цих цілей, було вирішено формалізувати знання з використанням семантичної мережі.

За допомогою семантичної мережі були формалізовані знання про предметну область «Робочий процес підприємства».

На рисунку 2.1 показано взаємозв'язок сутностей та акторів зі системою. Акторами у даному випадку виступають адміністратор, менеджер та розробник.



Рисунок 2.1 – Відношення ERP-системи з зовнішніми акторами

Як видно з рисунку, актори є частиною сутності «Роль», що представляє базове розподілення обов'язків в робочому процесі. Також було проілюстровано відношення ключових сутностей відносно системи.

Прийнявши до уваги відношення сутностей та акторів у цій мережі, можна також формалізувати знання про предметну область процесів усереднені системи спроектувавши семантичну мережу (див. рис. 2.2).



Рисунок 2.2 – Формалізація предметної області ERP-системи

Як видно з рисунку, актором у даній мережі виступає «Користувач». Дана мережа у більшій мірі представляє розподіл обов'язків між ролями, що допомагає спланувати майбутні моделі поведінки та бізнес-логіку ERP-системи.

За допомогою обох мереж формується загальне представлення про предметну область та її можливості на концептуальному рівні, що пришвидшить подальшу розробку архітектури програми та моделювання окремих бізнес-процесів.

2.2 Вибір мов програмування, технологій та середовища розробки

Вибір мови програмування та супутніх технологій (бібліотек та фреймворків) є значним, так як сприятиме швидкості розробки проєкту та впливатиме на його довгострокову підтримку. Далі було обрано мову програмування для вебдодатку програмного комплексу ERP. На таблиці 2.1 наведено декілька критеріїв порівняння мов програмування, які можуть використовуватися для створення веб (браузерних) додатків.

Як видно з таблиці, JavaScript лідує по всім представленим критеріям бо є невід’ємною складовою більшості вебдодатків. Навіть якщо додаток був написаний за допомогою, наприклад, PHP, то за динаміку та програмні анімації все одно відповідатиме JavaScript.

Таблиця 2.1 – Порівняльна характеристика мов програмування для розробки вебдодатку

Критерій порівняння	Мова програмування				
	JS	Java	PHP	C#	Python
Швидкість виконання коду	++	+	+	+	++
Масштабованість	++	±	++	±	+
Популярність	++	+	++	±	++
Універсальність	++	++	++	+	++
Простота розроблення вебінтерфейсу	++	±	+	±	++

JavaScript (JS) – динамічна, об’єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв вебсторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд вебсторінки [19].

JavaScript – це нестрога типізована мова програмування, тому це дещо пришвидшує розробку коду, проте розробник має бути знайомий з можливими проблемами, які виникають унаслідок даної типізації аби уникнути їх на етапі розробки.

Тепер, необхідно обрати мову програмування для серверного додатку. Навіть зважаючи на те, що мовою JS можна розробляти серверні додатки за допомогою платформи Node.js, все ж необхідно порівняти можливі популярні варіанти за спеціальними критеріями, які наведені в таблиці 2.2.

Таблиця 2.2 – Порівняльна характеристика мов програмування для розробки серверного додатку

Критерій порівняння	Мова програмування				
	JS (Node.js)	Java	PHP	C#	Python
Швидкість виконання коду	+	++	+	++	+
Масштабованість	++	+	++	+	+
Популярність	++	+	++	+	++
Універсальність	++	++	++	+	++
Простота розроблення API додатків	++	+	++	+	++

Як видно з таблиці, Node.js та PHP лідують за даними критеріями, проте було обрано PHP за ще одним значним критерієм – наявність повноцінних MVC (англ. Model View Controller – модель-уявлення контроллер) фреймворків, які значно пришвидшать розробку додатка, та які надають необхідний мінімум коду та підходів для успішного побудування додатка [20].

Для вебдодатку було порівняно декілька найпопулярніших фреймворків за критеріями, які представлені у таблиці 2.3.

Таблиця 2.3 – Порівняльна характеристика найпопулярніших фреймворків мови JS

Критерій порівняння	JavaScript фреймворк				
	Vue	React	Ember	Svelte	Angular 2
Паралельне програмування	++	+	+	++	++
Масштабованість	++	++	++	+	+
Популярність	+	+++	+	+	++
Швидкість розробки кодової бази	+	++	+	++	+
Простота вивчення	++	++	+	+	+
Підтримуваність браузером	+	+	++	+	++

Як видно з таблиці, React є найпопулярнішим фреймворком, який легко освоїти та використовувати.

React (іноді React.js або ReactJS) – JavaScript-бібліотека з відкритим вихідним кодом для розробки призначених для користувача інтерфейсів. React розробляється і підтримується Facebook, Instagram і співтовариством окремих розробників і корпорацій. React може використовуватися для розробки односторінкових і мобільних додатків. Його мета – надати високу швидкість, простоту і масштабованість [21].

Проте в даній програмній реалізації було обрано його SSR (Server Side Rendering) підвид Next.js – відкрита бібліотека зі своїм набором функціональності, побудована базуючись на React.

Щодо PHP фреймворку, наразі існує декілька лідерів, які були порівняні певними критеріями, наведеними у таблиці 2.4.

Таблиця 2.4 – Порівняльна характеристика популярних PHP фреймворків

Критерій порівняння	PHP фреймворк				
	Laravel	CodeIgniter	Synfony	CakePHP	Yii 2
1	2	3	4	5	6
Підтримка патернів	++	+	++	+	+
Масштабованість	++	+	++	+	+

Продовження таблиці 2.4

1	2	3	4	5	6
Популярність	+++	+	++	+	++
Швидкість розробки кодової бази	++	++	++	+	+
Простота вивчення	++	++	+	+	+
Підтримка різних баз даних	+++	++	++	++	++

Як видно з таблиці, Laravel є ідеальним вибором для побудування додатків. Laravel – безкоштовний вебфреймворк з відкритим кодом, призначений для розробки з використанням архітектурної моделі MVC [22].

Laravel надає фундамент, на якому можна легко та швидко побудувати веб, API додатки, тощо. Також Laravel, за допомогою програмних бібліотек-драйверів та Eloquent ORM задає єдиний стандарт роботи з базами даних, проте водночас підтримує як реляційні, так і нереляційні їх версії.

Щоб обрати необхідне середовище розробки, треба порівняти декілька популярних варіантів за критеріями, які наведені у таблиці 2.5.

Таблиця 2.5 – Порівняльна характеристика середовищ розробки

Критерій порівняння	Середовище розробки			
	VS Code	IntelliJ IDEA	NetBeans	Eclipse
Регулярність оновлень та модернізацій	++	++	±	±
Інтеграція системи контролю версій	++	++	+	+
Популярність	++	++	±	±
Модульність	++	++	-	-
Доступність	++	-	++	++
Багатоплатформність	++	++	++	+
Наявність додаткового інструментарію	+	++	+	+

Як видно з таблиці, VS Code являється актуальним, доступним та потужним інтегрованим середовищем для розробки, проте наявність величезної кількості вбудованих інструментів у IntelliJ IDEA переважає вибір на свою сторону. IntelliJ IDEA має декілька варіацій, створених для різних цілей та мов. В даній реалізації буде використано PhpStorm IDE. JetBrains PhpStorm – комерційне крос-платформове інтегроване середовище розробки для PHP, яке розробляється компанією JetBrains на основі платформи IntelliJ IDEA.

PhpStorm являє собою інтелектуальний редактор для PHP, HTML і JavaScript з можливостями аналізу коду на льоту, запобігання помилок у сирцевому коді і автоматизованими засобами рефакторинга для PHP і JavaScript. Автодоповнення коду в PhpStorm підтримує специфікацію PHP 5.3/5.4/5.5/5.6/7.0/7.1/7.2/7.3/7.4/8.0 (сучасні і традиційні проєкти), включаючи генератори, співпрограми, простори імен, замикання, типажі і синтаксис коротких масивів. Присутній повноцінний SQL-редактор з можливістю редагування отриманих результатів запитів.

PhpStorm розроблений на основі платформи IntelliJ IDEA, написаної на Java. Користувачі можуть розширити функціональність середовища розробки за рахунок установки плагінів, розроблених для платформи IntelliJ, або написавши власні плагіни [23]. PhpStorm підтримує більшість PHP та JS фреймворків, що надає дещо поліпшений аналіз коду та автодоповнення коду.

2.3 Висновки за розділом 2

У даному розділі було розкрито питання про використання семантичних мереж для формалізації знань про предметну область, а також формалізовано знання про предметну область «Робочий процес ІТ підприємства».

Було проаналізовано та порівняно популярні мови програмування для створення веб- та серверних додатків, у результаті якого було обрано JavaScript для веб додатку та PHP для серверного. Також було проаналізовано та порівняно фреймворки для визначених мов, а саме React (Next.js) для веб додатку та Laravel

для серверного. Зважаючи на вибір мов та фреймворків, було також проаналізовано найпопулярніші IDE та обрано середовище розробки PhpStorm (IntelliJ IDEA).

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Опис основних вимог до системи

Головна вимога до розроблюваного програмного комплексу: зменшити до мінімуму затрачуваний час на балансування ресурсів компанії та аудит бюджету. Опираючись на цю вимогу, було встановлено наступні критерії:

- наявність функціональності калькуляції робочого часу для виконавчого персоналу;
- наявність функціональності підрахування доходів та витрат;
- наявність функціональності генерування рахунків-фактур для платежу.

Щоб програмна реалізація системи була повноцінною та готовою до використання в реальних умовах, а також для її вдосконалення, було встановлено додаткові критерії до функціональності програмного комплексу:

- наявність функціональності авторизації;
- наявність функціональності запрошень до системи через e-mail;
- наявність RBAC функціональності;
- невимогливість до серверного обладнання.

3.2 Проєктування структури ERP-системи

При проєктуванні структури ERP-системи було вирішено розділити її на два додатки або модулі: графічний вебінтерфейс та API додаток. Аргументовано даний вибір тим, що система буде поєднувати у собі забагато функціональності, яка в свою чергу навантажуватиме кодову базу та робитиме її складнішою для подальшої підтримки та вдосконалення. Обидва додатки утворюють програмний комплекс, який в подальшому можливо розширити, розробивши додатки для мобільних пристроїв та персональних комп'ютерів.

У вебдодатку можна виділити наступні функціональні модулі:

- функціональність та графічний інтерфейс авторизації;

- роздільний графічний інтерфейс, закріплений за роллю та доступами;
- підтримка сортування, фільтрації та пагінації записів сутностей;
- валідація даних при створенні або редагуванні записів сутностей чи відношень між ними.

Роздільний графічний інтерфейс необхідний для того, щоб захистити від перегляду непотрібної або не дозволеної інформації користувачами нижчих ролей або без особливих доступів. На прикладі ролей адміністратора підприємства та менеджера проєкту, очевидно, що останній матиме доступ лише до проєктів та персоналу, до якого менеджер відноситься. Адміністратор у свою чергу матиме необмежений доступ.

Наявність функціональності сортування, фільтрування та пагінації являється необхідним для швидкого оперування великими об'ємами даних. Фільтрація, наприклад, надає змогу менеджеру швидко знайти у великому списку вільного від завдань розробника, відфільтрувавши список по статусу. Пагінація у свою чергу пришвидшує процес побудування інтерфейсу через невелику кількість записів, яку опрацьовує за один запит на сервер.

Функціональність валідації даних необхідна як превентивна міра, щоб користувач не мав змогу зберегти запис без необхідних полів, неправильним типом даних або некоректних відношень. Також, дана функціональність зменшить кількість запитів на сервер що позитивно відобразиться на оптимізації програмного комплексу в цілому.

До загальної функціональності серверної частини можна віднести наступне:

- модуль, який реалізує підходи REST API та CRUD;
- функціональність авторизації;
- функціональність побудування відповіді на запит зважаючи на доступи авторизованого користувача;
- функціональність перевірки доступу до запрошуваного маршруту;
- модуль запису та зчитування даних до БД;

- функціональність генерування файла рахунка-фактури.

REST (скор. англ. Representational State Transfer, «передача репрезентативного стану») – підхід до архітектури мережеских протоколів, які надають доступ до інформаційних ресурсів. Побудування API додатків на принципах REST являється найбільш популярним в сучасних умовах [24].

Функціональність побудування відповіді зважаючи на доступи та функціональність перевірки доступу до запрошуваного маршруту є частиною зональної функціональності RBAC.

3.3 Моделювання програмного забезпечення ERP-системи

Функціональність передбачено набула значних розмірів, тому задля представлення логіки майбутнього програмного комплексу у достатній мірі, було вирішено змоделювати його за допомогою UML діаграм.

Задля представлення усіх можливих варіантів використання системи, було сконструйовано графічну модель у вигляді діаграми прецедентів, наведеною на рисунку 3.1.

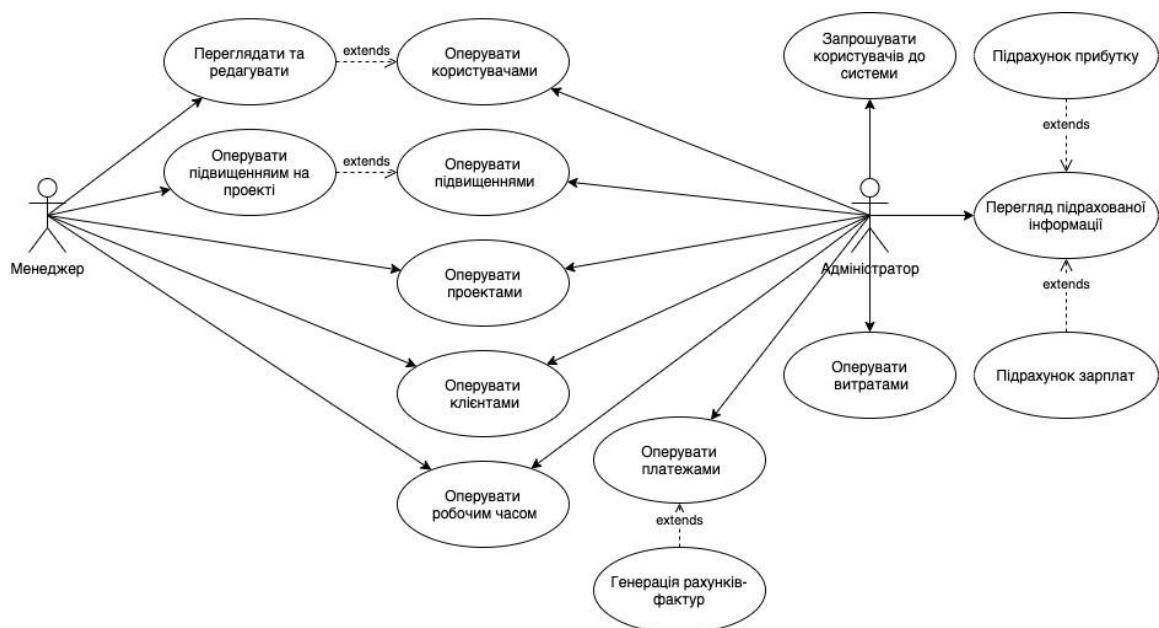


Рисунок 3.1 – Діаграма прецедентів в нотації UML 2.0

На рисунку 3.1 представлено можливі варіанти використання системи акторами: адміністратором та менеджером. Очевидно, в даній моделі роль «Розробник» не приймає участі, тому що його потенціальний варіант використання системи (оперування робочим часом) буде обов'язком зв'язаного з ним менеджера задля спрощення системи. Як показано на рисунку, адміністратор має необмежений доступ в оперуванні усіма сутностями системи, має доступ до фінансової функціональності, а також до підрахованої інформації, до котрих не має доступу менеджер. Також, у менеджера дещо зменшені доступи до сутностей підвищень та користувачів.

За допомогою діаграми класів на рисунку 3.2 представлено взаємозв'язок сутностей у більш детальному вигляді.

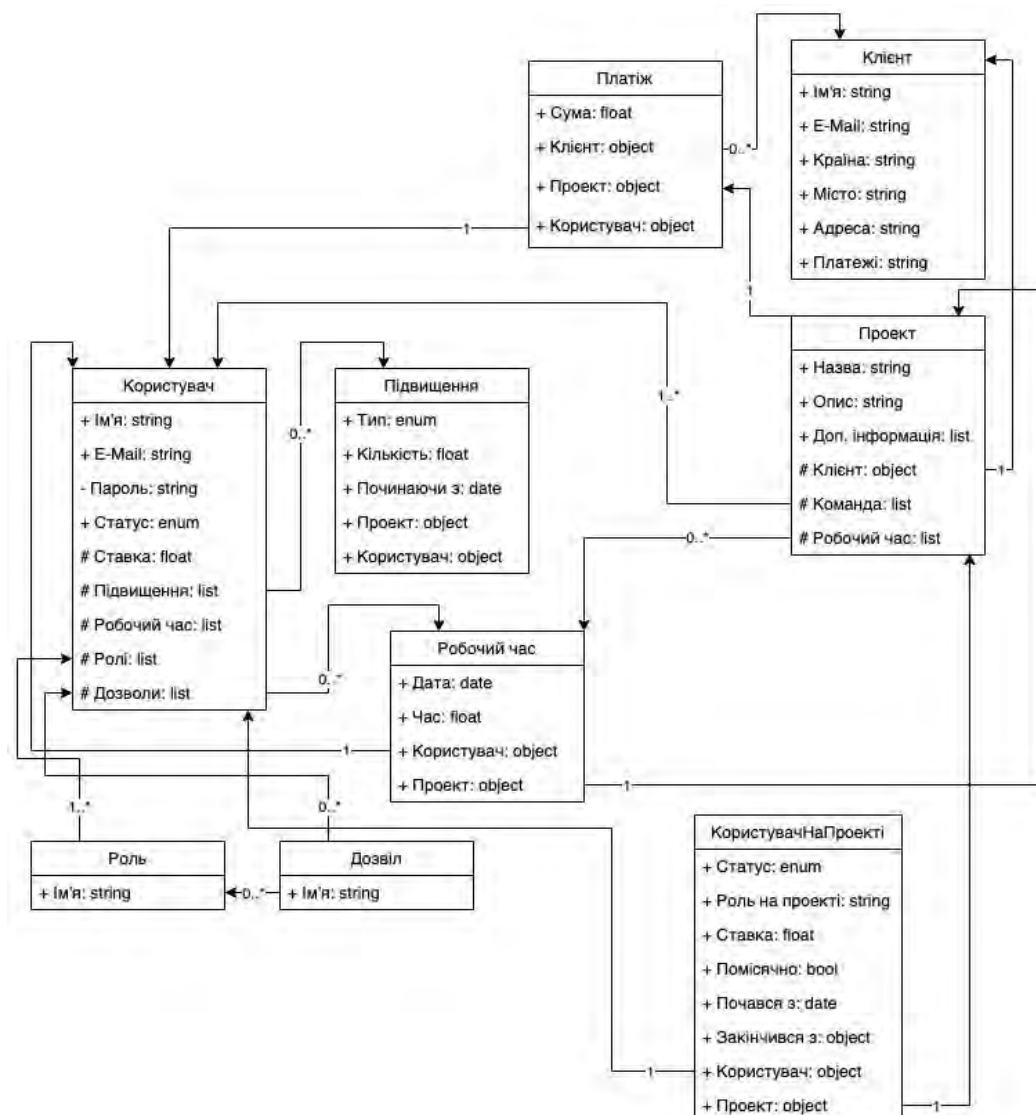


Рисунок 3.2 – Діаграма класів в нотації UML 2.0

За допомогою цієї діаграми можна значно спростити проектування бази даних. Чому за допомогою діаграми класів було представлено взаємозв'язок сутностей аргументовано у розділі вибору технології та мов програмування.

На рисунку 3.3 представлено сценарій додавання запису клієнта до системи за допомогою діаграми послідовності. Модель проілюстрована за умови, що користувач вже авторизований в системі.

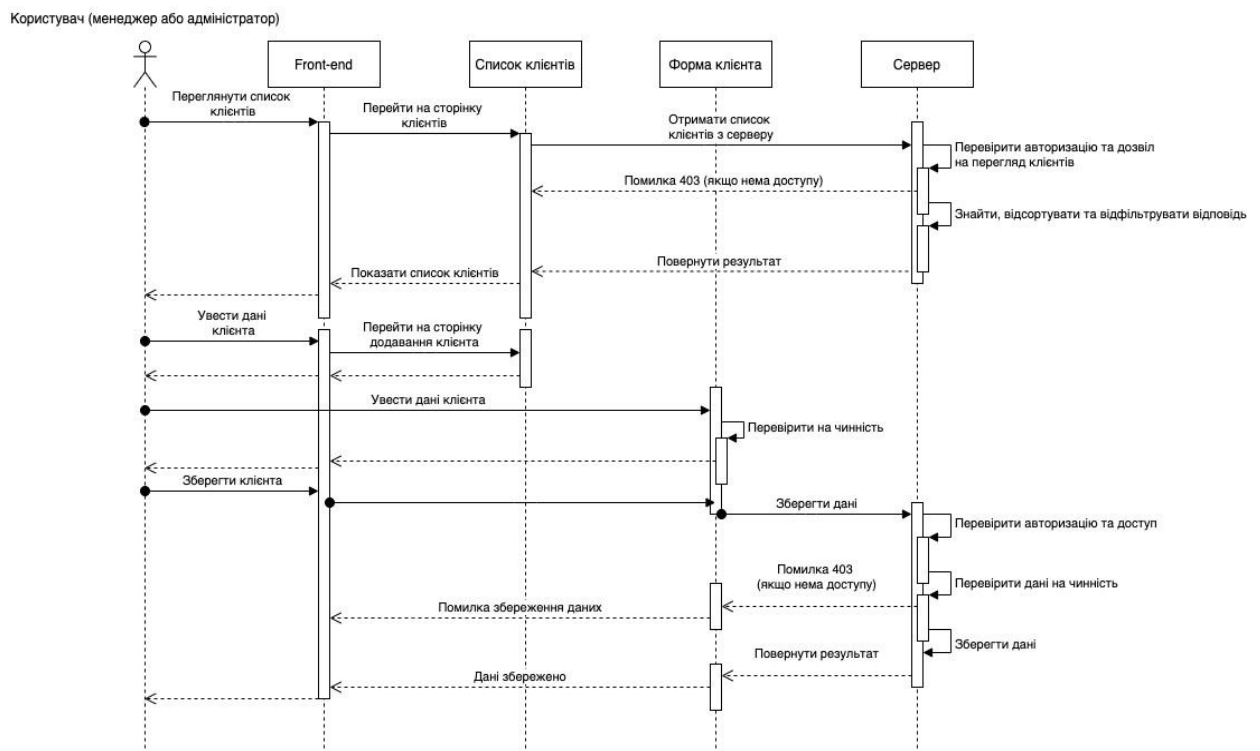


Рисунок 3.3 – Діаграма послідовності в нотації UML 2.0

Як показано на рисунку, користувач має перейти на сторінку клієнтів, де знаходиться посилання на додавання нового запису до системи. Більшість дій, які наведені на рисунку повторюються для інших сценаріїв додавання сутності до системи.

3.4 Проектування бази даних

За допомогою змодельованої діаграми класів, на якій проілюстровано взаємозв'язок сутностей, проектування концептуальної моделі бази даних

значно спрощується. На рисунку 3.4 представлено діаграму зв'язків майбутньої БД.

Важливо зауважити, що дана діаграма не є фінальною версією майбутньої БД, так як на неї впливатимуть вибір технологій у наступному розділі. Як видно на рисунку, центральною сутністю являється користувач (users). Також на діаграмі присутня таблиця «invitations», яка буде використовуватися у функціональності запрошень до системи.

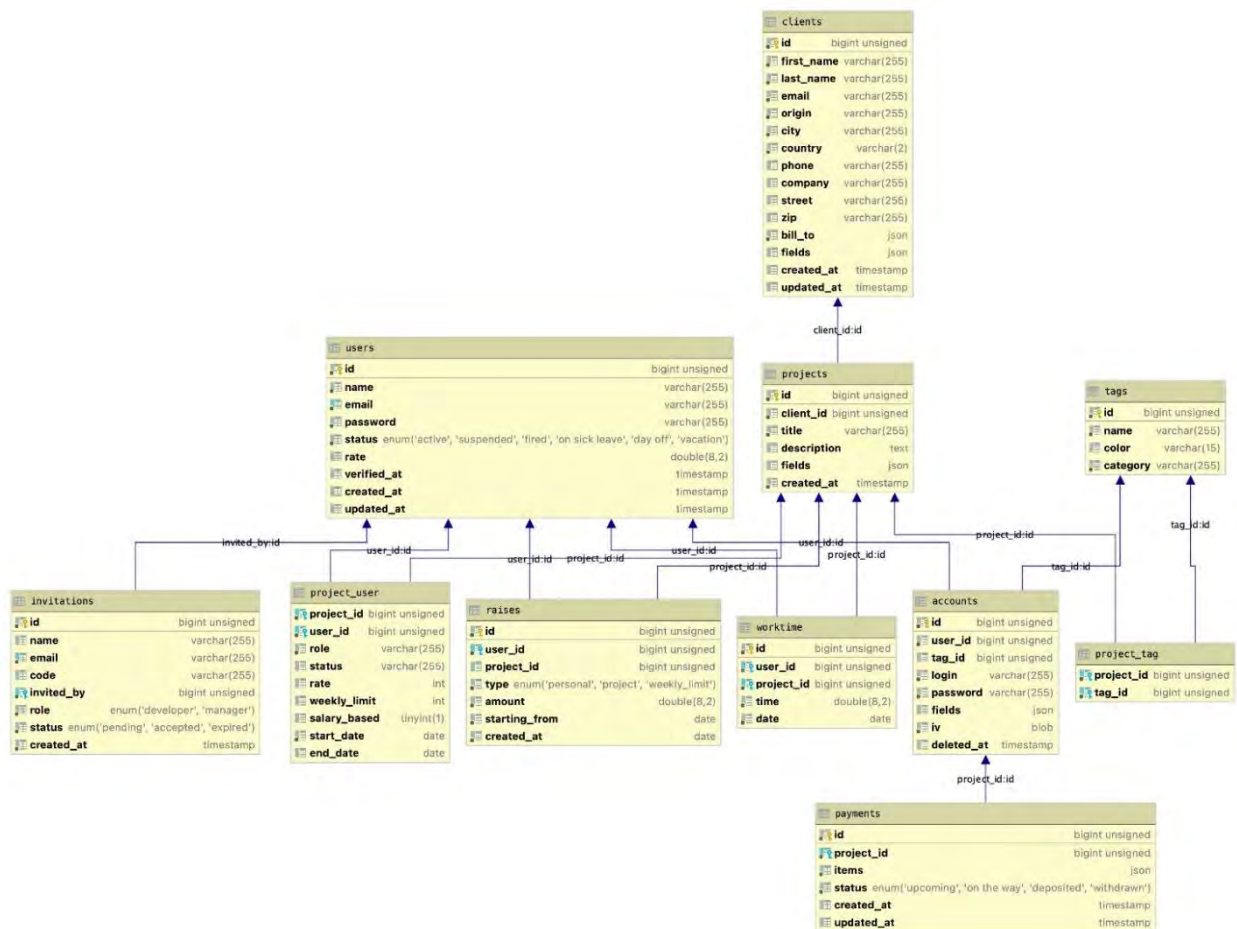


Рисунок 3.4 – Діаграма зав'язків БД

3.5 Вимоги до технічних засобів

Щоб розробляти та запускати програмний комплекс без перешкод, методом створення віртуальної машини за допомогою Virtual Box було

визначено наступні умови до апаратної частини персональної електронної обчислювальної машини:

- будь-яка ОС, яка підтримує визначену IDE, браузер та інтерпретатор PHP;
- процесор із тактовою частотою 2,2 GHz або більше;
- оперативна пам'ять DDR3/DDR4 у розмірі 4 ГБ або більше;
- вільний простір на інформаційному носіїві (бажано SSD) 20 ГБ або більше.

Для розгортання та використання програмного комплексу без розробки було встановлено наступні вимоги:

- Unix-подібна ОС;
- процесор із тактовою частотою 1,8 GHz або більше;
- оперативна пам'ять DDR3/DDR4 у розмірі 1 ГБ або більше;
- вільний простір на інформаційному носіїві (бажано SSD) 10 ГБ або більше;

Значний вільний простір необхідний для додаткового розташування логів програмного комплексу.

3.6 Висновки за розділом 3

Під час роботи над розділом було описано основні вимоги до розроблюваного програмного комплексу, а також спроектовано його архітектуру та змодельовано бізнес-логіку. Було спроектовано концептуальну модель бази даних, а також були визначені основні вимоги до технічних засобів.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ ПРОГРАМНОГО КОМПЛЕКСУ

4.1 Вибір бібліотек для реалізації функціональних модулів

Для подальшої розробки програмного комплексу, для серверного та вебдодатків було обрано додаткові бібліотеки та модулі задля повноцінної та швидкої реалізації функціональності програмного комплексу, зазначеної на етапі проектування та модулювання.

4.1.1 Вибір бібліотек для реалізації функціональних модулів вебдодатку

Для повноцінної реалізації функціональності вебдодатку було обрано наступні бібліотеки:

- React Redux – це менеджер станів. Найчастіше його використовують з React, але його можливості не обмежуються однією цією бібліотекою. Хоча в React є власний метод управління станами, він погано масштабується. Переміщення стану вгору по дереву працює для простих додатків, але в більш складних архітектур зміна стану проводиться через властивості (props). Ще краще робити це через зовнішнє глобальне сховище;

- Axios – це широко відома JavaScript-бібліотека. Вона являє собою HTTP-клієнт, заснований на Promise і призначений для браузерів і для Node.js. У жовтні 2019 року пакет Axios був завантажений з npm 25 мільйонів разів. Найбільшою перевагою являється підтримка статусу завантаження фалів на сервер;

- Ant design – популярна бібліотека UI компонентів, яка найчастіше використовується з React фреймворком. За допомогою Ant design можна побудувати повноцінний UI/UX не інвестуючи при цьому багато часу та ресурсів;

- PDF.js – бібліотека для відображення парсингу та рендерингу PDF фалів у браузері, створена Mozilla. Ця бібліотека знадобиться для відображення згенерованого файлу рахунка-фаткури в майбутній реалізації.

4.1.2 Вибір бібліотек та веб інтерфейсів для реалізації функціональних модулів серверного додатку

Для повноцінної реалізації функціональності серверного було обрано наступні бібліотеки:

- Laravel Permission – неофіційна бібліотека, яка доповнює фреймворк Laravel функціональністю RBAC, яку в свою чергу можна налаштувати та розширити відповідно до вимог проєкту. Присутня базова функціональність ролей та доступів та їх зв'язок з вказаною моделлю даних;

- Laravel Sanctum – офіційна бібліотека, яка запроваджує до фреймворку Laravel необхідний функціонал для авторизації, як за допомогою токену, так і як за допомогою Cookies. Sanctum дозволяє кожному користувачеві додатку генерувати кілька токенів API для свого облікового запису. Цим токенам можуть бути надані доступи / області дії, які визначають, які дії токенам дозволено виконувати.

Для генерування рахунків фактур було обрано відкритий API інтерфейс invoicegenerator.com, за допомогою якого можна згенерувати файл лише відправивши запит на інтерфейс з його вмістом.

4.2 Основні етапи розробки програмного комплексу

Після підготовки середовища розробки, налаштувань репозиторіїв з фреймворками додатків та встановлення бібліотек, було розпочато безпосередню розробку програмного комплексу. У даному розділі наведено основні положення щодо реалізації важливих функцій та функціональних модулів у зв'язку з великим об'ємом виконуваної роботи.

4.2.1 Основні положення та зауваження щодо реалізації серверного додатку програмного комплексу

Зважаючи на змодельовану архітектуру та бізнес-логіку програмного комплексу, було проведено наступну роботу у конфігуруванні, структуризації та розробці функціональності серверного додатку:

- налаштування з'єднання до БД;
- налаштування бібліотек Sanctum та Laravel Permission;
- написання моделей Eloquent та міграцій БД;
- створення необхідних контролерів для сутностей (надалі моделей) за принципами CRUD;
- створення спеціального функціоналу пагінації, фільтрації та сортування;
- створення функціональності калькуляції фінансових даних проєктів та користувачів;
- створення функціональності запрошень до системи.

Далі було розглянуто пункти створення функціональності (починаючи з 3 пункту і далі).

За допомогою концептуальної моделі бази даних та діаграми класів, яка насамперед проєктує відношення моделей (сутностей) у системі. На основі цих даних були створені файли міграції Laravel, які за допомогою команди створюють необхідні таблиці, поля та відношення. Фінальну структуру бази даних наведено на рисунку 4.1.

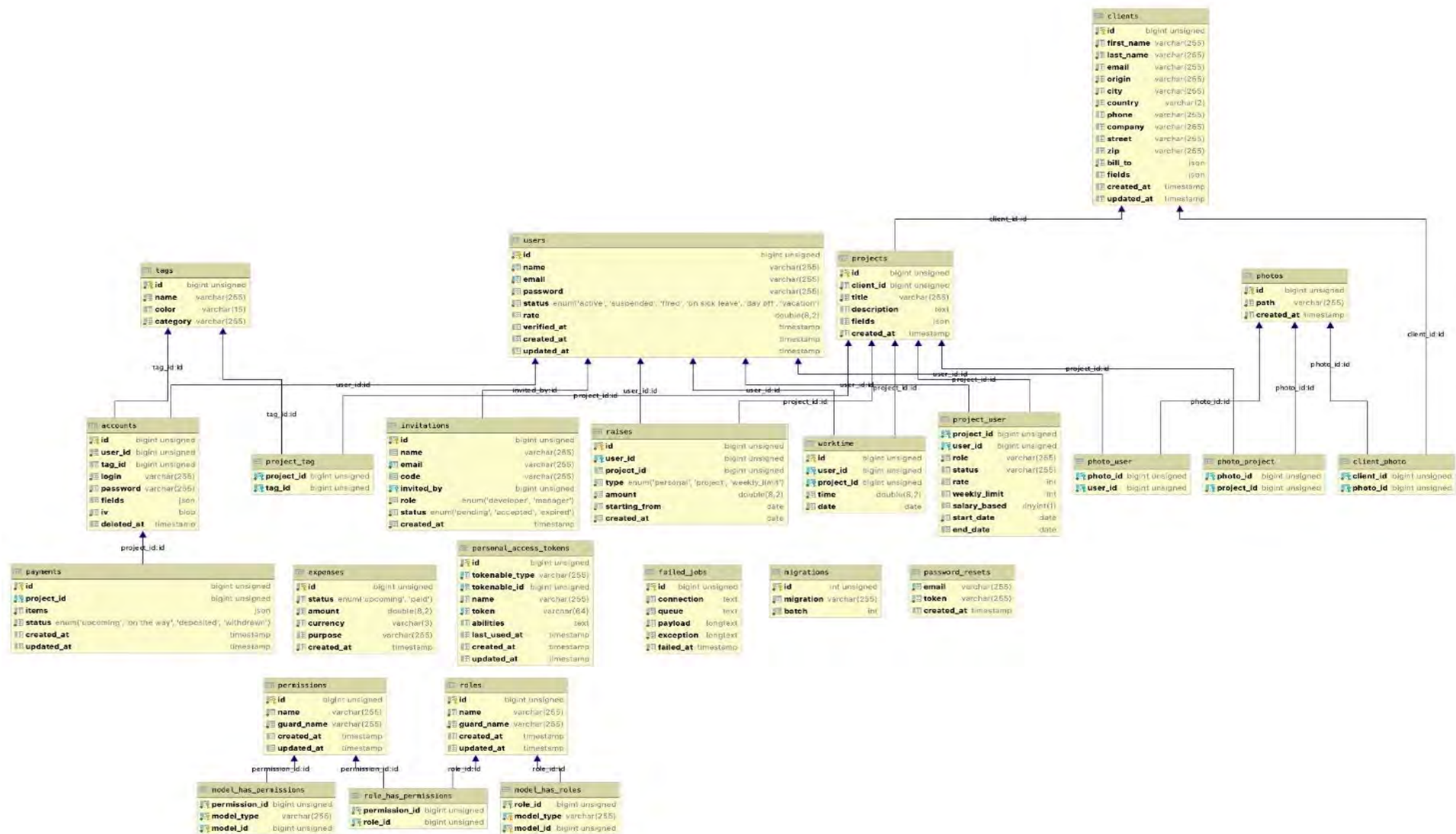


Рисунок 4.1 – Фінальна структура бази даних

Відповідно до побудованої БД, було створено моделі Eloquent, які відповідають основним сутностям системи та описують їх поля та відношення між собою (див. рис. 4.2).

Далі, для кожної сутності було утворено свій клас контролер, а також описані функції-маршрути у файлі «api.php». Утворені класи контролери використовують методи так званого ресурс-контроллера, функціональність якого стандартизує Laravel.

```
<?php

namespace App;

use ...

class Project extends Model
{
    use HasFactory;

    public $timestamps = false;

    protected $fillable = [...];

    public function tags(){...}

    public function users(){...}

    public function accounts(){...}

    public function client(){...}

    public function developers(){...}

    public function managers(){...}

    public function photos(){...}

    public function raises(){...}

    public function worktime(){...}
}
```

Рисунок 4.2 – Клас-модель Eloquent проєкту

За допомогою ресурс контроллера Laravel визначає стандартні маршрути за принципами CRUD, розробнику треба лише заповнити методи класу функціональністю (див. рис 4.3).

Кожен маршрут, який потребує авторизації від користувача, захищений декількома основними механізмами:

- CSRF protection – базовий механізм у Laravel і не тільки, який захищає додаток від атак підробленими запитами;
- CORS – система, яка перевіряє, чи має доступ запит з даним походженням за допомогою спеціального HTTP методу OPTIONS та опціональних налаштувань;
- Sanctum Auth Middleware – клас-посередник бібліотеки Sanctum, який перевіряє запит на наявність токена авторизації, перевіряє його та авторизує або відхилює запит;
- FormRequest – клас, який надає функціональність валідації та авторизації запиту який можна унаслідувати та використати для необхідного методу класу-контролеру за допомогою dependency injection.

```
class UserController extends Controller
{
    public function index(UsersRequest $request){...}
    public function show(Request $request, $id){...}
    public function store(CreateUserRequest $request){...}
    public function update(UpdateUserRequest $request, $id){...}
    public function roles(){...}
    public function statuses(){...}
    public function projects(UserProjectsRequest $request, $id){...}
    public function payments(PaginationRequest $request, $id){...}
    public function raises(UserRaisesRequest $request, $id){...}
    public function worktime(UserWorktimeRequest $request, $id){...}
    public function calendar(UserCalendarRequest $request, $id){...}
    public function accounts(UserAccountsRequest $request, $id){...}
    public function project(UserProjectRequest $request, $id, $projectId){...}
    public function addWorktime(CreateWorktimeRequest $request, $id){...}
    public function deleteWorktime(Request $request, $id, $worktimeId){...}
}
```

Рисунок 4.3 – Клас-контроллер користувача

Про останній пункт детальніше. Класи-спадкоємці класу «FormRequest» містять першу частину функціональності RBAC, яка перевіряє загальний доступ до маршруту.

У класу «FormRequest» та його спадкоємців можна виділити наступні методи:

- `authorize()` – метод повинен повернути Булеве значення, яке і зазначить, чи матиме запит доступ;
- `rules()` – метод, який повинен повернути масив ключів - значень з правилами валідації тіла запиту. Клас «CreateUserRequest» на рисунку 4.4 відповідає за запит створення запису користувача у системі.

```
class CreateUserRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        /** @var User $user */
        $user = Auth::user();
        return $user->can(abilities: Permissions::ADD_USERS);
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'name' => 'required|string',
            'email' => 'required|string|unique:users,email',
            'rate' => 'required|numeric|min:1',
            'role' => Rule::in(UserRole::getAllValues()),
            'status' => Rule::in(UserStatus::getValues()),
            'photos' => 'nullable|array',
            'photos.*' => 'distinct|integer|exists:photos,id',
        ];
    }
}
```

Рисунок 4.4 – Клас «CreateUserRequest»

Друга половина функціональності RBAC пов'язана з двома базовими класами Laravel: «JsonResource» та «ResourceCollection». «Клас JsonResource», а саме його спадкоємці, використовуються для перетворення моделей у JSON відповідь, яка повернеться клієнту. Клас «ResourceCollection», а саме його

спадкоємці, у свою чергу відповідальні за перетворення набору моделей у колекцію та надають додатковий функціонал сортування, фільтрації та пагінації.

Базуючись на цих класах, було створено базові класи, які будуть використовуватися для побудування ресурсів та колекцій, замість «JsonResource» та «ResourceCollection»: «ApiResource» та «ApiCollection» відповідно. Даний крок було зроблено задля пов'язання функціональності RBAC з функціональністю побудування відповіді від серверу, аби зважаючи на доступи користувача, не відправляти захищені записи або поля ресурсів.

У базовому класі «ApiCollection» можна виділити наступні методи (див. рис. 4.5):

```
class ApiCollection extends ResourceCollection
{
    /**
     * @var User $user;
     */
    protected $user;

    public function __construct($resource){...}

    /** If query builder passed, adjust response by role and inbound filters ...*/
    protected function processResource($resource){...}

    /** Paginate API request ...*/
    protected function paginate(Request $request){...}

    /** Transform the resource collection into an array. ...*/
    public function toArray($request){...}

    /** For conditional usage ...*/
    protected function prepareArray($request){...}

    /* Methods below must be re-defined at target collection class in order to match your needs */

    /** Adjust response by user role ...*/
    protected function adjustByPermissions($builder){...}

    /** Filter your response by request parameters ...*/
    protected function sortAndFilter($resource, Request $request){...}

    /** Array of allowed filters that will be used on front-end ...*/
    public static function filters(){...}
}
```

Рисунок 4.5 – Базовий клас «ApiCollection»

– processResource() – головна функція, яка вирішує, чи потрібно перетворювати ресурс;

- `paginate()` – функція, яка містить поліпшену функціональність пагінації, відмінну від стандартної;
- `toArray()` – перетворення колекції ресурсів до масиву;
- `adjustByPermissions()` – функція, яка міститиме функціональність формування запиту до БД, зважаючи на доступи користувача;
- `sortAndFilter()` – функціональність сортування та фільтрування колекції;
- `filters()` – функція, яка містить масив ключ-значення, який описує за котрими полями запису можна буде сортувати.

Базовий клас «`ApiResource`» (див. рис. 4.6), який містить наступні основні методи:

- `adjustByPermissions()` – функція перетворення JSON відповіді, яка, зважаючи на доступи, приховує або додає захищені поля;
- `withFields()` – функція, яка додає до відповіді дозволені поля зв'язаної сутності, яку запросив користувач.

```
class ApiResource extends JsonResource
{
    /** @var User $user ...*/
    protected $user;

    public function __construct($resource){...}

    /** @param $request ...*/
    protected function originalArray($request){...}

    /** Transform the resource into an array ...*/
    public function toArray($request){...}

    /** Adjust final payload by user permissions ...*/
    protected function adjustByPermissions($payload){...}

    /** Additional fields to be included ...*/
    protected function withFields($payload, Request $request){...}
}
```

Рисунок 4.6 – Базовий клас «`ApiResource`»

Клас «`ApiResource`» також пов'язаний з методикою видалення непотрібних клієнту полів відповіді. Наприклад, коли веб додаток запрошує список користувачів, в таблиці необхідні лише наступні поля: унікальний ідентифікатор, ім'я, статус, дата додавання до системи. Усі інші поля, які

пов'язані з користувачем у даному випадку не мають повертатися. Цю поведінку регулює вебдодаток, додаючи до запиту параметр `mode`, від якого сервер розуміє, яку саме відповідь потребує вебдодаток. Проте у виключних випадках шаблонної відповіді за режимом недостатньо, тому тут допомагає передача параметру `with` у запиті, де додаток перераховує необхідні поля для запиту у форматі «`field,field.child_field`».

Щодо менш широко направленої функціональності, було також додано функціональність запрошень до систему, узгоджено з спроектованою архітектурою програми. Сутність запрошення також має модель та контроллер, проте має додаткову функціональність відправки e-mail повідомлень одержувачу.

Дана функціональність працює за наступним алгоритмом:

- користувач запросив до системи персонал, створивши запис у БД за допомогою запиту на сервер;
- подію створення запису отримує так званий спостерігач (Observer), котрі є базовою функціональністю Laravel;
- клас-спостерігач, у відповідь на подію, посилає за вказаною e-mail адресою листа з посиланням до форми вводу даних про персону.

На рисунку 4.7 представлено клас «InvitationObserver».

```
class InvitationObserver
{
    public function created(Invitation $invitation)
    {
        Mail::to($invitation->email)->send(new InviteUser($invitation));
    }
}
```

Рисунок 4.7 – Клас-спостерігач «InvitationObserver»

Даний клас обсервер має метод «`created()`» який приймає об'єкт запрошення після додавання останнього до БД. За допомогою фасаду Mail (статичного допоміжного класу-сервісу) обсервер відправляє повідомлення, яке побудоване за допомогою класу «InviteUser» (див. рис. 4.8). Клас наслідує

стандартний клас Laravel «Mailable», який використовується для описання вмісту повідомлення (метод «build()»).

```
class InviteUser extends Mailable
{
    use SerializesModels;

    /**
     * @var Invitation
     */
    private $invitation;

    /**
     * Create a new message instance.
     *
     * @param Invitation $invitation
     */
    public function __construct(Invitation $invitation)
    {
        $this->invitation = $invitation;
        $this->subject = $invitation->inviter->name . ' has invited you to erp';
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build()
    {
        return $this->view('mail.invitation')->with([
            'inviter' => $this->invitation->inviter,
            'inviteLink' => config('app.front_url') . '/invite/' . $this->invitation->code
        ]);
    }
}
```

Рисунок 4.8 – Клас «InviteUser»

Не менш важливою функціональністю є генерування рахунків фактур – стандартна процедура фінансових взаємовідношень між компанією та клієнтом. Зазвичай це займає багато часу, проте за допомогою даної функціональності це можна зробити в один клік. Було створено клас-сервіс «InvoiceGenerator», який містить функціональність перетворення вхідних даних у файл (див. рис. 4.9).

Основний метод класу – «generate()», який утворює запит необхідної структури, та запитує за нею файл у відкритому програмному HTTP інтерфейсі invoice-generator.com.

```

class InvoiceGenerator
{
    const URL = 'https://invoice-generator.com';

    const DEFAULT_OPTIONS = [...];

    const TAX_TYPE_MAP = [...];

    protected static function getClient(){...}

    /** @param Project $project ...*/
    public static function generate(Project $project, Account $account, $items, $options = [])
    {
        $data = array_merge([
            'currency' => $options['currency'] ?? self::DEFAULT_OPTIONS['currency'],
            'due_date' => $options['due_date'] ?? Carbon::now()->addDays( value: 2)->toDateString(),
            'date' => $options['date'] ?? Carbon::now()->toDateString(),
            'quantity_header' => $options['quantity_header'] ?? self::DEFAULT_OPTIONS['quantity_header'],
            'unit_cost_header' => $options['unit_cost_header'] ?? self::DEFAULT_OPTIONS['unit_cost_header'],
            'number' => $options['number'] ?? $account
                ->payments()
                ->where( column: 'project_id', operator: '=', $project->id)
                ->count() + 1,
            'from' => $options['from'] ?? $account->user->name . PHP_EOL . $account->user->email,
            'to' => $options['to'] ?? $project->client->name . PHP_EOL . $project->client->email,
            'notes' => $options['notes'] ?? 'Kindly make a payment at your earliest convenience.' . PHP_EOL .
                'Please do not hesitate to contact me with any questions.' . PHP_EOL .
                'Many thanks, ' . PHP_EOL .
                $account->user->name,
            'items' => array_map(function ($item) { return [...]; }, $items),
            'fields' => [...],
            'tax_title' => $options['tax_title'] ?? 'Tax',
            'tax' => $options['tax_amount'] ?? 0
        ]);

        try {
            $guzzle = static::getClient();
            $response = $guzzle->post( uri: '/', ['form_params' => $data]);
            $path = 'invoices/' . 'invoice-' . Str::random( length: 6 ) . '.pdf';
            Storage::disk( name: 'public')->put($path, $response->getBody());
            return Storage::disk( name: 'public')->path($path);
        } catch (\GuzzleHttp\Exception\GuzzleException $exception) {
            Log::error($exception);
            throw new InvoiceGenerationException;
        }
    }
}

```

Рисунок 4.9 – Клас-сервіс «InvoiceGenerator»

4.2.2 Розробка вебдодатку програмного комплексу

Зважаючи на змодельовану архітектуру та бізнес-логіку програмного комплексу, було визначено наступні структурні та функціональні компоненти вебдодатку:

- функціональність та сторінка авторизації;
- загальний компонент, який відповідатиме за перевірку стану авторизації користувача та робитиме необхідні дії зважаючи на цей стан;
- сторінки для усіх сутностей, з якими оперує система: користувач, проєкт, клієнт, робочий час, платіж, підвищення, запрошення, а також

функціональність запиту, додавання, редагування та видалення записів з БД через інтерфейс;

- функціональність та інтерфейс запрошень до системи;
- компонент, який перевірятиме доступ до сторінки, а також доступ до дії (кнопки, посилання тощо);
- функціональність та інтерфейс навігації між сторінками.

За допомогою Ant design побудування інтерфейсів не займає багато часу та, як правило, не має особливих зауважень щодо реалізації, які варті уваги. Нижче представлена форма авторизації, створена за допомогою Ant design компоненту «Form» та його дочірніх підкомпонентів «Form.Item» та «Input» (див. рис 4.10).

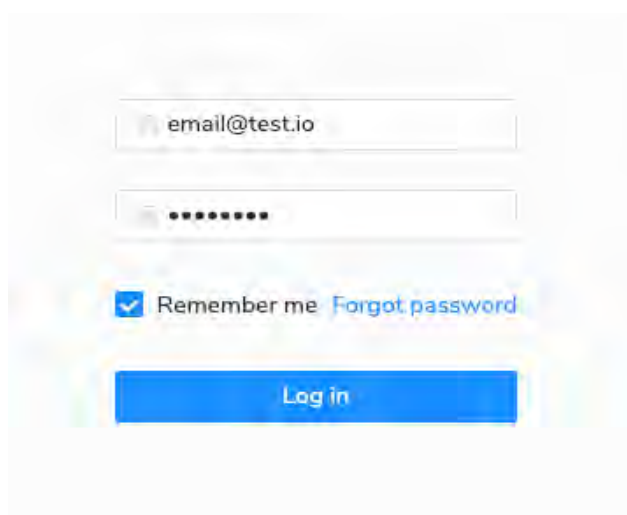


Рисунок 4.10 – Графічний інтерфейс форми авторизації

Варто зауважити, що Ant design також має вбудовану функціональність валідації форм, один із способів використання якої є компонент «Form.Item» та його атрибут `rules`, в який передається набір необхідних правил у спеціальному форматі. Також функціональність валідації має інтерфейс відображення помилок, який також можна використати для відображення помилок від серверу, наприклад, при неправильних даних.

Щоб розділити та захистити маршрути вебдодатку, узгоджено із визначеними структурними компонентами, було створено компоненти «Layout»

та «AuthorizedLayout». Компонент «Layout» буде використаний для сторінок, які не потребують авторизації (див. рис. 4.11). Наразі у проєкті вебдодатку є лише одна сторінка, яка використовує даний компонент – сторінка авторизації.

```
const Layout = ({ title, children }) => (
  <>
    <Head title={title} />
    <AntLayout className={styles.container}>
      { children }
    </AntLayout>
  </>
);

Layout.propTypes = {
  title: PropTypes.string,
  children: PropTypes.any.isRequired,
};

Layout.defaultProps = {
  title: 'ERP',
};

export default Layout;
```

Рисунок 4.11 – Компонент «Layout»

Як зрозуміло з коду компоненту, він являється «обгорткою» для інших компонентів. Також в коді присутній компонент «Head», який необхідний фреймворку Next.js для динамічної генерації тегу `<head />` та його вмісту.

Компонент «AuthorizedLayout» являється обгорткою для усіх сторінок для яких необхідна авторизація користувача. Компонент матиме функціональність запиту авторизованого користувача з серверу, якщо інформація про останнього не присутня у глобальному стані додатку. Також компонент містить дочірні компоненти «Header» та «Sidebar», необхідні для навігації між сторінками. (див. рис. 4.12).

Основна частина функціональності та інтерфейсу додатку – це сторінки управління сутностей, а саме: сторінка зі списком записів сутності, сторінка профілю сутності, сторінка додавання запису сутності до БД, сторінка редагування запису сутності. Більшість сутностей системи мають даний набір функціональності та маршрутів, тож далі буде показано функціональність на прикладі сутності клієнта.

```

import ...

const AuthorizedLayout = ({ title, children }) => {
  const dispatch = useDispatch();
  const [user] = useSelector(signedUserSelector);
  const [sidebarCollapsed, toggle] = useState( initialState: false);

  useEffect( effect: () => {
    if (!user) dispatch(fetchSignedUser());
  }, deps: [user, dispatch]);

  return (
    <>
      <Head title={title} />
      <AntLayout className={styles.container}>
        <Sidebar collapsed={sidebarCollapsed} />
        <AntLayout className='site-layout'>
          <AntLayout.Header className={classNames(styles.background, styles.header)}>
            {React.createElement( a: sidebarCollapsed ? MenuUnfoldOutlined : MenuFoldOutlined, b: {
              className: styles.trigger,
              onClick: () => toggle( value: (state : boolean ) => !state),
            })}
            <AvatarArea />
          </AntLayout.Header>
          <AntLayout.Content
            style={{
              margin: '24px 16px',
              minHeight: 'unset',
            }}
          >
            { children }
          </AntLayout.Content>
        </AntLayout>
      </AntLayout>
    </>
  );
};

AuthorizedLayout.propTypes = {
  title: PropTypes.string,
  children: PropTypes.any.isRequired,
};

AuthorizedLayout.defaultProps = {
  title: 'dygit',
};

export default AuthorizedLayout;

```

Рисунок 4.12 – Компонент «AuthorizedLayout»

Компонент «Clients» використовує компонент «Table» для побудування таблиці записів, а також використання вбудованої функціональності пагінації, яку легко використати для запитів даних із серверу (див. рис. 4.13).

Також компонент «Table» надає широкий набір функціональності для сортування та фільтрування, яка конфігурується після отримання першої відповіді від серверу на запит отримання записів сутності. Компонент

автоматично викличе функцію запиту записів клієнтів з серверу при першому завантаженні сторінки.

```
import ...

const COLUMNS = [...];

const Clients = () => {
  const dispatch = useDispatch();
  const [clients, total, loading] = useSelector(clientsSelector);
  const [pagination, setPagination] = useState( initialState: {
    page: 1,
    size: 10,
  });

  const handlePaginationChange = useCallback( callback: (page, size) => {
    setPagination( value: { page, size });
  }, deps: []);

  useEffect( effect: () => {
    dispatch(fetchClients( query: { ...pagination, mode: RESPONSE_MODE.SIMPLIFIED }));
  }, deps: [dispatch, pagination]);

  return (
    <>
      <SuspenseLoader loading={loading} className={styles.buttons}>
        <Space className={styles.buttons}>
          <GuardedLink href='/clients/new' gate={PERMISSION.ADD_CLIENTS} hideIfFailed>
            <Button type='primary'>
              Add
            </Button>
          </GuardedLink>
        </Space>
      </SuspenseLoader>
      <Table
        loading={loading}
        dataSource={clients}
        columns={COLUMNS}
        pagination={{
          total,
          showSizeChanger: true,
          onShowSizeChange: handlePaginationChange,
          onChange: handlePaginationChange,
        }}
      />
    </>
  );
};

export default Clients;
```

Рисунок 4.13 – Компонент «Clients»

Також варто звернути увагу на компонент «GuardedLink» який використовує у собі компонент «Can» для перевірки доступів авторизованого користувача (див. рис. 4.14). Обидва компоненти використовуються майже у всіх компонентах-сторінках вебдодатку.

```

import ...

function retrieveIdsFromPermission(permission) {...}

const Can = ({
  perform, yes, no, loading,
}) => {
  const [user] = useSelector(signedUserSelector);

  const check = useCallback( callback: (permission, type) => {
    if (user) {...}
    return false;
  }, deps: [user]);

  const canPerform = useMemo( factory: () => {...}, deps: [perform, check]);

  if (!user) return loading || null;

  return canPerform ? yes : no;
};

Can.propTypes = {
  perform: PropTypes.oneOfType( arrayOfTypeCheckers: [PropTypes.string, PropTypes.shape( shapeTypes: {
    all: PropTypes.arrayOf(PropTypes.string),
    any: PropTypes.arrayOf(PropTypes.string),
    except: PropTypes.arrayOf(PropTypes.string),
  })]).isRequired,
  yes: PropTypes.any.isRequired,
  no: PropTypes.any,
  loading: PropTypes.element,
};

Can.defaultProps = {
  no: null,
  checkType: 'all',
};

export default Can;

```

Рисунок 4.14 – Компонент «Can»

Обидві сторінки додавання та редагування запису сутності використовують компонент «ClientForm», який містить у собі функціональність що описує усі можливі поля заповнення та їх валідацію (див. рис. 4.15).

Також компонент вміє запитувати та форматовувати інформацію із серверу, яка необхідна для побудування форми.

```

import ...

const ClientForm = ({
  onSubmit, submitting, initialValues, formRef, ...props
}) => {
  const dispatch = useDispatch();
  const uploader = useRef( initialValue: null);
  const [form] = Form.useForm();
  const [additionalFields, setFields] = useState(form.getFieldsValue( nameList: ['fields']).fields);
  const [origins, originsLoading] = useSelector(clientOriginsSelector);
  const [fieldTypes, fieldTypesLoading] = useSelector(clientFieldTypesSelector);
  const [uploadPhoto, deletePhoto] = usePhotos(form, uploader);

  const originOptions = useMemo( factory: () => origins.map((o) => ({ value: o, label: o })), deps: [origins]);

  const fieldTypeOptions = useMemo( factory: () => fieldTypes.map((f) => ({ value: f, label: ucFirst(f) })), deps: [fieldTypes]);

  const renderLinksField = useCallback( callback: (fields, { remove }) => fields.map((field, idx) => (
    <Row key={idx.toString()} gutter={10}>...>
  )), deps: [form, additionalFields, fieldTypeOptions, fieldTypesLoading]);

  const handleCopyDetails = useCallback( callback: (val) => {...}, deps: [form]);

  const clientDetails = useCallback( callback: (prefix = '', noOrigin) => (
    <...>
  ), deps: [originOptions, originsLoading]);

  useEffect( effect: () => {
    dispatch(fetchClientOrigins());
    dispatch(fetchClientFieldTypes());
  }, deps: [dispatch]);

  useEffect( effect: () => {...}, deps: [form, initialValues]);

  useEffect( effect: () => {...}, deps: [form, formRef]);

  return (
    <Form
      form={form}
      layout='vertical'
      className={styles.form}
      initialValues={initialValues}
      {...props}
    >...>
  );
};

```

Рисунок 4.15 – Компонент «ClientForm»

Також компонент вміє запитувати та форматовувати інформацію із серверу, яка необхідна для побудування форми.

Компонент профілю клієнта «ClientProfile» використовується для автоматичного запиту даних останнього із серверу та відображення даних запису сутності даного клієнта (див. рис. 4.16).

Компонент використовує загальний компонент «Can», щоб приховувати від перегляду захищені блоки.

```

const ClientProfile = () => {
  const dispatch = useDispatch();
  const { back, query } = useRouter();
  const [client, loading, isFound] = useSelector(clientSelector);

  useEffect( effect: () => {
    if (query.id) dispatch(fetchClient(query.id));
  }, deps: [dispatch, query]);

  const renderTab = useCallback( callback: (tab, idx) => (
    <Tabs.TabPane
      key={idx}
      tab={{
        <span...>
      }}
    ...>
  ), deps: []);

  const content = useMemo( factory: () => (
    <Row gutter={[16, 16]}...>
  ), deps: [client, renderTab, loading]);

  return (
    <
      <PageHeader
        className={styles.pageHeader}
        title='Client Profile'
        subTitle={client?.name}
        onBack={back}
        extra={[
          <GuardedLink
            key={0}
            gate={{ any: [wildcard(PERMISSION.EDIT_CLIENTS, client.id)] }}
            href='/clients/[id]/edit'
            as={` /clients/${client.id}/edit`}
          >
            <Button type='primary'>Edit</Button>
          </GuardedLink>,
        ]}
      />
      {
        !isFound && !loading
          ? <Result status='404' title='Not Found' subTitle='Sorry, such client does not exist.' />
          : content
      }
    </>
  );
};

```

Рисунок 4.16 – Компонент профілю клієнта «ClientProfile»

4.3 Розгортання програмного комплексу

Після завершення основної частини розробки програмного комплексу, останній було розгорнуто у хмарному середовищі Vercel (вебдодаток) та Digitalocean (серверний додаток). Розподілення додатків між середовищами було вирішено щоб зменшити можливість поганої роботи системи, яка працює на одній віртуальній машині.

4.3.1 Розгортання вебдодатку у хмарному середовищі Vercel

Сервіс Vercel надає простий та швидкий спосіб розгортання будь-якого вебдодатку, який написаний за допомогою підтримуваного сервісом фреймворку (в даному випадку – Next.js). Для розгортання достатньо зв'язати проєкт на Vercel з репозиторієм на Github, або скористатися ручним імпортуванням (див. рис. 4.17).

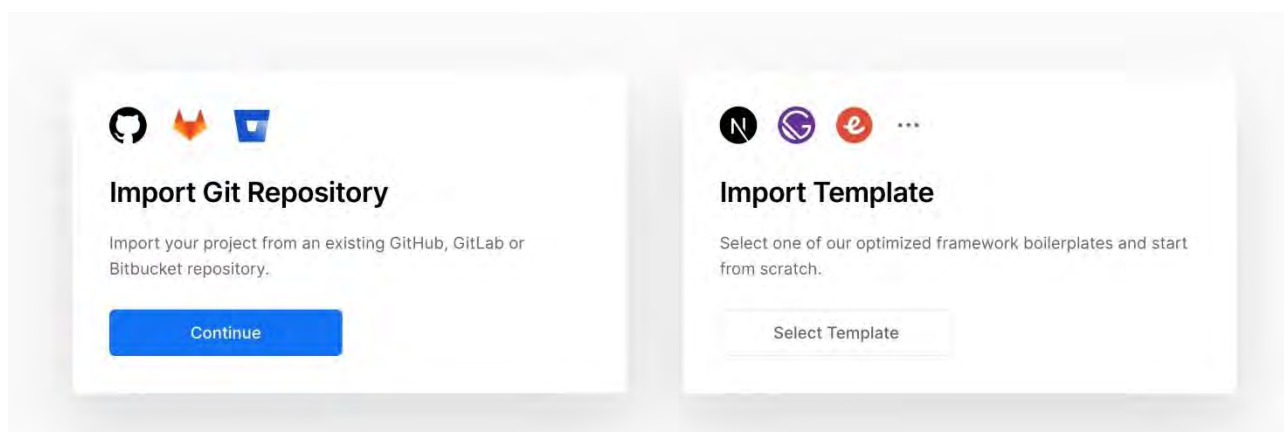


Рисунок 4.17 – Сторінка створення проєкту на Vercel

Під час імпортування сервіс надає змогу налаштувати або встановити змінні середовища, які використовує додаток, та додати свій домен, проте система за замовчуванням виділяє домен для безкоштовного використання.

Також варто відмітити наявність підтримки CI/CD, який пов'язаний з віткою master імпортованого репозиторію, тож після кожного оновлення у вітці, Vercel автоматично побудує нову версію додатку без втручання розробника.

4.3.2 Розгортання серверного додатку у хмарному середовищі DigitalOcean

DigitalOcean, Inc. – американський провайдер хмарних інфраструктур з центрами обробки даних по всьому світу. Сервіс надає хмарні послуги для розробників, дає можливість розгорнути і масштабувати додатки одночасно на

декількох комп'ютерах. Станом на лютий 2017, DigitalOcean була другою найбільшою хостинг-компанією за кількістю загальнодоступних серверів із заявленим аптаймом на рівні 99.99%.

Згідно з вимогами до апаратної частини ПЕВМ, на сервісі DigitalOcean було створено віртуальну машину (дроплет) з наступними параметрами:

- ОС Ubuntu 20.10 x64;
- 1 ГБ оперативної пам'яті;
- 25 ГБ вільного простору на жорсткому диску.

Сервіс надає широкий інструментарій управління та моніторингу за створеними дроплетами, такі, як: графіки використання ресурсу центрального процесору, графік пропускної здатності, управління живленням дроплету, управління доступом тощо (див. рис. 4.18)

На створеному дроплеті було проведено наступні дії:

- розгорнуто репозиторій із серверним додатком;
- встановлено інсталяцію серверу MySQL;
- налаштовано .env файл Laravel;
- застосовано міграції до БД;
- сконфігуровано HTTP сервер Nginx.

Після проведення даних операцій, сервер готовий до використання.



Рисунок 4.18 – Дашборд дроплету на DigitalOcean

4.4 Висновки за розділом 4

У даному розділі було розглянуто обрані бібліотеки, описано основні етапи програмної реалізації ERP-системи, розглянуті та проілюстровані найважливіші нюанси щодо розробки окремих компонентів програмного комплексу. Додатково у розділі було розглянуто сервіси для хмарного розгортання Vercel та DigitalOcean, а також основні деталі щодо розгортання веб- та серверного додатків.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМНОГО КОМПЛЕКСУ

У цьому розділі дано визначення програмного комплексу і визначено декілька прикладів, які освітлюють основну функціональність програмного комплексу. Також було висвітлено основні принципи тестування та налагодження розгорнутого додатку.

5.1 Опис застосування програмного комплексу

Програмний комплекс ERP призначений для оптимізації менеджменту ресурсами та для поліпшення аудиту фінансів компанії. Основна функціональність комплексу: формування зв'язків між записами сутностей та їх організація, підрахунок витраченого робочого часу на проєкті а загалом, підрахування прибутків та витрат на проєкті, генерація рахунків-фактур. Програмний комплекс не є самостійним та вимагає додаткове програмне забезпечення для роботи:

- підключення до БД MySQL;
- наявність встановленого інтерпретатору PHP версії 7.3 або вище, а також його розширення: ext-mysqld, ext-json. Також необхідний менеджеру пакетів PHP – Composer, версії 1.2 та вище, для встановлення модулів серверного додатку;
- наявність встановленого програмного середовища Node.js версії 12.14.0 або вище, а також менеджер пакетів NPM, який розповсюджується з даною версією Node, для встановлення модулів вебдодатку;
- наявність зконфігурованого HTTP-серверу для маршрутизації запитів до програмного комплексу.

5.2 Інструкція по роботі з системою ERP

5.2.1 Запуск програмного комплексу

Для успішного запуску усього програмного комплексу, необхідно провести дії, за умови, що усі необхідні вимоги, перелічені у попередньому пункті виконані.

Для запуску вебдодатку програмного комплексу необхідно:

- перейти за допомогою терміналу до директорію з додатком;
- увести команду «`npm i && npm run start`», яка встановить пакети додатку та запустить його, що зробить його доступним за посиланням <http://localhost:3000> (за замовчуванням).

Для запуску серверного додатку програмного комплексу необхідно:

- перейти за допомогою терміналу до директорію з додатком;
- увести команду «`composer install && php artisan serve`», яка встановить пакети додатку та запустить його, що зробить його API інтерфейс доступним за посиланням <http://localhost:8000> (за замовчуванням).

5.2.2 Методика роботи з програмним комплексом

Для того, щоб користувач мав змогу користуватися системою, йому потрібно перейти на домашню сторінку додатку та увійти за допомогою форми авторизації. Далі, система переводить користувача на сторінку зі списком користувачів за замовчуванням (див. рис. 5.1). Далі перераховано основні складові графічного інтерфейсу сторінки-списку, на прикладі списку користувачів.

Графічний інтерфейс цієї сторінки складається з наступних компонентів:

- 1 – посилання до сторінок зі списками сутностей;
- 2 – кнопка приховування бокової панелі;
- 3 – блок з аватаром та ім'ям авторизованого користувача, який є кнопкою, що показує меню з опцією виходу із системи;

- 4 – таблиця з користувачами;
- 5 – кнопки «Add», що переводять користувача на сторінку з формою створення користувача, та кнопка «Invite», що відкриває модальне вікно з формою створення запрошення до системи;
- 6 – Ім'я користувача, що є посиланням на його профіль;
- 7 – кнопки управління пагінацією.

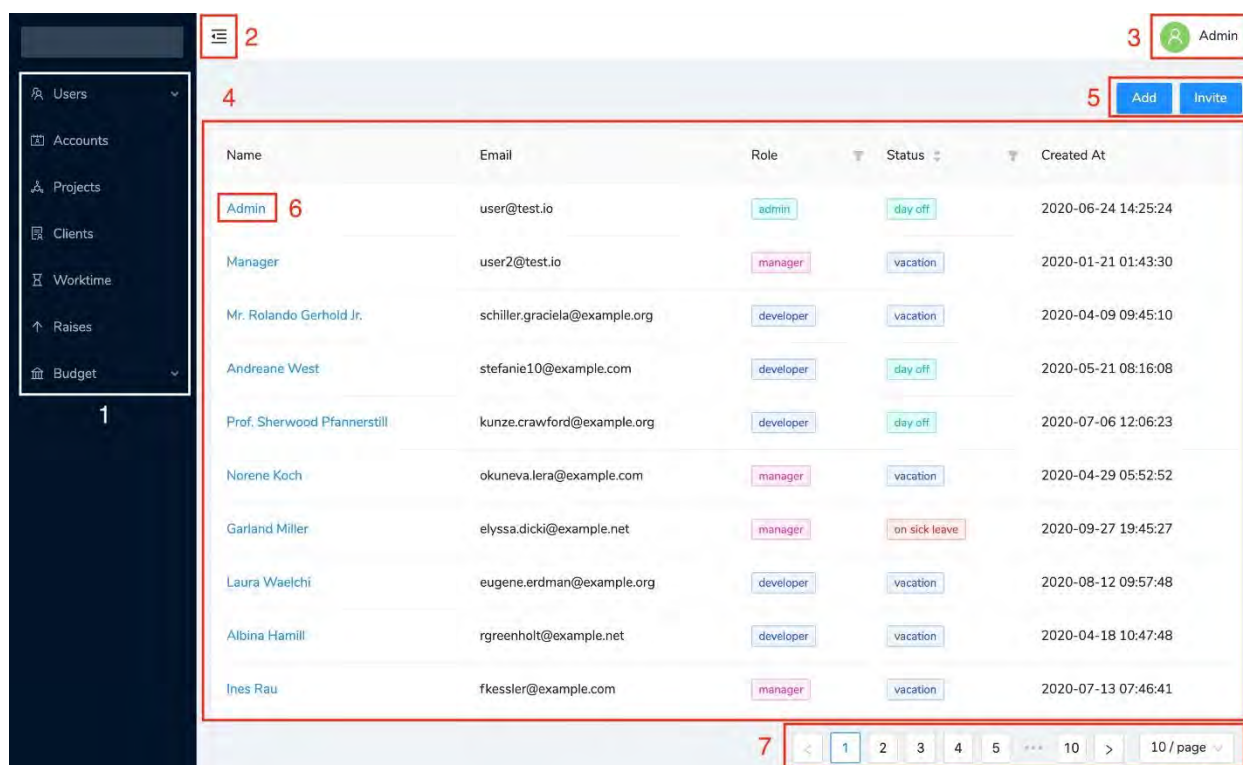


Рисунок 5.1 – Графічний інтерфейс сторінки зі списком користувачів

Складові графічного інтерфейсу сторінка з формою додавання запису сутності показано на прикладі форми додавання користувача (див. рис. 5.2):

- 1 – блок з назвою сторінки та посиланням для переходу на попередню сторінку;
- 2 – форма додавання користувача;
- 3 – обов'язкове поле для заповнення;
- 4 – необов'язкове поле для заповнення;
- 5 – кнопка підтвердження форми та відправки запиту на додавання запису до БД.

Компоненти навігації являються незмінними.

The screenshot shows a web application interface for adding a new user. The form is titled "Add User" and is located within a sidebar navigation menu. The form contains several input fields and buttons, which are highlighted with red boxes and numbered 1 through 5:

- 1**: A red box highlights the back arrow and the "Add User" title.
- 2**: A red box highlights the entire form container.
- 3**: A red box highlights the "Name" and "Status" input fields.
- 4**: A red box highlights the "Email" and "Rate" input fields.
- 5**: A red box highlights the "Photo" upload section, which includes a "Click to upload" button, a text input for a link, and an "Upload" button. Below this, there is a "Submit" button.

Рисунок 5.2 – Графічний інтерфейс форми додавання нового користувача до БД

Складові сторінки профілю сутності перелічено на прикладі профілю користувача (розробника) (див. рис. 5.3):

- 1 – посилання на сторінку редагування запису;
- 2 – загальна інформація про користувача;
- 3 – скалькульована інформація про фінансовий оборот користувача;
- 4 – календар робочого часу;
- 5 – зв'язана з записом інформація у таблицях (проєкти, платежі, підвищення, робочий час).

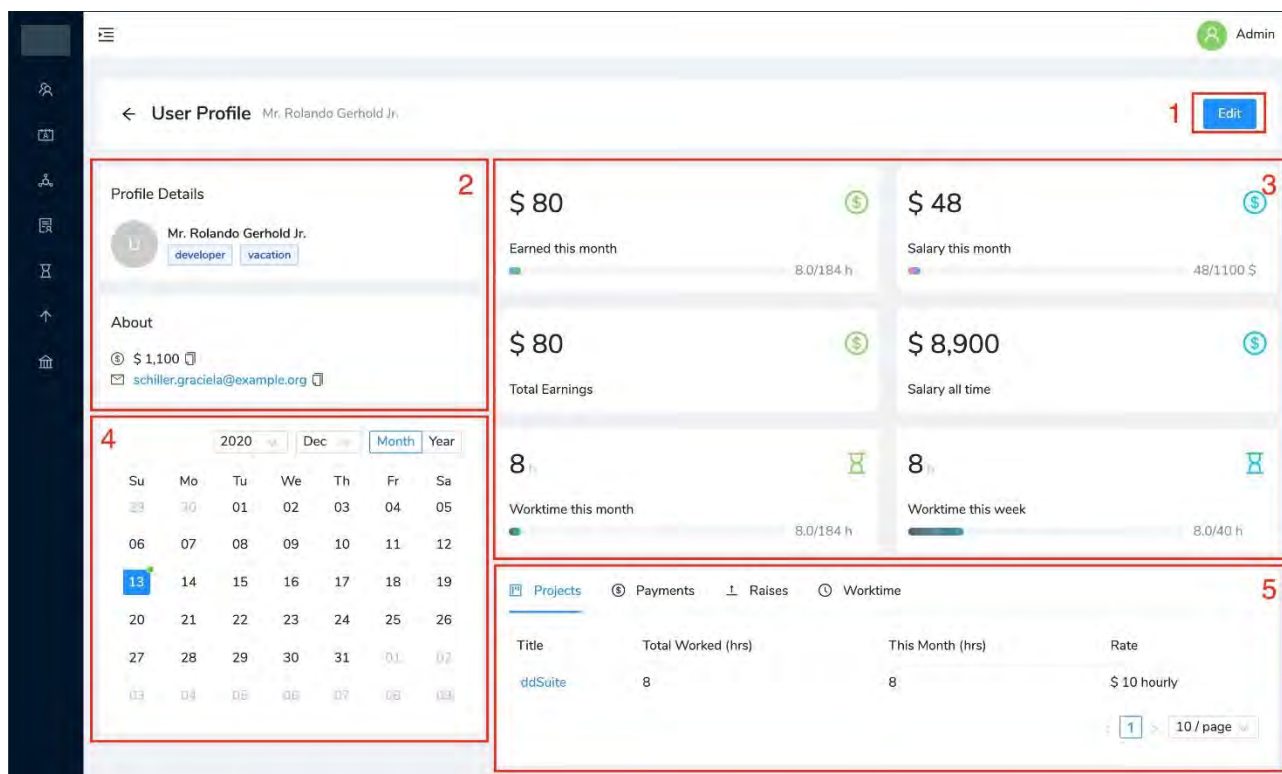


Рисунок 5.3 – Графічний інтерфейс профілю користувача (розробника)

5.3 Методика тестування програмного комплексу

Для проведення коректного тестування роботоспроможності програмного комплексу, для бази даних було створено декілька класів-сідерів, які наповнюватимуть її умовними даними. Також, було встановлено додаткову бібліотеку для моніторингу помилок серверного додатку – Laravel Telescope, що надає широкий набір інструментів для моніторингу та налагодження додатку. Бібліотека надає власний графічний інтерфейс для додатку, за допомогою якого і можна оперувати можливостями Telescope (див. рис. 5.4).

Друга фаза тестування, робота з даними користувачем, головне завдання якого – це перевірка правильності калькульованої інформації та утворених зв'язків між сутностями.

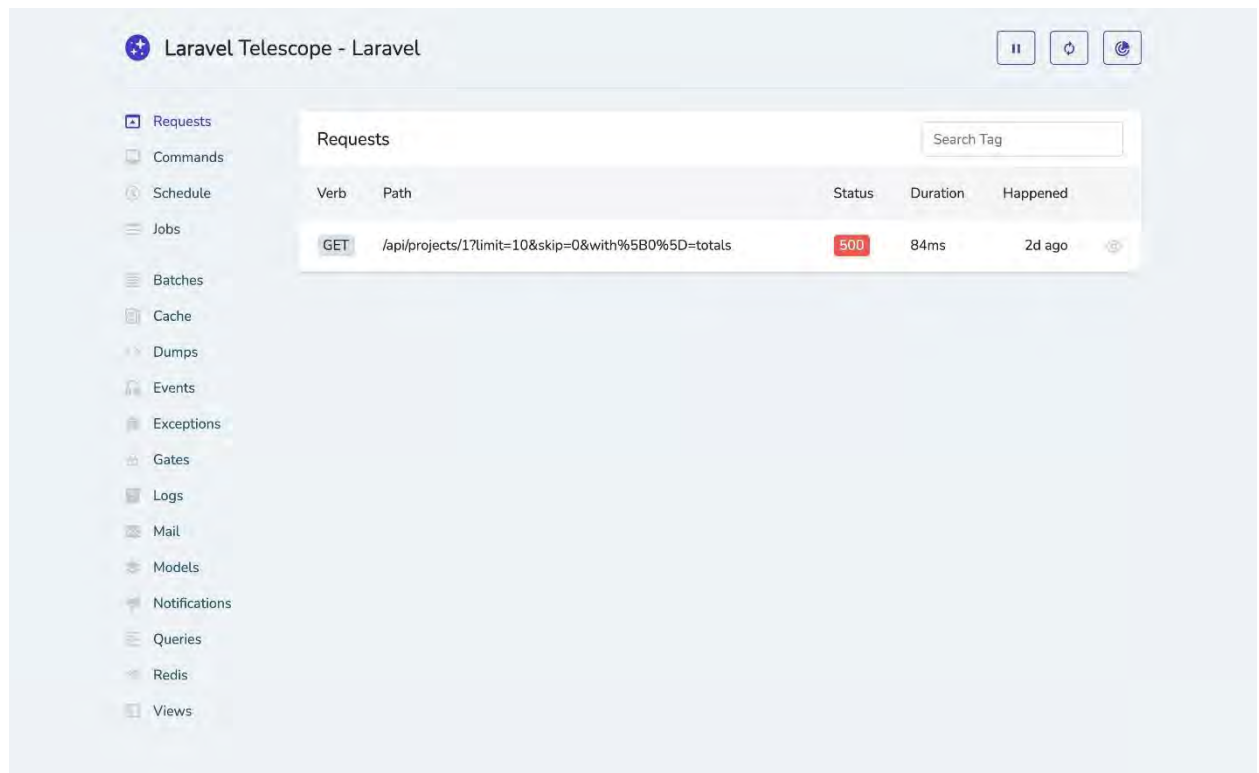


Рисунок 5.4 – Графічний інтерфейс бібліотеки Laravel Telescope

5.4 Практичні завдання для тестування та експериментального дослідження програмного комплексу

Для визначення даних, які необхідні для уведення в систему задля її тестування, треба припустити випадковий сценарій.

Припустимо, що клієнт замовив у компанії вебсайт, який повинні розробляти два розробники з погодинною ставкою у 10\$ та 8\$ відповідно.

Тижневий ліміт у робочому часі – не менше ніж 40 годин. Обидва розробники оцінили свою частину роботи в 160 і 180 годин виконання відповідно. Замовник також підвищить погодинну ставку другому розробнику до 10\$, починаючи з 80 години виконання роботи.

Тож можна навести підсумки даного сценарію тестування (див. табл. 5.1).

Також, припустимо, що місячний оклад розробників: 10000 та 12000 гривень відповідно, а дата старту проєкту – 01.10.2020.

Таблиця 5.1 – Вимоги до тестового проєкту

Розробники	Вимоги до проєкту			
	Тижневий ліміт робочого часу	Погодинна ставка (\$)	Час до виконання	Підвищення ставки на проєкті
Розробник 1	40 годин або більше	10	160 годин	Ні
Розробник 2	40 годин або більше	8	180 годин	Починаючи з 80 години виконання

Для оцінки результатів тестування системи, увагу було приділено коректності калькульованих фінансових даних.

5.5 Результати тестування та експериментального дослідження програми

Після введення даних та створення необхідних сутностей в ERP системі, було встановлено наступні результати калькуляції доходу та витрат (див. табл. 5.2).

Таблиця 5.2 – Доходи та витрати за тестовим проєктом

Походження даних	Дохід розробника 1 (\$)	Дохід розробника 2 (\$)	Зарплатня розробника 1 (грн)	Зарплатня розробника 2 (грн)
Система	1600	1640	9 090,9	10 909,10
Власна калькуляція	1489.6	1526.84	9 090,9	10 909,10

Як видно з таблиці, дані відрізняються через те, що система не враховує комісію банківських транзакцій та податок на прибуток. Власні розрахунки прибутку розробників були проведені за допомогою наступної формули (5.1):

$$E = (sum - sum \cdot C) - (sum - (sum \cdot C)) \cdot T, \quad (5.1)$$

де E – чистий прибуток;

sum – сума оплати за виконану роботу замовником;

C – коефіцієнт комісії за банківську транзакцію; $C = 0,2$;

T – коефіцієнт податку на прибуток; $T = 0,5$;

Рекомендується поліпшити калькуляцію прибутку в системі за допомогою додавання до формули калькуляції необхідних коефіцієнтів.

5.6 Висновки за розділом 5

У даному розділі було описано розроблений програмний комплекс, його функціональність та графічний інтерфейс користувача. Було описано основні методи тестування та налагодження системи, а також було поставлено завдання для його тестування, що дозволило експериментально підтвердити ефективність системи, та скласти рекомендації щодо поліпшення системи.

6 ЕКОНОМІКО-ОРГАНІЗАЦІЙНА ЧАСТИНА

6.1 Опис ідеї

Розроблюваний програмний комплекс дозволяє ІТ підприємствам інтегрувати до свого робочого процесу інструменти оптимізації та балансування ресурсів. Програмний комплекс підвищує загальний виробничий показник підприємства шляхом оптимізації та автоматизації завдань управлінського персоналу, а також підвищує якість фінансового аудиту. Зміст ідеї, напрямки застосування та вигода для споживачів наведені у таблиці 6.1.

Таблиця 6.1 – Опис ідеї проєкту

Зміст ідеї	Напрямки застосування	Вигоди для споживачів (користувачів)
Програмний комплекс, направлений на підвищення загальної продуктивності управлінського персоналу та підвищення виробничого показника підприємства	1. Менеджмент	Поліпшений робочий процес та автоматизація рутинних завдань.
	2. Виробнича сфера	
	3. Контроль фінансів	

6.2 Аналіз ринкових можливостей запуску проєкту

Наразі, ринок ERP-систем наповнений різноманітними програмними рішеннями, які, у переважній більшості, направлені на інтегрування до промислово-продуктових компанії, або на загальне користування, що породжує попит на більш специфічні програмні рішення, направлені на інтеграцію в певну галузь [25]. Показники стану ринку та характеристика наведені у таблиці 6.2.

Таблиця 6.2 – Попередня характеристика потенційного ринку

№	Показники стану ринку (найменування)	Характеристика
1	Головні конкуренти: Бітрікс 24 1С:ERP OneBox	Сервіси для організації робочого процесу підприємства
2	Динаміка ринку (якісна оцінка)	Зростає
3	Наявність обмежень для входу (вказати характер обмежень)	Фінансові обмеження
4	Специфічні вимоги до стандартизації та сертифікації	

Враховуючи кількість головних гравців по ринку, зростаючу динаміку ринку, невелику кількість конкурентів та середню норму рентабельності можна зробити висновок, що на даний момент, ринок для входження продукту є привабливим та надає змогу реалізувати продукцію на світовому ринку, оскільки на національному ринку економічний розвиток не надає необхідного результату для вдалої економічної реалізації проєкту.

У таблиці 6.3 наведено попередню характеристику цільових груп клієнтів.

Таблиця 6.3 – Попередня характеристика потенційних клієнтів

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Вимоги споживачів (користувачів)
1	Тайм-менеджмент	Малий та середній бізнес	Контролювати робочий час працівників
2	Контроль виплат	Малий та середній бізнес	Формувати виплати та генерувати рахунки-фактури клієнтам
3	Ресурс-менеджмент	Малий та середній бізнес	Виділяти певну кількість співробітників на виконання того чи іншого проєкту

В таблиці 6.4 наведено аналіз сильних та слабких сторін, можливості та загрози програмному продукту.

Таблиця 6.4 – SWOT-аналіз

Сильні сторони:	Слабкі сторони:
-Швидкість використання -Інтуїтивне користування -Низька складність -Більш низька вартість в порівнянні з конкурентами	- Велика конкурентна середа
Можливості:	Загрози:
- Можливість розширення - Можливість виділення платформи як послугу для окремої компанії - Можливість охопити велику кількість ринків	- Загроза стагнації та занепаду у випадку недоліку інвестицій - Загроза поглинання конкурентами - Загроза низької рентабельності

6.3 Планування розробки програмного комплексу

Для розрахунку витрат, необхідних для розробки програмного комплексу ERP-системи було визначено основні етапи планування та розробки, а також їх тривалість.

У розробці бере участь команда з 2 програмістів, яка розроблятиме програмний комплекс протягом 3 місяців. Початок розробки припадає на 1 вересня і закінчиться 1 грудня 2020 року. Характеристика робіт наведена у таблиці 6.5.

Таблиця 6.5 – Характеристика робіт з розробки програмного комплексу ERP-системи

Етап	Трудовістість		Виконавці	Тривалість, дні
	люд.-дні	% до підсумку		
1. Аналіз предметної області	4	3,63	Програмісти	2
2. Визначення вимог до програмного комплексу	8	7,27	Програмісти	4
3. Узгодження та документування вимог	4	3,63	Програмісти	2
4. Розробка архітектури програмного комплексу	8	7,27	Програмісти	4
5. Моделювання програмного комплексу	10	9,09	Програмісти	5
6. Програмна реалізація програмного комплексу	50	45,45	Програмісти	32
7. Тестування та налагодження програмного комплексу	16	14,54	Програмісти	10
8. Складання програмної документації	10	9,09	Програмісти	5
Разом	110	100		64

За даними, наведеними у таблиці 6.5, було складено стрічковий графік розробки програмного комплексу (див. рис. 6.1). Так як персонал складається тільки з програмістів, колір заповнення – суцільний.

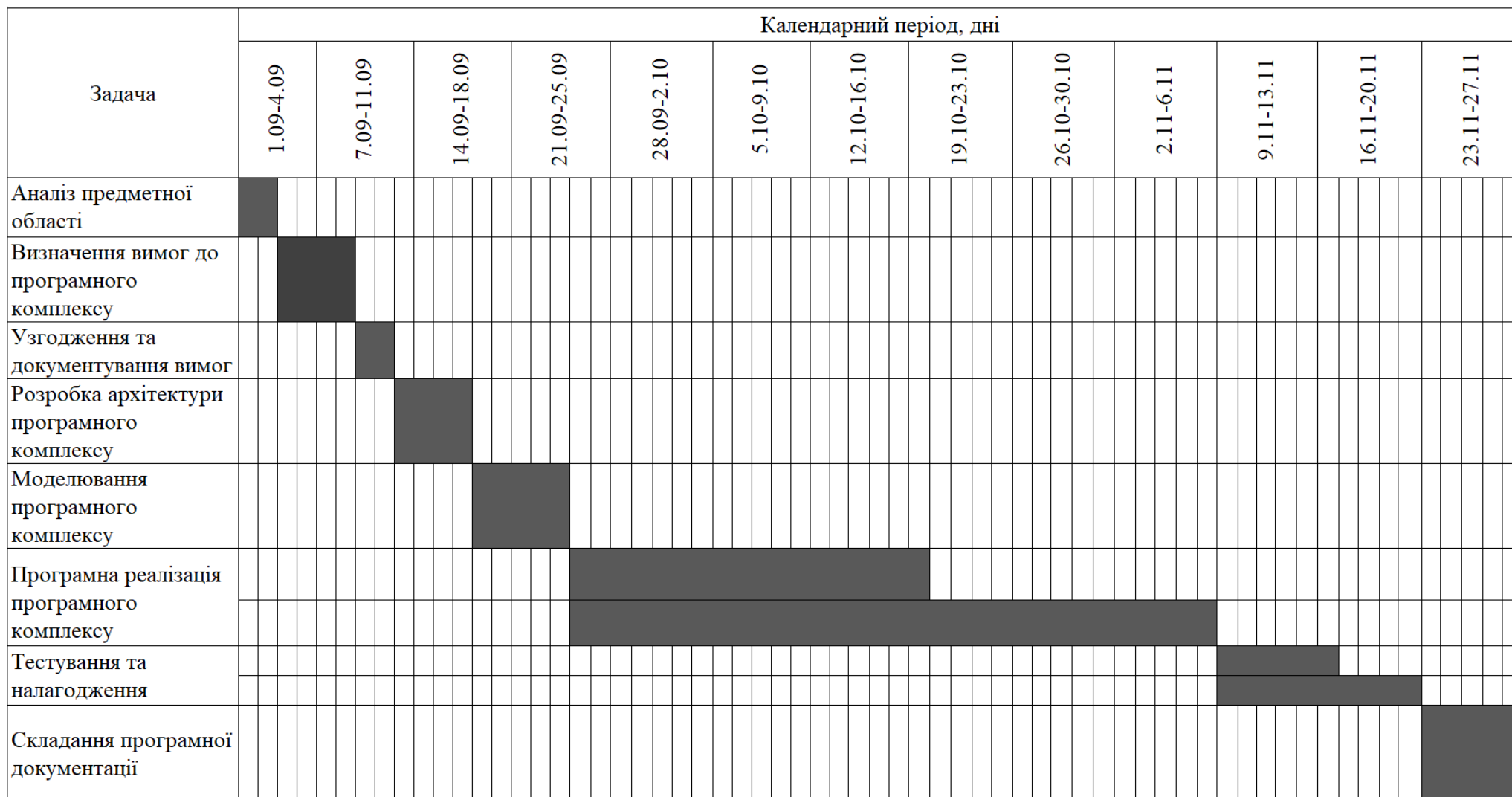


Рисунок 6.1 – Стрічковий графік

6.4 Визначення витрат на оплату праці

У таблиці 6.6 наведено основні витрати на заробітну плату працівникам, а також обчислено нарахування ЄСВ і додаткової заробітної плати (премії).

Таблиця 6.6 – Склад, чисельність та фонд заробітної плати персоналу

Посада	Кількість осіб	Посадовий оклад, грн	Преміальний відсоток до окладу, %	Сума премії, грн	Місячна заробітна плата, грн	ЄСВ (22%), грн
Програміст	2	16000	15 %	2400	16000	4048
Разом	2	32000		4800	32000	8096

6.5 Визначення витрат на матеріали

У проектуванні та розробці було використано наступні матеріали та найменування: папір А4 – 120 грн (1 упаковка), тонер для картриджа принтеру – 160 грн, віртуальна машина на сервісі DigitalOcean – 136 грн.

Для підрахунку загальних витрат на матеріали було використано формулу (6.1) [26]:

$$M = \sum_{i=1}^n (P_i \cdot N_i \cdot (1 + K_{т.з}) - P_{io} \cdot N_{io}), \quad (6.1)$$

де M – це загальні витрати на матеріали, грн.;

n – кількість найменувань матеріалів, напівфабрикатів і комплектуючих;

$K_{т.з}$ – коефіцієнт транспортно-заготівельних витрат, $K_{т.з} = 0,05$;

P_i – ціна i -го найменування або комплектуючого, грн.;

N_i – ціна зворотних відходів i -го найменування матеріалу, грн.;

P_{io} – ціна зворотних відходів i -го найменування матеріалу, грн, $P_{io} = 0$;

N_{io} – кількість зворотних відходів i -го найменування, $N_{io} = 0$.

Результати обчислень:

$$M = (1 + 0,05) \cdot (120 + 160 + 136) = 436,8 \text{ грн.}$$

6.6 Витрати на технічне обладнання

Персональні комп'ютери працівників мають наступну конфігурацію: Intel Core i5-8400, 16 ГБ RAM, 256 ГБ SSD, блок живлення потужністю 400 Вт, монітор Samsung C27R500FHIX, миша, клавіатура. Додаткова техніка: принтер. Середня ціна збірки ПК – 19000 грн, а середня ціна принтеру – 3500 грн.

Амортизаційні відрахування визначають за формулою (6.2):

$$A = \Phi_6 \cdot \frac{H_a}{100}, \quad (6.2)$$

де Φ_6 – балансова вартість обчислювальної техніки, грн., $\Phi_6 = 41500$ грн;

H_a – норма амортизаційних відрахувань на повне відновлення обчислювальної техніки, %, $H_a = 25\%$.

Підраховані амортизаційні відрахування:

$$A = 41500 \cdot 0,25 = 10375 \text{ грн.}$$

Щоб розрахувати витрати на користування обладнанням, було використано формулу (6.3):

$$C_e = N_n \cdot \Phi_{\text{еф}} \cdot K_{\text{зч}} \cdot K_{\text{зп}} \cdot C_e, \quad (6.3)$$

де N_n – номінальна потужність ЕОМ, кВт;

$\Phi_{\text{еф}}$ - річний ефективний фонд часу роботи ЕОМ;

$K_{зм}$ – середній коефіцієнт завантаження по часу;

$Ц_e$ – ціна одного кВт*год. електроенергії, грн./(кВт*год.).

Номінальна потужність ПК дорівнює 0,425 кВт. Загальна кількість часу роботи ПК за рік становить 1800 годин. Завантаження за часом – 0,8, а потужності – 0,6. Ціна однієї кВт години становить 1,63 грн.

Підраховані витрати на користування ПК:

$$C_e = 2 \cdot (0,425 \cdot 1800 \cdot 0,8 \cdot 0,6 \cdot 1,63) = 1197,08 \text{ грн.}$$

Заробітна плата обслуговуючого персоналу розраховується за формулою (6.4):

$$З_{обсл} = \Phi ЗП_p \cdot (1 + K_{від}) \cdot \frac{t_{обсл}}{\Phi_{еф.обсл}}, \quad (6.4)$$

де $\Phi ЗП_p$ – річний фонд заробітної плати (основної та додаткової) обслуговуючого персоналу, грн., $\Phi ЗП_p = 57600$ грн;

$K_{від}$ – коефіцієнт відрахувань на соціальне страхування та до інших фондів, %, $K_{від} = 22\%$;

$t_{обсл}$ – час протягом року, необхідний на технічне обслуговування, год/рік, $t_{обсл} = 12$ год/рік;

$\Phi_{еф.обсл}$ – річний ефективний фонд часу обслуговуючого персоналу, год/рік, $\Phi_{еф.обсл} = 1800$ год/рік.

Результати обчислень:

$$З_{обсл} = 57600 \cdot (1 + 0,22) \cdot \frac{12}{1800} = 468,48 \text{ грн.}$$

Витрати на «Поточний ремонт обладнання» становлять 3% від вартості придбаного обладнання і дорівнюють 1245 грн. «Інші витрати» становлять 5% від суми попередніх витрат на утримання та експлуатацію обладнання. Сума попередніх витрат дорівнює 13285,56 грн, 5% від якої становлять 664,28 грн.

Розрахунки витрат занесено до таблиці 6.7.

Таблиця 6.7 – Кошторис витрат на утримання та експлуатацію обладнання

Найменування витрати	Сума, грн.
Амортизація обладнання	10375
Експлуатація ПК (окрім витрат на поточний ремонт)	1197.08
Заробітна плата основна та додаткова обслуговуючих робітників з відрахуваннями на соціальні заходи	468,48
Поточний ремонт обладнання	1245
Інші витрати	664,28
Усього	13949,84

Витрати на оплату машинного часу ЕОМ (ПК) для розробки програмного комплексу було розраховано за формулою (6.5):

$$C_{\text{мч}} = V_{\text{екс}} \cdot t_{\text{мч}}, \quad (6.5)$$

де $V_{\text{екс}}$ – експлуатаційні витрати на одну годину машинного часу даної ЕОМ, грн/машино-год.;

$t_{\text{мч}}$ – машинний час, необхідний для розробки програмного комплексу, машино-год.

Експлуатаційні витрати на одну годину машинного часу використовуваної ЕОМ розраховують діленням суми витрат за даними кошторису «Витрати на утримання та експлуатацію устаткування (ЕОМ)» (табл. 6.8) на річний ефективний фонд часу роботи ЕОМ. Річний ефективний фонд часу роботи ЕОМ дорівнює 1800 годин. Результати обчислень:

$$V_{\text{екс}} = \frac{13949,84}{1800} = 7.75 \text{ грн/машино-час.}$$

ЕОМ експлуатують протягом 110 людино-днів, по 8 годин в день, що в сумі становить 880 годин. Результати обчислень витрат на машинний час:

$$C_{\text{мч}} = 7,75 \cdot 880 = 6820 \text{ грн.}$$

6.7 Розрахунок накладних витрат

До накладних витрат відносяться витрати на загальне управління, такі, як витрати на заробітні плати працівникам, додаткові канцелярські витрати, тощо. Накладні витрати додаються до вартості розробки програмного комплексу непрямым шляхом – у відсотках до основної заробітної плати працівників. В даному випадку накладні витрати становлять 40%, що дорівнює 12800 грн.

Результати обчислень занесено до таблиці 6.8.

Таблиця 6.8 – Витрати на розробку програмного комплексу

Стаття витрат	Сума, грн.	Питома вага до підсумку, %
Основна заробітна плата	32000	44,39
Додаткова заробітна плата (премія)	4800	6,66
ЄСВ	8096	11,24
Матеріали та комплектуючі	436,8	0,6
Витрати на утримання та обслуговування тех. обладнання	13949,84	19,35
Накладні витрати	12800	17,76
Усього	72082,64	100

6.8 Розрахунок економічної ефективності програмного продукту

Даний програмний комплекс орієнтований на продаж у вигляді цифрової ліцензії на користування.

Орієнтовано, кількість проданих ліцензій очікується в 800 од., з мінімальною вартістю в 280 грн за ліцензію.

Кошти, отримані від продажу ліцензій вважатимуться прибутком, якщо включити податок на прибуток (5%, спрощена система оподаткування для ТОВ).

Чистий прибуток розраховується за формулою (6.6):

$$S = N \cdot P - (N \cdot P \cdot T), \quad (6.6)$$

де S – чистий прибуток, грн.;

N – кількість проданих ліцензій на користування програмним комплексом;

P – ціна за одну ліцензію, грн.;

T – податок на прибуток, %.

Результати обчислень:

$$S = 800 \cdot 280 - (800 \cdot 280 \cdot 0,05) = 212800 \text{ грн.}$$

Коефіцієнт ефективності вкладених коштів в програмний комплекс розраховується за формулою (6.7):

$$E = \frac{S}{K}, \quad (6.7)$$

де S – чистий прибуток, грн.;

K – обсяг капітальних втрат, грн.

Результати розрахування коефіцієнту ефективності:

$$E = \frac{212800}{72082,64} = 2,95.$$

Термін окупності капітальних витрат на розробку розраховується за формулою (6.8):

$$T_{ок} = \frac{K}{S}, \quad (6.8)$$

де $T_{ок}$ – термін окупності, рік;

S – чистий прибуток, грн.;

K – обсяг капітальних витрат, грн.

Результати розрахування терміну окупності ефективності:

$$T_{ок} = \frac{72082,64}{212800} = 0,34 \text{ року.}$$

Таким чином економічна доцільність розробки програмного комплексу є допустимою, оскільки термін окупності витрат на розробку становить 0,34 року, що менше за допустиме значення (6,67 року).

6.9 Висновки за розділом 6

У даному розділі було описано головну ідею проєкту, стратегії та підходи з продажу розробленого програмного комплексу ERP, проаналізовано наявність попиту та розглянуто основну конкуренцію. Розглянувши потенційну цільову групу споживачів та перспективи розвитку проєкту було встановлено, що розроблений програмний комплекс є перспективною альтернативою популярних та об'ємних ERP систем.

7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

7.1 Аналіз потенційних небезпек

На сьогодні, будь-яка професія несе певний ризик здоров'ю фахівця, тому професії, які щоденно та тривало користуються сучасними досягненнями науки та техніки, а саме персональними електронно-обчислювальними машинами (персональними комп'ютерами, смартфонами, планшетами тощо) не стали виключенням.

Неправильна організація робочого місця матиме значні наслідки для працівника, так як розташування техніки, меблів та її ергономіки впливають скелетно-м'язову систему, що нерідко впливають у погіршення самопочуття, м'язових болях та погіршення осанки.

Засоби передачі інформації, зокрема монітори, а також недостатнє освітлення робочого місця впливають на органи зору фахівця через шкідливого електромагнітного та «синього» спектру випромінювання, що призводить до короткозорості.

Комп'ютерне обладнання також має властивість швидко запилюватись, що погано впливає на дихальну систему працівника.

Більшість ІТ-підприємств організовують робочі місця за принципами офісу відкритого типу, що нерідко призводить до надмірного шуму у приміщенні, які заважають фахівцю підтримувати концентрацію та знижують загальну працездатність.

Недотримання принципів чергування роботи та відпочинку приводить до загального погіршення самопочуття та підвищення психофізичного навантаження.

Засоби введення інформації, зокрема клавіатури та миші також швидко забруднюються, збираючи піт та сальні виділення, які сприяють поширенню мікроорганізмів.

Недотримання правил пожежної безпеки, відсутність плану виходу з приміщень у разі НС, відсутність засобів гасіння пожеж, або несправність електрообладнання несе ризик прямий ризик здоров'ю працівника [27].

Розроблюваний програмний комплекс ERP-системи передбачає роботу в офісному приміщенні відкритого типу командою програмістів з двох або більше людей.

Розміри приміщення $12 \times 28 \times 3.5$ м.

Площа приміщення становить $12 \times 28 = 336 \text{ м}^2$.

Об'єм приміщення - $12 \times 28 \times 3.5 = 1176 \text{ м}^3$.

7.2 Заходи із забезпечення безпеки

Робоче місце працівника ІТ-галузі оточене електроприладами, що несуть ризик ураження електричним струмом. За ступенем небезпеки приміщення відносяться до групи «без підвищеної небезпеки».

Заходи щодо забезпечення електробезпеки розроблено відповідно до ДСТУ 7237:2011 (Електробезпека.) і глави 1.7. ПУЕ-7 (Заземлення і захисні заходи електробезпеки) [28],[29].

Електроживлення обладнання електрообладнання приміщення живиться від мережі змінного струму з напругою 220 В і частотою 50 Гц. Комп'ютерне обладнання кожного робочого місця приєднується до мережі за допомогою мережевих фільтрів з функцією запобігання різкої зміни напруги. Середня потужність системного блоку персонального комп'ютеру становить 500-600 Вт. Корпуси електрообладнання виготовлені з неструмопровідних матеріалів або мають струмопровідне покриття. Усі прилади відносяться до І категорії захисту від ураження електричним струмом та мають захисне заземлення. Відкриті електрокабелі приладів однофазні, трьожильні, з подвійною ізоляцією.

Внутрішня електрична мережа приміщення розділена на підмережі та відсікається від зовнішньої за допомогою диференційного реле з номінальним струмом 40 А та номінальною напругою 230/400 В, та за допомогою реле

напруги з номінальним струмом 5 А та номінальною напругою 230/400 В у відношенні 1 реле напруги на 1 підмережу.

Згідно ДСТУ 7237:2011 в приміщенні встановлено автоматичні вимикачі, пристрій заземлення контуру.

Згідно з ПУЕ опір захисного заземлення у приміщенні становить 3 Ом (відповідно до ДСТУ 7237:2011 – не більше 4 Ом).

7.3 Заходи з забезпечення виробничої санітарії та гігієни праці

Заходи щодо забезпечення виробничої санітарії та гігієни праці для обчислювального центру обладнаного ПК з ВДТ (візуальними дисплейними терміналами) розроблені відповідно до вимог Державних санітарних норм та правил «Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу», МЮУ 06.05.2014 р. за № 472/25249, ДСанПіН 3.3.2.007-98 «Державні стандартні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» і НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин».

Мікрокліматичні умови та їх забезпечення (відповідно до вимог ДСН 3.3.6-042-99 «Санітарні норми мікроклімату виробничих приміщень» та ДБН В.2.5-67:2013 «Опалення, вентиляція та кондиціонування» були встановлені, зважаючи на тип роботи працівника у приміщенні, який відносять до категорії 1а - роботи, що виконуються сидячи і не потребують фізичного напруження, при яких витрати енергії складають менше 139 Вт [30],[31]. Оптимальні рівні параметрів наведені у таблиці 7.1.

Таблиця 7.1 - Мікрокліматичні норми у приміщенні

Період року	Категорія робіт	Температура повітря, °С	Відносна вологість, %	Швидкість руху повітря, м/с
Холодний	Легка Іа	22 – 24	60 – 40	0,1
Теплий	Легка Іа	23 – 25	60 – 40	0,1

Відповідно до ДБН В.2.5-67:2013 «Опалення, вентиляція та кондиціювання», для підтримання даних мікрокліматичних умов, приміщення забезпечене системою водяного опалення для забезпечення необхідної температури повітря у холодний період року. Для підтримки температури та вологості повітря у теплий період року, приміщення використовує центральну систему постачання та охолодження повітря, яка також має підсистему іонізації та зволоження повітря, яке поставляється у приміщення [32].

Відповідно до вимог ДБН В.2.5-28-2018 «Природне і штучне освітлення», тип освітлення у приміщенні суміщене. Коефіцієнт освітлення (КО) становить 2,5 %, що відповідає зоровій роботі середньої точності. Для забезпечення достатнього рівня освітлення, у приміщенні встановлені точкові світлодіодні масиви, об'єднані в лінійні лампи. Було розраховано кількість ламп необхідних для освітлення приміщення площею 336 м² освітленістю $E_n = 400$ лк (середня точність робіт та світлий/середній фон, коефіцієнт відбиття 70% стелі, та 50% стін) за допомогою методу світлового потоку. Рівняння кількості світильників розраховується за формулою (7.1):

$$N = \frac{E_n \cdot S \cdot k_z \cdot z}{n \cdot \phi \cdot \eta}, \quad (7.1)$$

де N – кількість ламп

E_n – нормована мінімальна освітленість, лк (люкс); $E_n = 400$ лк;

S – площа освітлюваного приміщення, м²; $S = 336$ м²;

z – відношення середньої освітленості до мінімальної; $z = 1,1$;

k_3 – коефіцієнт запасу; $k_3 = 1,5$ (для світлодіодних ламп);

n – кількість ламп у світильнику; $n = 2$;

ϕ – світловий потік однієї лампи; $\phi = 2000$ лм (люмен), лампа потужністю 20 Вт;

η – коефіцієнт використання світлового потоку, що характеризується коефіцієнтами відбиття стелі та стін, який можна знайти за допомогою індексу приміщення.

Рівняння індексу приміщення розраховується за формулою (7.2):

$$i = \frac{A \cdot B}{h \cdot (A + B)}, \quad (7.2)$$

де A - довжина приміщення, м; $A = 28$ м;

B - ширина приміщення, м; $B = 12$ м;

h - висота приміщення, м; $h = 3,5$ м.

Розрахований індекс:

$$i = \frac{28 \cdot 12}{(3,5 - 0,7) \cdot (28 + 12)} = 3.$$

Підставивши індекс до таблички, зазначеної у ДБН В.2.5-28-2018, отримано $\eta = 68$ %.

Розрахована кількість ламп:

$$N = \frac{400 \cdot 1,1 \cdot 1,5 \cdot 336}{2000 \cdot 0,68} \approx 78 \text{ шт.}$$

При розташуванні світильників у 3 ряди вибираємо 39 світильників, розміщених по 13 у кожному ряді.

Згідно до вимог ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку», приміщення під визначення «Приміщення, обладнані персональними комп'ютерами та/або технікою для бізнесу», рівень

шуму якої не перевищує 50 дБА. Заміри фонового шуму у приміщенні були проведені по кутах та в центрі приміщення. Визначений середній фоновий рівень шуму у приміщенні – 43 дБА [33],[34].

В приміщенні наявні 3 медичні аптечки першої допомоги.

7.4 Заходи безпеки у надзвичайних ситуаціях

7.4.1 Заходи з пожежної безпеки

Робота над цим підрозділом проводилась відповідно до вимог ППБУ «Правила пожежної безпеки України» (НАПБА.01.001-2014).

Пожежі в приміщенні становлять особливу небезпеку, тому що пов'язані з великими матеріальними втратами. Як відомо, пожежа може виникнути при взаємодії горючих речовин, окислення і джерел запалювання. У приміщеннях присутні всі три основні чинники, необхідні для виникнення пожежі.

Для відводу теплоти від ПК у приміщенні постійно діє потужна система кондиціонування. Тому кисень, як окислювач процесів горіння, є в будь-якій точці обчислювального центру.

Приміщення, згідно ДСТУ Б В.1.1-36:2016 «Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою» відноситься до категорії «В» (пожежно-небезпечна), а клас можливої пожежі, згідно ДБН В.1.1-7:2016 «Пожежна безпека об'єктів будівництва. Загальні вимоги», визначається, як «А» [35].

До засобів гасіння пожежі, призначених для локалізації невеликих загорянь, відносяться пожежні стовбури, внутрішні пожежні водопроводи, вогнегасники, сухий пісок, азбестові ковдри тощо.

Оскільки приміщення що обладнане ПК з ВДТ має площу 324 м², тому відповідно до вимог п. 3.8 розділу «Типові норми належності вогнегасників» ДСТУ 4297:2004 «Пожежна техніка. Технічне обслуговування вогнегасників. Загальні технічні вимоги» для гасіння електроустановок, що знаходяться під напругою, передбачені вуглекислотні вогнегасники типу ВВК-3,5 у кількості

16 штук (з розрахунку один вогнегасник з величиною заряду вогнегасної речовини 3 кг і більше, на 20 м² площі приміщення). Відстань між вогнегасниками та місцями можливих загорянь не перевищує 10 м.

Усі працівники робочого приміщення під час прийняття на роботу і в процесі праці повинні проходити протипожежний інструктаж, перевірку знань з питань пожежної безпеки. Евакуаційні виходи і проходи утримуються постійно вільними та нічим не заблоковані.

Відповідальний за протипожежний стан приміщень повинен періодично (не рідше одного разу на місяць) перевіряти працездатність пристрою, який забезпечує автоматичне вимкнення системи вентиляції у разі пожежі, а також вогнедимозатримувальних пристроїв. Під час ремонту слід не допускати руйнування або порушення цілісності негорючих діафрагм у фальш-підлоговому просторі.

Фільтри припливно-витяжної вентиляції слід чистити згідно з затвердженим графіком, але не рідше двох разів на рік. Роботи з ремонту вузлів (блоків) ЕОМ, як правило, проводити не дозволяється. Їх необхідно проводити в окремому приміщенні (майстерні). У разі необхідності проведення ремонту або технічного обслуговування ЕОМ безпосередньо в залі допускається мати не більше 0,5 л легкозаймистих рідин у небиткій, щільно закритій тарі. Для промивання деталей необхідно застосовувати, як правило, негорючі миючі препарати.

Відповідальний за пожежну безпеку приміщень зобов'язаний негайно сповістити технічну службу у разі виявлення незадовільного технічного стану димових та теплових сповіщувачів автоматичної пожежної сигналізації, а також про несправності автоматичної установки об'ємного (газового) пожежогасіння [36].

Усі працівники повинні вміти привести в дію (в разі потреби) первинні засоби пожежогасіння. Регулярно, але не рідше одного разу на квартал, слід очищати від пилу ЕОМ, контрольно-вимірювальну апаратуру, кабельні траншеї та фальшпідлоговий простір за допомогою пиловсмоктувача.

У приміщеннях забороняється:

- залишати без нагляду електричну апаратуру, що використовується;
- зберігати постійно в залах ЕОМ носії інформації, запасні блоки та деталі в кількості більшій, ніж необхідно для поточного використання;
- користуватися груповими розетками на горючій панелі;
- використовувати килими та доріжки із синтетичних матеріалів;
- ставити на вікна глухі ґрати (в разі потреби встановлюються ґрати, що розкриваються зсередини);
- застосовувати електронагрівальні побутові прилади;
- курити та застосовувати відкритий вогонь (газоелектрозварювальні роботи проводяться за наявності спеціального дозволу адміністрації у встановленому порядку).

Після закінчення робочого часу всі робочі місця мають бути прибрані, з приміщень необхідно видалити горючі відходи та вимкнути всі струмоприймачі.

У разі виявлення ознак пожежі працівник, який їх помітив, повинен:

- негайно повідомити про це службу порятунку за номером телефону – 101 (112). При цьому необхідно назвати адресу об'єкта, вказати кількість поверхів будівлі, місце виникнення пожежі, обстановку на пожежі, наявність людей, а також повідомити своє прізвище;
- вжити заходів щодо евакуації людей та матеріальних цінностей;
- вжити заходів до гасіння пожежі з використанням наявних вогнегасників та інших засобів пожежогасіння.

Керівник підрозділу, якого повідомлено про пожежу, зобов'язаний:

- перевірити, чи викликані пожежно-рятувальні підрозділи;
- вимкнути у разі необхідності струмоприймачі та вентиляцію;
- за умови загрози життю людей негайно організувати їх рятування (евакуацію), вивести за межі небезпечної зони всіх осіб, не пов'язані із ліквідацією пожежі;
- перевірити чи оповіщено людей про пожежу;

- забезпечити дотримання техніки безпеки працівниками, які беруть участь у гасінні пожежі;
- організувати зустріч підрозділів Державної пожежної охорони, надати їм допомогу під час локалізації та ліквідації пожежі;
- після прибуття на пожежу пожежно-рятувальних підрозділів забезпечити безперешкодний доступ їх до місця виникнення пожежі.

7.4.2 Заходи з цивільного захисту

До систем протипожежного захисту належать технічні засоби, які розташовані на об'єкті та використовуються для запобігання, ліквідації та локалізації пожеж. Ці засоби допомагають захищати людей, довкілля та матеріальні цінності від пожежі.

У будівлях, приміщеннях та спорудах мають бути обладнані системи протипожежного захисту (СПЗ). Це передбачено Правилами пожежної безпеки в Україні, затвердженими наказом МВС від 30.12.2014 № 1417 (пункт 1.2 глави 1 розділу V). Питання, пов'язані з обладнанням об'єктів СПЗ та підтриманням їхньої експлуатаційної придатності, врегульовані ДБН В.2.5-56:2014 «Інженерне обладнання будинків і споруд. Системи протипожежного захисту» та іншими нормативно-правовими актами з питань пожежної безпеки.

До систем протипожежного захисту відносять:

- системи протидимного захисту;
- системи пожежної сигналізації;
- автоматичні системи пожежогасіння;
- автономні системи пожежогасіння локального застосування;
- системи диспетчеризації СПЗ;
- системи оповіщення про пожежу та управління евакуацією людей.

Системи пожежної сигналізації призначені для раннього виявлення пожежі та подання сигналу тривоги для вжиття необхідних заходів.

При побудові, проєктуванні, монтуванні, перевірці відповідності і технічному обслуговуванні систем пожежної сигналізації необхідно користуватися вимогами ДСТУ-Р CEN/TS 54-12. Вибирати тип димового сповіщувача рекомендується згідно з ДСТУ EN 54-7 та ДСТУ EN 54-12 з урахуванням чутливості до різних типів димів. Пожежні сповіщувачі повинні використовуватися згідно з вимогами експлуатаційних документів та з урахуванням середовища контрольованих приміщень. Допускається встановлювати в одному приміщенні один пожежний сповіщувач за умов виконання усіх вимог. При проєктуванні контролю приміщень пожежними сповіщувачами їх рекомендується розташовувати за схемою трикутного (див. рис. 7.1) або квадратного розміщення (див. рис. 7.2) (де, a – відстань між сповіщувачами; b – відстань від стіни до сповіщувача) [37].

Системи оповіщення про пожежу та управління евакуюванням людей призначена для оповіщення людей, що перебувають в будинку (споруді), про виникнення пожежі з метою створення умов для їх своєчасного евакуювання.

Експлуатування та технічне обслуговування систем оповіщення в будинках та спорудах слід здійснювати згідно з вимогами НАПБ А.01.003 та інших чинних нормативних документів.

При проєктуванні систем оповіщення необхідно користуватися даними про акустичні та планувальні характеристики приміщень будинку, акустичні та світлові характеристики оповіщувачів, що плануються для використання. Правильний вибір оповіщувачів, їх кількість та правильність розміщення є надзвичайно важливим для досягнення якісного та своєчасного сприйняття переданої системою оповіщення інформації людьми, що перебувають у будинку.

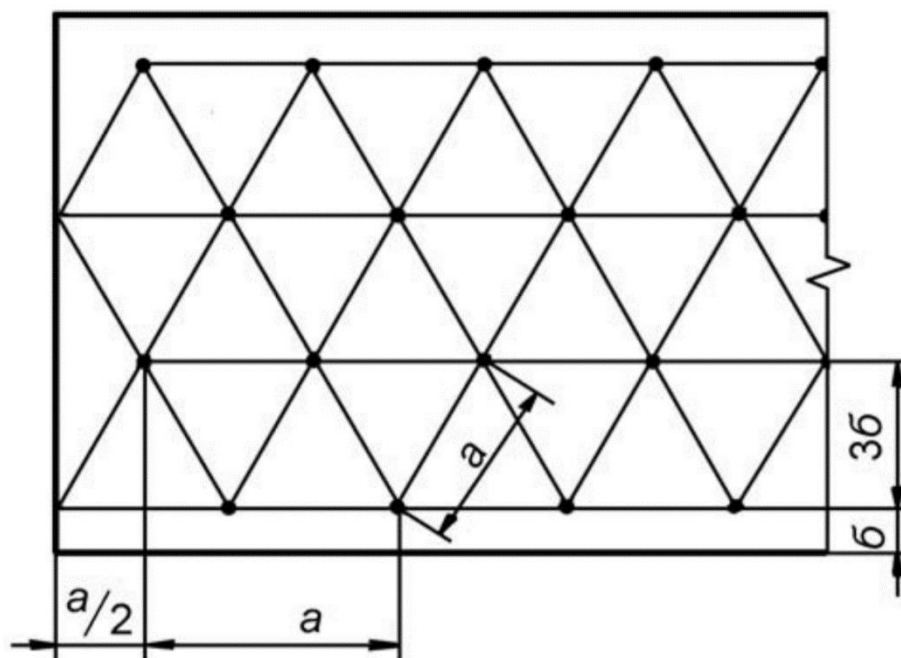


Рисунок 7.1 – Схема трикутного розміщення сповіщувачів [37]

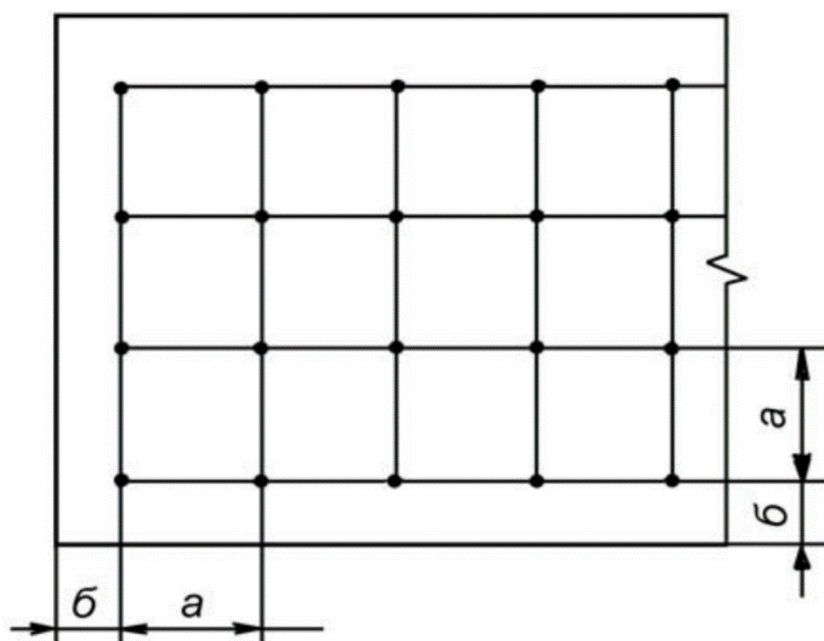


Рисунок 7.2 – Схема квадратного розміщення сповіщувачів [37]

Настінні звукові оповіщувачи та гучномовці необхідно встановлювати так, щоб відстань між верхньою частиною оповіщувача і рівнем підлоги було не менше 2,2 м, а відстань між верхньою частиною оповіщувача та рівнем стелі була не менше 0,15 м.

Системи протидимного захисту передбачають з метою досягнення таких цілей:

- забезпечення умов для безпечного евакуювання;
- забезпечення умов для гасіння пожежі та проведення пожежно-рятувальних робіт;
- зниження ймовірності займання предметів, обладнання, речовин і матеріалів під впливом теплового випромінювання.

Повітроводи і вентиляційні канали системи протидимного захисту повинні мати клас вогнестійкості не нижче ніж:

- EI 180 – у висотних житлових і громадських будинках для транзитних повітроводів і шахт;
- EI 150 – для транзитних повітроводів і шахт за межами обслугованого протипожежного відсіку;
- EI 45 – для вертикальних повітроводів в каналів у межах обслугованого протипожежного відсіку.
- EI 30 – у решті випадків у межах обслугованого протипожежного відсіку.

Диспетчеризація системами протипожежного захисту та автоматизація систем протипожежного захисту забезпечує контроль, сигналізацію і електрокерування роботою систем згідно з вимогами.

Диспетчеризація та автоматизація систем протипожежного захисту встановлюється на об'єктах. Де передбачені системи оповіщення людей про пожежу та управління евакуюванням 4-го та 5-го типу, у будинках громадського призначення умовною висотою 47 м.

При проєктуванні слід передбачати його інтеграцію в автоматизовану систему моніторингу та управління ДСТУ-Н В.2.5-37. За цієї інтеграції для здійснення моніторингу та передачі інформації щодо систем протипожежного захисту необхідно використовувати спеціалізоване програмне забезпечення інтегрованої системи АСМУ або відкритий стандартизований інтерфейс.

У приміщенні пожежного поста повинен бути встановлений ППКП, на якому відображається світлова і звукова сигналізація згідно з вимогами ДСТУ EN 54-2.

У приміщенні пожежного поста повинна бути:

а) світлова і звукова сигналізація про:

- 1) несправність у системі оповіщення;
- 2) несправність електроживлення систем оповіщення;
- 3) несправність ланцюгів пуску системи оповіщення;
- 4) відключення автоматичного поста;

б) світлова сигналізація:

- 1) спрацювання (з розшифруванням зон оповіщення);

в) модулі:

- 1) запуску системи оповіщення (з вибиранням зон оповіщення);
- 2) запуску системи оповіщення (для всього будинку);
- 3) зупинка системи оповіщення (з вибиранням зон оповіщення);
- 4) зупинка системи оповіщення (з вибиранням зон оповіщення)

[37],[38].

ВИСНОВКИ

З метою підвищення ефективності менеджменту ресурсів та оптимізації робочого процесу управлінського персоналу на ІТ підприємстві, було реалізовано програмний комплекс для підвищення якості фінансового аудиту, підвищення ефективності менеджменту ресурсів та оптимізації робочого процесу управлінського персоналу на ІТ підприємстві, були вирішені наступні завдання:

- проведено огляд ERP-системи;
- досліджено та проаналізовано основні архітектури ERP-систем;
- розглянуто основні методи представлення знань;
- формалізовано знання про предметну область «Робочий процес підприємства»;
- розроблено моделі програмного забезпечення ERP-системи;
- зпроектувати власну архітектуру ERP-системи на основі формалізованих знань про предметну область;
- розроблено програмний комплекс ERP-системи на основі зпроектованих моделей та архітектури;
- розгорнуто у хмарному середовищі розроблений програмний комплекс ERP-системи;
- експериментально досліджено та протестувано розроблений програмний комплекс ERP-системи.

Розроблений програмний комплекс має велике практичне значення та рекомендується до використання в ІТ підприємствах для балансування та оптимізації ресурсів, а також підвищення загального показника продуктивності.

Наукова новизна результатів дослідження полягає у формалізації знань про предметну область робочого процесу на ІТ підприємстві, на підставі яких було зпроектовано архітектуру програмного комплексу ERP-системи та змодельовано комплекс моделей програмного забезпечення, що поліпшує процес розробки.

Розроблений програмний комплекс дозволяє підвищити загальний виробничий показник у ІТ підприємстві шляхом оптимізації робочого процесу та підвищення продуктивності управлінського персоналу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Leon, A. Enterprise Resource Planning / A. Leon – New Dehli: McGraw-Hill, 2008. – P. 224–500.
2. O'Leary D. Enterprise resource planning systems / D. O'Leary – Cambridge University Press, 2000. – 232 p.
3. Wylie L. A Vision of Next Generation MRP II / L. Wylie – Gartner Scenario, 1990. – P. 300-339.
4. A Delphi examination of public sector ERP implementation issues / [eds.: S. Chang, G. Gable, E. Smythe, G. Timbrell]. – Proceedings of the XXI international conference on Information systems. – Atlanta: ACM, 2000. – P. 494–500.
5. Bell S. ERP, CRM, PLM working together / S.Bell // Lean Enterprise Systems. – N. Y.: McGraw-Hill, 2006. – P. 242–296.
6. О'Лири Д. ERP системы. Современное планирование и управление ресурсами предприятия / Д. О'Лири. – М. : Изд-во "Вершина", 2004. – 272 с.
7. Планування ресурсів підприємства [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Планування_планування_ресурсів.
8. Meer K. H. Best Practices in ERP Software Applications. / K. H. Meer – Lincoln, NE: iUniverse, 2005. – 232 с.
9. Бабенчук С. П. Методологии внедрения ERP-систем и управления проектами. Подходы к оценке эффективности внедрения ERP / С. П. Бабенчук. // Открытое образование. 2011. № 6. – С.51–59.
10. Бунтова О.Г. Введение в ERP-системы: учебное пособие. / О.Г. Бунтова – УГУ им. А.М. Горького, 2007.
11. Гаврилова, Т. А. Базы знаний интеллектуальных систем / Т. А. Гаврилова, В. Г. Хорошевский. – СПб.: Питер, 2000. – 384 с.
12. Логіка першого порядку [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Логіка_першого_порядку.
13. Мариничева М.К.: Управление знаниями на 100%: Путеводитель для практиков / М.К. Мариничева. – М. : Альпина Бизнес Букс, 2008. – 320 с.

14. Методи представлення знань [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Методи_представлення_знань.

15. Субботін С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень : навчальний посібник. / С.О. Субботін. – Запоріжжя : ЗНТУ, 2008. – 341 с.

16. Ланде Д.В. Модель розповсюдження інформації з урахуванням поняття сприйняття і пам'яті / Д.В. Ланде, В.А. Додонов // Авіаційна та екстремальна психологія у контексті технологічних досягнень: збірник наукових праць. – К. : Аграр Медіа Груп, 2017. – С. 148-153

17. Коваль А.В., Зайцева Е.А., Бойко Ю.Д.: Верифікація комп'ютерної моделі системи інформаційного управління // Збірник "Вісник НТУУ "КПІ". Інформатика, управління та обчислювальна техніка". – 2014. – № 2. – С.26.

18. Куперштейн Л.М. Технологія створення програмних та інтелектуальних систем / Л.М. Куперштейн // конспект лекції. – Вінниця: ВФЕУ, 2011.

19. JavaScript [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/JavaScript>.

20. Node.js [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Node.js>.

21. React [Электрон. ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/React>.

22. Laravel [Электрон. ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Laravel>.

23. PhpStorm [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/PhpStorm>.

24. REST [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/REST>.

25. Остапенко В.В. Економіка та організація виробництва програмних продуктів [Текст] / В.В. Остапенко, Т.А. Колпакова. Запоріжжя: ЗНТУ. – 2012. – 132 С.

26. Методичні рекомендації щодо виконання економічного розділу дипломного проекту здобувачами вищої освіти технічних спеціальностей за освітнім ступенем «магістр» [Текст] / Уклали Т.О. Пожусьва, – Запоріжжя: ЗНТУ, 2020 – 13 с.

27. Керб Л.П. Основи охорони праці. / Л.П. Керб. – Київ : КНЕУ, 2006. 216 с.

28. ДСТУ 7237:2011. Система стандартів безпеки праці. Електробезпека. Загальні вимоги та номенклатура видів захисту: наказ Держспоживстандарту України від 2 лютого 2011 р. № 37 // Держспоживстандарт України. – 2011. – 2 лютого.

29. Правила улаштування електроустановок: наказ Міненерговугілля України від 21.07.2017 № 476 // Міненерговугілля України. – 2017. – 21 липня.

30. Коробко О.В. Охорона праці та безпека у надзвичайних ситуаціях / методичні вказівки. Укл.: О.В. Коробко – Запоріжжя: ЗНТУ, 2018 – 27с.

31. Санітарні норми мікроклімату виробничих приміщень [Електронний ресурс] : ДСН 3.3.6.042-99. – Чинний від 1999-12-01. – К. : МОЗ України, 1999. – URL: <http://zakon2.rada.gov.ua/rada/show/va042282-99>. – (Державні санітарні норми)

32. Опалення, вентиляція та кондиціонування [Текст] : ДБН В.2.5-67:2013. – На заміну СНиП 2.04.05-91 ; крім розділу 5 та додатка 22. ; чинний від 2014-01-01. – К. : Мінрегіонбуд України, 2013. – 149 с. – (Державні будівельні норми України)

33. Санітарні норми виробничого шуму, ультразвуку та інфразвуку, [Електронн. ресурс] : ДСН 3.3.6.037-99. – Чинний від 1999-12-01. – К. : МОЗ України, 1999.– URL: <http://zakon2.rada.gov.ua/rada/show/va037282-99>. – (Державні санітарні норми).

34. Кучерявий В. О. Охорона праці. / В.О. Кучерявий. – Львів: Ороїяна-Нова, 2007. 368 с.

35. Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою [Текст] : ДСТУ Б В.1.1-36:2016.

– На заміну НАПБ Б.03.002-2007 ; чинний від 2017-01-01. – К. : Мінрегіонбуд України, 2016. – 66 с. – (Національний стандарт України)

36. Кузнєцов В. А. Пожежна безпека. / В.А. Кузнєцов – Харків : Фактор, 2008. 575 с.

37. Системи протипожежного захисту: ДБН В.2.5-56:2014. – [Чинний від 2015-07-01]. – К.: Мінрегіон України, 2015. – 134 с. – (Державні будівельні норми України).

38. Протипожежні заходи // Термінологічний словник-довідник з будівництва та архітектури / Р. А. Шмиг, В. М. Боярчук, І. М. Добрянський, В. М. Барабаш ; за заг. ред. Р. А. Шмига. — Львів, 2010. — С. 164.

ДОДАТОК А**Код програми**

A.1 Основний код серверного додатку

A.1.1 Клас ApiCollection

```

<?php

namespace App\Base;

use App\Types\ResponseMode;
use App\User;
use
Illuminate\Database\Eloquent\Builder;
use
Illuminate\Database\Eloquent\Collection;
use Illuminate\Http\Request;
use
Illuminate\Http\Resources\Json\ResourceC
ollection;
use Illuminate\Support\Facades\Auth;

class      ApiCollection      extends
ResourceCollection
{
    /**
     * @var User $user;
     */
    protected $user;

    public      function
__construct($resource)
    {
        $this->user = Auth::user() ?:
(object) ['role' => null];
        parent::__construct($this-
>processResource($resource));
    }

    /**
     * If query builder passed, adjust
response by role and inbound filters

```

```

*
* @param $resource
* @return Collection
*/
protected      function
processResource($resource)
    {
        if
(in_array(get_class($resource), [
'Illuminate\Database\Eloquent\Builder',
'Illuminate\Database\Eloquent\Relations\
BelongsTo',
'Illuminate\Database\Eloquent\Relations\
BelongsToMany',
'Illuminate\Database\Eloquent\Relations\
HasOne',
'Illuminate\Database\Eloquent\Relations\
HasMany',
'Illuminate\Database\Eloquent\Relations\
HasManyThrough',
]))) {
            if (Auth::user()) $resource =
$this->adjustByPermissions($resource);
            $resource = $this-
>sortAndFilter($resource, \request()-
>get());
        }

        return $resource;
    }

/**

```

```

* Paginate API request
*
* @param Request $request
* @return array
*/
protected function paginate(Request
$request)
{
    $skip = $request->get('skip') ??
0;
    $limit = $request->get('limit')
?? 10;

    /** @var Collection $collection
*/
    $collection = $this->collection;

    return $collection->skip($skip)-
>take($limit)->values()->all();
}

/**
* Transform the resource collection
into an array.
*
* @param $request
* @return array
*/
public function toArray($request)
{
    $mode = $request->get('mode');
    $data = ($mode &&
ResponseMode::MINIMAL === $mode ||
$request->get('pagination') == 'false')
? parent::toArray($request)
: $this->paginate($request);

    return [
        'data' => $data,
        'total' => $this-
>collection->count(),

```

```

        'filters' =>
static::filters(),
    ];
}

/**
* For conditional usage
* Copy of the toArray method
*
* @param $request
* @return array
*/
protected function
prepareArray($request)
{
    $mode = $request->get('mode');
    $data = ($mode &&
ResponseMode::MINIMAL === $mode)
? parent::toArray($request)
: $this->paginate($request);

    return [
        'data' => $data,
        'total' => $this-
>collection->count(),
        'filters' =>
static::filters(),
    ];
}

/* Methods below must be re-defined
at target collection class in order to
match your needs */

/**
* Adjust response by user role
*
* @param $builder
* @return mixed
*/

```

```

        protected
adjustByPermissions($builder)
    {
        return $builder;
    }

    /**
     * Filter your response by request
parameters
     *
     * @param $resource
     * @param Request $request
     * @return Builder
     */
    protected
sortAndFilter($resource,
$request)
    {
        return $resource;
    }

    /**
     * Array of allowed filters that will
be used on front-end
     *
     * @return array
     */
    public static function filters()
    {
        return [];
    }
}

```

A.1.2 Клас ApiResource

```

<?php

namespace App\Base;

use App\Types\ResponseMode;
use App\User;

```

```

use Illuminate\Http\Request;
use
Illuminate\Http\Resources\Json\JsonResou
rce;
use Illuminate\Support\Facades\Auth;

class ApiResource extends JsonResource
{
    /**
     * @var User $user
     */
    protected $user;

    public
function
__construct($resource)
    {
        $this->user = Auth::user() ?:
(object) ['role' => null];
        parent::__construct($resource);
    }

    /**
     * @param $request
     * @return array
     */
    protected
function
originalArray($request)
    {
        return
parent::toArray($request);
    }

    /**
     * Transform the resource into an
array
     *
     * @param $request
     * @return array
     */
    public function toArray($request)
    {

```

```

        $mode = $request->get('mode');
        if ($mode && method_exists($this,
$mode)) {
            $array =
call_user_func(array($this, $mode),
$request);
        } else {
            $array = method_exists($this,
ResponseMode::FULL)
                ?
call_user_func(array($this,
ResponseMode::FULL), $request)
                :
parent::toArray($request);
        }
        return $this->adjustByPermissions($this->withFields($array, $request));
    }

```

```

/**
 * Adjust final payload by user
permissions
 *
 * @param $payload
 * @return mixed
 */

```

```

protected function
adjustByPermissions($payload)
{
    return $payload;
}

```

```

/**
 * Additional fields to be included
 *
 * @param $payload
 * @param Request $request
 * @return mixed
 */

```

```

protected function
withFields($payload, Request $request)
{
    return $payload;
}

```

A.1.3 Клас Client

```
<?php

namespace App;

use
Illuminate\Database\Eloquent\Factories\H
asFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Facades\Log;

class Client extends Model
{
    use HasFactory;

    protected $fillable = [
        'first_name',      'last_name',
        'email',    'origin',    'zip',    'city',
        'country', 'phone', 'street', 'fields',
        'bill_to',
        'company'
    ];

    protected $appends = [
        'name'
    ];

    public function getNameAttribute()
    {
        return trim($this->first_name . '
'. $this->last_name);
    }

    public function photos()
    {
        return $this-
>belongsToMany(Photo::class);
    }

    public function projects()
```

```

    {
        return $this-
>hasOne(Project::class);
    }

    public function payments()
    {
        return $this->hasManyThrough(
            Payment::class,
            Project::class,
            'client_id',
            'project_id',
            'id',
            'id'
        );
    }

    public function
getFullAddressAttribute()
    {
        $address = $this-
>attributes['city'] . ', ' .
config('constants.countries.' . $this-
>attributes['country']);

        if ($this->attributes['street'])
            $address = $this->attributes['street'] .
            ', ' . $address;

        if ($this->attributes['zip'])
            $address = $address . ', ' . $this-
>attributes['zip'];

        return $address;
    }
}
```

Клас InvoiceGenerator

```
<?php

namespace App\Helpers;

use App\Account;
```

```

use
App\Exceptions\InvoiceGenerationException;
use App\Project;
use App\Types\InvoiceTaxType;
use Carbon\Carbon;
use GuzzleHttp\Client as GuzzleClient;
use Illuminate\Support\Facades\Log;
use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Str;

```

A.1.4 Клас InvoiceGenerator

```

{
    const URL = 'https://invoice-
generator.com';

    const DEFAULT_OPTIONS = [
        'logo' =>
'https://dygit.io/assets/img/logo.svg',
        'quantity_header' => 'Hours',
        'unit_cost_header' => 'Rate/Hour
($)',
        'number' => '02',
        'currency' => 'usd',
    ];

    const TAX_TYPE_MAP = [
        InvoiceTaxType::FLAT => true,
        InvoiceTaxType::PERCENT => '%'
    ];

    protected static function getClient()
    {
        return new GuzzleClient([
            'base_uri' => static::URL,
            'verify' =>
config('app.env') !== 'local'
        ]);
    }
}

```

```

/**
 * @param Project $project
 * @param Account $account
 * @param $items
 * @param array $options
 * @return string
 * @throws InvoiceGenerationException
 */
public static function
generate(Project $project, Account
$account, $items, $options = [])
{
    $data = array_merge([
        'currency' =>
$options['currency'] ??
self::DEFAULT_OPTIONS['currency'],
        'due_date' =>
$options['due_date'] ?? Carbon::now()-
>addDays(2)->toDateString(),
        'date' =>
$options['date'] ?? Carbon::now()-
>toDateString(),
        'quantity_header' =>
$options['quantity_header'] ??
self::DEFAULT_OPTIONS['quantity_header']
    ,
        'unit_cost_header' =>
$options['unit_cost_header'] ??
self::DEFAULT_OPTIONS['unit_cost_header']
    ],
        'number' =>
$options['number'] ?? $account
->payments()
        -
>where('project_id', '=', $project->id)
->count() + 1,
        'from' =>
$options['from'] ?? $account->user->name
. PHP_EOL . $account->user->email,

```

```

        'to' =>
$options['to'] ?? $project->client->name
. PHP_EOL . $project->client->email,
        'notes' =>
$options['notes'] ?? 'Kindly make a
payment at your earliest convenience.' .
PHP_EOL .
        'Please do not hesitate
to contact me with any questions.' .
PHP_EOL .
        PHP_EOL .
        'Many thanks, ' . PHP_EOL
.
        $account->user->name,
        'items' => array_map(function
($item) { return [
                'name' =>
$item['description'],
                'description' =>
$item['description'],
                'quantity' =>
$item['time'],
                'unit_cost' =>
$item['rate'],
            ]; }, $items),
        'fields' => [
                'tax' =>
!empty($options['tax']) &&
$options['tax']
                ?
self::TAX_TYPE_MAP[$options['tax_type']]
                : false,
            ],
        'tax_title' =>
$options['tax_title'] ?? 'Tax',
        'tax' =>
$options['tax_amount'] ?? 0
    ]);

    try {

```

```

        $guzzle =
static::getClient();
        $response = $guzzle->
post('/', ['form_params' => $data]);
        $path = 'invoices/' .
'invoice-' . Str::random(6) . '.pdf';
        Storage::disk('public')->
put($path, $response->getBody());
        return
Storage::disk('public')->path($path);
    } catch
(\GuzzleHttp\Exception\GuzzleException
$exception) {
        Log::error($exception);
        throw new
InvoiceGenerationException;
    }
}

```

A.1.5 PermissionManager

Клас

```
<?php

namespace App\Helpers;

use App\Project;
use App\Types\Permissions;
use Illuminate\Support\Facades\Log;
use
Spatie\Permission\Exceptions\PermissionD
oesNotExist;
use Spatie\Permission\Models\Permission;

class PermissionManager
{
    // to create wildcard permissions
    const MANAGER_USER = [
        Permissions::ADD_USER_WORKTIME,

Permissions::ADD_USER_RAISES_PROJECT,

        Permissions::VIEW_USERS,
        Permissions::VIEW_USER_WORKTIME,
        Permissions::VIEW_USER_PROJECTS,

Permissions::VIEW_USER_RAISES_PROJECT,
    ];

    const MANAGER_CLIENT = [

Permissions::VIEW_CLIENT_PROJECTS,
    ];

    private static function
grantAccess($user, $permissionName)
    {
        try {
```

```
        $user->
>givePermissionTo($permissionName);
        } catch (PermissionDoesNotExist
$exception) {
            $permission =
Permission::create([
                'name' =>
$permissionName,
                'guard_name' => 'web'
            ]);
            $user->
>givePermissionTo($permission);
        }

        public static function
syncPermissionsByProject(Project
$project)
        {
            if (count($project->developers) >
0) {
                $usersId = implode(',',
$project->developers()->get()-
>pluck('id')->toArray());
                $clientId = $project->client-
>id;
                $managerUser =
array_map(function ($item) use ($usersId)
{ return "$item.$usersId"; },
self::MANAGER_USER);
                $managerClient =
array_map(function ($item) use
($clientId) { return "$item.$clientId";
}, self::MANAGER_CLIENT);

                foreach ($project->managers
as $manager) {

                    self::grantAccess($manager,
Permissions::VIEW_PROJECTS . '.' .
$project->id);
```

```

self::grantAccess($manager,
Permissions::VIEW_CLIENTS . '.' .
$clientId);

        foreach ($managerUser as
$permissionName) {

self::grantAccess($manager,
$permissionName);
        }
        foreach ($managerClient
as $permissionName) {

self::grantAccess($manager,
$permissionName);
        }
    }
}

    public static function
wildcard($permission, $id)
    {
        return "$permission.$id";
    }
}

```

A.1.6 Клас Project

```

<?php

namespace App;

use App\Helpers\Helper;
use App\Types\UserRole;
use
Illuminate\Database\Eloquent\Factories\H
asFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Carbon;

```

```

class Project extends Model
{
    use HasFactory;

    public $timestamps = false;

    protected $fillable = [
        'title', 'status', 'start_date',
'end_date'
    ];

    public function tags()
    {
        return $this-
>belongsToMany(Tag::class);
    }

    public function users()
    {
        return $this-
>belongsToMany(User::class)
        ->using(ProjectUser::class)
        ->withPivot([
            'account_id',
            'start_date',
            'end_date',
            'rate',
            'salary_based',
            'weekly_limit',
            'status',
            'role',
        ]));
    }

    public function accounts()
    {
        $ids = [];
        foreach ($this->users()-
>cursor() as $user) {

```

```

        if ($user->pivot-
>account_id) $ids[] = $user->pivot-
>account_id;
    }
    return Account::whereIn('id',
$ids);
}

public function client()
{
    return $this-
>belongsTo(Client::class);
}

public function developers()
{
    return $this-
>belongsToMany(User::class)
        ->using(ProjectUser::class)
        ->wherePivot('role',
UserRole::DEVELOPER)
        ->withPivot(['start_date',
'end_date', 'rate', 'salary_based',
'weekly_limit', 'account_id']);
}

public function managers()
{
    return $this-
>belongsToMany(User::class)
        ->using(ProjectUser::class)
        ->wherePivot('role',
UserRole::MANAGER)
        ->withPivot(['start_date',
'end_date']);
}

public function photos()
{
    return $this-
>belongsToMany(Photo::class);

```

```

}

public function raises()
{
    return $this-
>hasMany(Raise::class)-
>orderBy('starting_from');
}

public function worktime()
{
    return $this-
>hasMany(Worktime::class);
}

/**
 * @param integer $userId
 * @return array
 */
public function calculateProfitsByUser(int $userId)
{
    $pivot = $this->pivot ?:
(Helper::findInArray($userId, 'id',
$this->developers))[ 'pivot' ];
    $worktimeMapped = [];
    $totalWorktime = 0;
    $now = Carbon::now();

    $worktimeGrouped = $this-
>worktime()
        ->where('user_id', $userId)
        ->selectRaw("SUM(time) as
time, MONTH(date) as month, YEAR(date) as
year")
        ->groupBy('year', 'month')
        ->get()
        ->toArray();
    $worktimeThisWeek = $this-
>worktime()->where('user_id', $userId)

```

```

        ->whereDate('date',      '>=',
$now->clone()->startOfWeek())
        ->whereDate('date',      '<=',
$now->clone()->endOfWeek())
        ->sum('time');
    // calculate total and monthly
    worktime and earnings
    $totalEarned = 0;
    $rate = $pivot->rate;
    $raises     = $this->raises()-
>where('user_id',      $userId)->get()-
>toArray();
    $raise = null;

    foreach ($worktimeGrouped as
$item) {
        if (!$raise && count($raises)
> 0) $raise = array_shift($raises);
        $date =
Carbon::createFromDate($item['year'],
$item['month'], 1); // start of month by
default
        $time = round($item['time'],
2);

        if ($raise &&
Carbon::parse($raise['starting_from'])-
>isBetween($date->toDate(), $date-
>clone()->endOfMonth()->toDate())) {
            $raiseDate =
Carbon::parse($raise['starting_from']);
            $firstPeriodDiff = $this-
>worktime()->where('user_id', $userId)
                ->whereDate('date',
'>=', $date)
                ->whereDate('date',
'<', $raiseDate)
                ->sum('time');
            $firstPeriodAmount =
$firstPeriodDiff ? $firstPeriodDiff *
$rate : 0;

```

```

        $rate +=
$raise['amount'];

        $secondPeriodDiff =
$this->worktime()->where('user_id',
$userId)
            ->whereDate('date',
'>=', $raiseDate)
            ->whereDate('date',
'<', $date->clone()->endOfMonth())
            ->sum('time');
        $secondPeriodAmount =
$secondPeriodDiff ? $secondPeriodDiff *
$rate : 0;

        $total =
$firstPeriodAmount + $secondPeriodAmount;
    } else {
        $total = $pivot-
>salary_based ? $rate : $rate * $time;
    }

    $totalWorktime += $time;
    $totalEarned += $total;
    $worktimeMapped[$date-
>firstOfMonth()->toDateString()] =
['time' => $time, 'earned' => $total];
}

    return [
        'total_worktime' =>
round($totalWorktime, 2),
        'worktime_this_month' =>
$worktimeMapped[$now->firstOfMonth()-
>toDateString()][ 'time' ] ?? 0,
        'worktime_this_week' =>
round($worktimeThisWeek, 2),
        'total_earned' =>
round($totalEarned),

```

```

        'earned_this_month' =>
round(($worktimeMapped[$now-
>firstOfMonth()-
>toDateString()][ 'earned' ] ?? 0), 2),
    ];
}

public function
calculateOverallProfits()
{
    $totals = [
        'total_worktime' => 0,
        'worktime_this_month' => 0,
        'worktime_this_week' => 0,
        'total_earned' => 0,
        'earned_this_month' => 0,
        'avg_rate' => 0,
        'team_count' => $this-
>users()->count()
    ];
    foreach ($this->developers()-
>cursor() as $developer) {
        $totals['avg_rate'] +=
$developer->pivot->rate;
        $result = $this-
>calculateProfitsByUser($developer->id);
        foreach ($result as $key =>
$value) {
            $totals[$key] = $value +
($totals[$key] ?? 0);
        }
    }
    $totals['avg_rate'] =
round($totals['avg_rate'] / $this-
>developers()->count(), 2);
    return $totals;
}

public function
getFieldsAttribute($value)
{

```

A.1.7 Клас User

```
<?php

namespace App;

use App\Helpers\Helper;
use App\Http\Resources\PhotoCollection;
use App\Types\RaiseType;
use
Illuminate\Database\Eloquent\Factories\H
asFactory;
use Illuminate\Support\Carbon;
use Carbon\CarbonPeriod;
use Illuminate\Foundation\Auth\User as
Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;
use
Spatie\Permission\Traits\HasPermissions;
use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable
{
    use      Notifiable,      HasApiTokens,
    HasFactory, HasRoles, HasPermissions;

    /**
     * The attributes that are mass
    assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password',
    'rate',
    ];

    /**
     * The attributes that should be
    hidden for arrays.

```

```

    *
    * @var array
    */
    protected $hidden = [
        'password', 'remember_token',
    'roles', 'permission_names'
    ];

    /**
     * The attributes that should be cast
    to native types.
     *
     * @var array
     */
    protected $casts = [
        'verified_at' => 'datetime',
        'created_at' => 'datetime',
    ];

    public function projects()
    {
        return $this-
    >belongsToMany(Project::class)
        ->using(ProjectUser::class)
        ->withPivot(['start_date',
    'end_date', 'rate', 'salary_based',
    'weekly_limit', 'status']);
    }

    public function accounts()
    {
        return $this-
    >hasMany(Account::class);
    }

    public function raises()
    {
        return $this-
    >hasMany(Raise::class);
    }

```

```

    public function worktime()
    {
        return $this->hasMany(Worktime::class);
    }

    public function payments()
    {
        return $this->hasManyThrough(
            Payment::class,
            ProjectUser::class,
            'user_id',
            'account_id',
            'id',
            'account_id',
        );
    }

    public function photos()
    {
        return $this->belongsToMany(Photo::class);
    }

    public function getRateAttribute($val)
    {
        return $val + array_reduce(
            $this->raises()
                ->where('type',
                    RaiseType::PERSONAL)
                -
                >whereDate('starting_from', '<=',
                    Carbon::now()->toDateString())
                ->get()
                ->toArray(),
            function ($all, $item) {
                return
                    $item['amount'] + $all;
            },
            0
        );
    }

    public function getPermissionNamesAttribute()
    {
        return $this->getAllPermissions()->pluck('name');
    }

    public function getSalaryAllTimeAttribute()
    {
        $raises = $this->raises()
            ->where('type',
                RaiseType::PERSONAL)
            ->whereDate('starting_from',
                '<=', Carbon::now()->toDateString())
            ->get()
            ->toArray();

        $val = 0;
        $rate = $this->attributes['rate']; // get original
        rate, not calculated

        $period =
            CarbonPeriod::create($this->attributes['created_at'], '1 month',
                Carbon::now()->toDateTimeString());

        $raise = null;
        foreach ($period as $date) {
            if (!$raise && count($raises)
                > 0) $raise = array_shift($raises);
            if ($raise &&
                Carbon::parse($raise['starting_from'])-
                >isBetween($date->firstOfMonth()-
                >toDate(), $date->lastOfMonth()-
                >toDate())) {
                $rate +=
                    $raise['amount'];
                $raise = null;
            }
        }
    }

```

```

        $val += $rate;
    }
    return $val;
}

public function getAvatarAttribute()
{
    $photos = new
    PhotoCollection($this->photos);
    return $photos->count() ?
    $photos[0] : null;
}

/**
 * array['total_worktime'] Total
worktime on all projects
 * array['worktime_this_month'] How
much worked this month on all projects
 * array['worktime_this_week'] How
much worked this week on all projects
 * array['total_earned'] How much
funds earned on the projects overall
 * array['earned_this_month'] How
much funds earned on the projects this
month
 * array['salary_this_month'] How
much salary earned on this month (actual)
 *
 * @return array
 */
public function
calculateOverallProfits()
{
    $totals = [
        'total_worktime' => 0,
        'worktime_this_month' => 0,
        'worktime_this_week' => 0,
        'total_earned' => 0,
        'earned_this_month' => 0,
    ];

```

```

        foreach ($this->projects as
        $project) {
            $result = $project-
            >calculateProfitsByUser($this-
            >attributes['id']);
            foreach ($result as $key =>
            $value) {
                $totals[$key] = $value +
                ($totals[$key] ?? 0);
            }
        }
        $totals['salary_this_month'] =
        round(($this->rate /
        Helper::hoursCapByMonth() *
        ($totals['worktime_this_month'] ?? 0));
        return $totals;
    }

    public function
    hasAccessTo($instanceId, $rule)
    {
        $can = $this-
        >can("{ $rule }. { $instanceId }");
        $isBlocked = $this-
        >permissions()->where('name', '=',
        "!$rule.$instanceId")->exists();
        return $can && !$isBlocked;
    }

    public function
    hasRelated($permissionName)
    {
        return $this->permissions()-
        >where('name', 'like',
        "$permissionName.%")->exists();
    }

    public function
    getAllowedIds($permissionName)
    {

```

```

        $permissions = $this->permissions()
            ->where('name', 'like',
                "$permissionName.%")
            ->cursor();

        $ids = [];
        foreach ($permissions as
            $permission) {
            $ids[] = explode('.',
                $permission->name)[2];
        }
        return $ids;
    }
}

```

A.1.8 Клас Worktime

```

<?php

namespace App;

use
Illuminate\Database\Eloquent\Factories\H
asFactory;
use Illuminate\Database\Eloquent\Model;

class Worktime extends Model

```

```

{
    use HasFactory;

    public $timestamps = false;

    protected $table = 'worktime';

    protected $fillable = [
        'time', 'date'
    ];

    public function project()
    {
        return $this->belongsTo(Project::class);
    }

    public function user()
    {
        return $this->belongsTo(User::class);
    }
}

```

A.2 Основний код веб-додатку

A.2.1 Компонент AddUser

```

import { useCallback, useState, useRef }
from 'react';
import { Card, message, PageHeader } from
'antd';
import { useDispatch } from 'react-
redux';
import { useRouter } from 'next/router';

```

```

import styles from './Users.module.scss';
import UserForm from
'./components/UserForm';
import { parseErrors } from
'../../utils';
import { addUser } from
'../../store/users/actions';

const AddUser = () => {
    const dispatch = useDispatch();
    const { back } = useRouter();

```

```

const form = useRef(null);
const [submitting, setSubmitting] =
useState(false);

const handleSubmit = useCallback(async
() => {
  try {
    const values = await
form.current.validateFields();
    setSubmitting(true);
    const { error, data } = await
dispatch(addUser(values));
    if (error?.response?.data?.errors)
{

form.current.setFields(parseErrors(error
.response.data.errors));
    } else if (error) {
      message.error(error.message);
    } else {
      form.current.resetFields();
      message.success(data.message);
    }
  } catch (e) {
    // validation failed
  } finally {
    setSubmitting(false);
  }
}, [form, dispatch]);

return (
  <>
    <PageHeader
      className={styles.pageHeader}
      title='Add User'
      onBack={back}
    />
    <Card
className={styles.container}>
      <UserForm
        onSubmit={handleSubmit}
        submitting={submitting}
        formRef={form}
      />
    </Card>
  </>
);
};

export default AddUser;

```

A.2.2 КОМПОНЕНТ

AuthorizedLayout

```
import React, { useState, useEffect } from
'react';
import PropTypes from 'prop-types';
import classNames from 'classnames';
import { Layout as AntLayout } from
'antd';
import { MenuUnfoldOutlined,
MenuFoldOutlined } from '@ant-
design/icons';
import { useDispatch, useSelector } from
'react-redux';

import styles from
'../Layout.module.scss';
import Head from '../Head';
import Sidebar from './Sidebar';
import AvatarArea from './AvatarArea';
import { signedUserSelector } from
'../../../../../store/auth/selectors';
import { fetchSignedUser } from
'../../../../../store/auth/actions';

const AuthorizedLayout = ({ title,
children }) => {
  const dispatch = useDispatch();
  const [user] =
useSelector(signedUserSelector);
  const [sidebarCollapsed, toggle] =
useState(false);

  useEffect(() => {
    if (!user)
dispatch(fetchSignedUser());
  }, [user, dispatch]);

  return (
    <>
```

```
      <Head title={title} />
      <AntLayout
        className={styles.container}>
        <Sidebar
          collapsed={sidebarCollapsed} />
        <AntLayout className='site-
layout'>
          <AntLayout.Header
            className={classNames(styles.background,
styles.header)}>
            {React.createElement(sidebarCollapsed ?
MenuUnfoldOutlined : MenuFoldOutlined, {
              className: styles.trigger,
              onClick: () =>
toggle((state) => !state),
            })}
          <AvatarArea />
        </AntLayout.Header>
        <AntLayout.Content
          style={{
            margin: '24px 16px',
            minHeight: 'unset',
          }}
        >
          { children }
        </AntLayout.Content>
      </AntLayout>
    </>
  );
};

AuthorizedLayout.propTypes = {
  title: PropTypes.string,
  children: PropTypes.any.isRequired,
};

AuthorizedLayout.defaultProps = {
  title: 'dygit',
};
```

```
export default AuthorizedLayout;
```

A.2.3 Компонент Can

```
import { useMemo, useCallback } from
'react';
import PropTypes from 'prop-types';
import { useSelector } from 'react-
redux';
```

```
import { signedUserSelector } from
'../../store/auth/selectors';
```

```
function
retrieveIdsFromPermission(permission) {
  const [, , ids] =
permission.split('.');
  let array = null;
  if (ids) array = ids.split(',');
  return array;
}
```

```
const Can = ({
  perform, yes, no, loading,
}) => {
  const [user] =
useSelector(signedUserSelector);
```

```
  const check = useCallback((permission,
type) => {
    if (user) {
      if (type === 'except') return
user.permissions.includes(permission);
      if
(user.permissions.includes('*')) return
true;
      if (permission === '*') return
true;
      const ids =
retrieveIdsFromPermission(permission);
      if (ids) {
```

```
        const [entity, action] =
permission.split('.');
        const hasAccessIds = [
          ...(
            new Set(
              user.permissions
                .filter((item) => {
                  const [e, a, i] =
item.split('.');
                  return i && e ===
entity && a === action;
                })
            )
          ]
        );
```

```
        .map(retrieveIdsFromPermission)
          .flat(),
        )
      ],
    ];
    for (let i = 0; i < ids.length; i
+= 1) {
      if
(!hasAccessIds.includes(ids[i])) return
false;
    }
    return true;
  }
  return
user.permissions.includes(permission);
}
return false;
}, [user]);
```

```
  const canPerform = useMemo(() => {
    if (typeof perform === 'object') {
      return
Object.keys(perform).reduce((all, key) =>
{
        const checks =
perform[key].map((i) => check(i, key));
        const results =
checks.filter((item) => item);
```

```

        switch (key) {
            case 'all': return all &&
results.length === checks.length;
            case 'any': return all &&
results.length > 0;
            case 'except': return all &&
results.length === 0;
            default: return all;
        }
    }, true);
}
return check(perform);
}, [perform, check]);

if (!user) return loading || null;

return canPerform ? yes : no;
};

Can.propTypes = {
    perform:
PropTypes.oneOfType([PropTypes.string,
PropTypes.shape({
    all:
PropTypes.arrayOf(PropTypes.string),
    any:
PropTypes.arrayOf(PropTypes.string),
    except:
PropTypes.arrayOf(PropTypes.string),
}]]).isRequired,
    yes: PropTypes.any.isRequired,
    no: PropTypes.any,
    loading: PropTypes.element,
};

Can.defaultProps = {
    no: null,
    checkType: 'all',
};

export default Can;

```

A.2.4 КОМПОНЕНТ

GuardedLink

```
import Link from 'next/link';
import PropTypes from 'prop-types';
import { Spin } from 'antd';

import Can from '../Can';

const GuardedLink = ({
  href, as, label, gate, children,
  hideIfFailed, loading,
}) => (
  loading ? <Spin size='small' style={{
marginTop: '5px' }} />
    : (
      <Can
        perform={gate}
        yes={()
          <Link href={href} as={as}>
            { /* eslint-disable-next-line
jsx-ally/anchor-is-valid */ }
          <a>{label || children}</a>
        </Link>
        }
        no={hideIfFailed ? null : (label
|| children)}
      />
    )
  );

GuardedLink.propTypes = {
  href: PropTypes.string.isRequired,
  label:
PropTypes.oneOfType([PropTypes.string,
PropTypes.number]),
  gate:
PropTypes.oneOfType([PropTypes.string,
PropTypes.shape({
```

```
all:
PropTypes.arrayOf(PropTypes.string),
  any:
PropTypes.arrayOf(PropTypes.string),
  except:
PropTypes.arrayOf(PropTypes.string),
  })).isRequired,
  as: PropTypes.string,
  children: PropTypes.any,
  hideIfFailed: PropTypes.bool,
  loading: PropTypes.bool,
});

GuardedLink.defaultProps = {
  as: '',
  children: null,
  label: '',
  hideIfFailed: false,
  loading: false,
};

export default GuardedLink;
```

A.2.5 Компонент Sidebar

```
import { useMemo, useCallback } from
'react';
import PropTypes from 'prop-types';
import Link from 'next/link';
import { Layout as AntLayout, Menu } from
'antd';
import {
  MailOutlined, DeploymentUnitOutlined,
  UnorderedListOutlined, TeamOutlined,
  SolutionOutlined, DollarCircleOutlined,
  BankOutlined, WalletOutlined,
  ContactsOutlined, HourglassOutlined,
  ArrowUpOutlined,
} from '@ant-design/icons';
import { useRouter } from 'next/router';

import { useSelector } from 'react-
redux';
import styles from
'../Layout.module.scss';
import { USER_ROLE } from
'../../../../../utils/constants';
import { signedUserSelector } from
'../../../../../store/auth/selectors';

const LINKS = {
  ACCOUNTS: {
    title: 'Accounts',
    icon: ContactsOutlined,
    route: '/accounts',
  },
  USERS: {
    title: 'Users',
    icon: TeamOutlined,
    items: [
      { title: 'List', icon:
UnorderedListOutlined, route: '/users' },
      { title: 'Invitations', icon:
MailOutlined, route: '/invitations' },
```

```
    ],
  },
  USERS_LIST: { title: 'Users', icon:
TeamOutlined, route: '/users' },
  PROJECTS: { title: 'Projects', icon:
DeploymentUnitOutlined, route:
'/projects' },
  CLIENTS: { title: 'Clients', icon:
SolutionOutlined, route: '/clients' },
  BUDGET: {
    title: 'Budget',
    icon: BankOutlined,
    items: [
      { title: 'Payments', route:
'/payments', icon: DollarCircleOutlined
},
      { title: 'Expenses', route:
'/expenses', icon: WalletOutlined },
    ],
  },
  WORKTIME: {
    title: 'Worktime',
    icon: HourglassOutlined,
    route: '/worktime',
  },
  RAISES: {
    title: 'Raises',
    icon: ArrowUpOutlined,
    route: '/raises',
  },
};

const LINKS_MAP = {
  [USER_ROLE.ADMIN]: [
    LINKS.USERS,
    LINKS.ACCOUNTS,
    LINKS.PROJECTS,
    LINKS.CLIENTS,
    LINKS.WORKTIME,
    LINKS.RAISES,
    LINKS.BUDGET,
```

```

],
[USER_ROLE.MANAGER]: [
  LINKS.USERS_LIST,
  LINKS.ACCOUNTS,
  LINKS.PROJECTS,
  LINKS.CLIENTS,
  LINKS.RAISES,
],
];

const Sidebar = ({ collapsed }) => {
  const router = useRouter();
  const [user] = useSelector(signedUserSelector);

  const [openedKey, selectedKey] = useMemo(
    () => {
      let tree = [];
      const rec = (link) => {
        if (link.items) {
          tree.push(link.title);
          return !!link.items.find(rec);
        }
        const res = link.route ===
router.route;
        if (res) tree.push(link.title);
        return res;
      };
      let arr = [];
      if
(user?.roles?.includes(USER_ROLE.ADMIN))
arr = LINKS_MAP[USER_ROLE.ADMIN];
      else if
(user?.roles?.includes(USER_ROLE.MANAGER
)) arr = LINKS_MAP[USER_ROLE.MANAGER];
      for (let i = 0; i < arr.length; i
+= 1) {
        if (rec(arr[i])) {

```

```

          return [tree.slice(0,
tree.length - 1).join('-'), tree.join('-
')]);
        }
        tree = [];
      }
      return [null, null];
    },
    [router, user],
  );

  const renderMenuItems =
useCallback((link, idx, tree = []) => {
    const key = tree.length
      ? `${tree.join('-')}-${
link.title}`
      : link.title;

    if (link.items) {
      return (
        <Menu.SubMenu title={link.title}
icon={<link.icon />} key={key}>
          {link.items.map((item, index)
=> renderMenuItems(item, index, [...tree,
idx]))}
        </Menu.SubMenu>
      );
    }

    return (
      <Menu.Item icon={<link.icon />}
key={key}>
        <Link href={link.route}>
          {/* eslint-disable-next-line
jsx-a11y/anchor-is-valid */}
          <a{ link.title }</a>
        </Link>
      </Menu.Item>
    );
  }, []);

```

```

const links = useMemo(() => {
  let array = [];
  if
(user?.roles?.includes(USER_ROLE.ADMIN))
array = LINKS_MAP[USER_ROLE.ADMIN];
  else if
(user?.roles?.includes(USER_ROLE.ADMIN))
array = LINKS_MAP[USER_ROLE.MANAGER];
  return array.map((item, idx) =>
renderMenuItems(item, idx));
}, [renderMenuItems, user]);

return (
  <AntLayout.Sider      trigger={null}
collapsible collapsed={collapsed}>
    <div className={styles.logo} />
    <Menu
      theme='dark'
      mode='inline'

defaultSelectedKeys={[selectedKey]}
      defaultOpenKeys={[openedKey]}
      selectedKeys={[selectedKey]}
    >
      { links }
    </Menu>
  </AntLayout.Sider>
);
};

Sidebar.propTypes = {
  collapsed: PropTypes.bool,
};

Sidebar.defaultProps = {
  collapsed: false,
};

export default Sidebar;

```

A.2.6 Компонент UserForm

```
import { useRef, useEffect, useMemo } from
'react';
import {
  Form, Input, Button, Col, Row, Select,
  Upload, Tag, InputNumber,
} from 'antd';
import { UploadOutlined } from '@ant-
design/icons';
import { useDispatch, useSelector } from
'react-redux';
import PropTypes from 'prop-types';

import          styles          from
'../Users.module.scss';
import          usePhotos        from
'../../utils/hooks/usePhoto';
import {
  URLs,  BASE_URL,  USER_STATUS_COLORS,
  USER_ROLE_COLORS,
} from ' ../../utils/constants';
import { getXsrfToken, normFile, ucFirst
} from ' ../../utils';
import          {          fetchUserRoles,
fetchUserStatuses          }          from
' ../../store/users/actions';
import          {          userRolesSelector,
userStatusesSelector          }          from
' ../../store/users/selectors';

const UserForm = ({
  onSubmit, submitting, initialValues,
formRef, ...props
}) => {
  const dispatch = useDispatch();
  const uploader = useRef(null);
  const [form] = Form.useForm();
  const [uploadPhoto, deletePhoto] =
usePhotos(form, uploader);
```

```
const [statuses, statusesLoading] =
useSelector(userStatusesSelector);
const [roles, rolesLoading] =
useSelector(userRolesSelector);

const statusOptions = useMemo(() =>
statuses.map((s, idx) => (
  <Select.Option key={idx.toString()}
value={s}>
    <Tag color={USER_STATUS_COLORS[s]
|| 'cyan'}>{ucFirst(s)}</Tag>
  </Select.Option>
)), [statuses]);

const rolesOptions = useMemo(() =>
roles.map((r, idx) => (
  <Select.Option key={idx.toString()}
value={r}>
    <Tag color={USER_ROLE_COLORS[r] ||
'cyan'}>{ucFirst(r)}</Tag>
  </Select.Option>
)), [roles]);

useEffect(() => {
  dispatch(fetchUserStatuses());
  dispatch(fetchUserRoles());
}, [dispatch]);

useEffect(() => {
  form.resetFields();
}, [form, initialValues]);

useEffect(() => {
  // eslint-disable-next-line no-param-
reassign
  formRef.current = form;
}, [form, formRef]);

return (
  <Form
    form={form}
```

```

    layout='vertical'
    initialValues={initialValues}
    {...props}
  >
    <Row gutter={10}>
      <Col span={12}>
        <Form.Item
          label='Name'
          name='name'
          rules={[
            { required: true, message:
'Name is required' },
          ]}
        >
          <Input placeholder="User's
full name" />
        </Form.Item>
        <Form.Item
          label='Email'
          name='email'
          rules={[
            { required: true, message:
'User email is required' },
            { type: 'email', message:
'Provide valid email' },
          ]}
        >
          <Input type='email'
placeholder='Email' />
        </Form.Item>
        <Form.Item
          label='Rate'
          name='rate'
          rules={[
            { required: true, message:
'Rate is required' },
          ]}
        >
          <InputNumber
placeholder='Monthly salary rate'
className={styles.fullWidth} />

```

```

      </Form.Item>
    </Col>
    <Col span={12}>
      <Form.Item
        label='Status'
        name='status'
        rules={[
          { required: true, message:
'User status is required' },
        ]}
      >
        <Select
          placeholder='Status'
          loading={statusesLoading}
        >
          { statusOptions }
        </Select>
      </Form.Item>
      <Form.Item
        label='Roles'
        name='roles'
        rules={[
          { required: true, message:
'User role is required' },
        ]}
      >
        <Select
          placeholder='Roles'
          loading={rolesLoading}
          mode='multiple'
        >
          { rolesOptions }
        </Select>
      </Form.Item>
    </Col>
  </Row>
  <Row gutter={10}>
    <Col span={12}>
      <Form.Item
        name='photos'
        label='Photo'

```

```

        valuePropName='fileList'
        getValueFromEvent={normFile}
      >
      <Upload
        action={BASE_URL +
URLS.PHOTOS}
        ref={uploader}
        accept='image/*'
        name='photo'
        listType='picture'

className={styles.uploadWrap}
        withCredentials
        multiple
        headers={{
          'X-XSRF-TOKEN':
getXsrfToken(),
        }}
        onRemove={deletePhoto}
      >
        <Button block
className={styles.upload}>
          <UploadOutlined />
          Click to upload
        </Button>
      </Upload>
    </Form.Item>
  </Col>
  <Col span={12}>
    <Input.Search
      placeholder='Or paste the
link here'
      enterButton='Upload'
      onSearch={uploadPhoto}

className={styles.uploadInput}
    />
  </Col>
</Row>
<Row gutter={10}>
  <Col span={24}>

```

```

    <Form.Item noStyle>
      <Button
        type='primary'
        htmlType='submit'
        className={styles.submit}
        onClick={onSubmit}
        loading={submitting}
      >
        Submit
      </Button>
    </Form.Item>
  </Col>
</Row>
</Form>
);
};

UserForm.propTypes = {
  initialValues: PropTypes.object,
  submitting: PropTypes.bool,
  onSubmit: PropTypes.func,
  formRef: PropTypes.object,
};

UserForm.defaultProps = {
  initialValues: {},
  submitting: false,
  onSubmit: () => {},
  formRef: null,
};

export default UserForm;

```

A.2.7 Компонент UserProfile

```

/* eslint-disable react/prop-types */
import { useEffect, useMemo, useCallback } from 'react';
import {
  PageHeader, Row, Col, Card, Avatar,
  Typography, Tag, Skeleton, Button, Tabs,
  Tooltip,
} from 'antd';
import { useRouter } from 'next/router';
import { useDispatch, useSelector } from 'react-redux';
import {
  ProjectOutlined, DollarOutlined,
  ToTopOutlined, MailOutlined,
  ClockCircleOutlined,
} from '@ant-design/icons';

import styles from './UserProfile.module.scss';
import Can from '../../common/Can';
import Profits from './Profits';
import Time from './Time';
import UserRaises from './Raises';
import UserProjects from './Projects';
import UserPayments from './Payments';
import UserCalendar from './Calendar';
import UserWorktime from './Worktime';
import GuardedLink from '../../common/GuardedLink';
import EntityAccessMiddleware from '../../common/EntityAccessMiddleware';
import SuspenseLoader from '../../common/SuspenseLoader';
import { fetchUser, clearUsers } from '../../store/users/actions';
import { userSelector } from '../../store/users/selectors';
import { formatCurrency, wildcard } from '../../utils';

```

```

import { signedUserSelector } from
  '../../../store/auth/selectors';
import {
  USER_ROLE, USER_ROLE_COLORS,
  USER_STATUS_COLORS, PERMISSION,
} from '../../../utils/constants';

const CARD_STYLE = {
  headStyle: {
    borderBottom: 'none',
    padding: '0 12px',
  },
  bodyStyle: {
    padding: '0 12px 12px 12px',
    display: 'flex',
    flexDirection: 'column',
  },
};

const TABS = {
  PROJECTS: {
    title: 'Projects',
    icon: ProjectOutlined,
    component: UserProjects,
  },
  PAYMENTS: {
    title: 'Payments',
    icon: DollarOutlined,
    component: UserPayments,
  },
  RAISES: {
    title: 'Raises',
    icon: ToTopOutlined,
    component: UserRaises,
  },
  WORKTIME: {
    title: 'Worktime',
    icon: ClockCircleOutlined,
    component: UserWorktime,
  },
};

```

```
const MAP = {
  [USER_ROLE.ADMIN]:
Object.values(TABS),
  [USER_ROLE.MANAGER]: [TABS.PROJECTS,
TABS.RAISES, TABS.WORKTIME],
};
```

```
const Loader = ({ spanSize = 12 }) => (
  <Col span={spanSize}>
    <Card className={styles.block}
{...CARD_STYLE}>
      <Skeleton loading active
className={styles.fullWidth} />
    </Card>
  </Col>
);
```

```
const UserProfile = () => {
  const dispatch = useDispatch();
  const { back, query } = useRouter();
  const [signedUser] =
useSelector(signedUserSelector);
  const [user, loading, response] =
useSelector(userSelector);
```

```
  useEffect(() => {
    if (query.id)
dispatch(fetchUser(query.id));
```

```
    return () => dispatch(clearUsers());
  }, [dispatch, query]);
```

```
const renderTab = useCallback((tab,
idx) => (
  <Tabs.TabPane
    key={idx}
    tab={{
      <span>
        <tab.icon />
        { tab.title }
      </span>
    }}
  />
```

```
    </span>
  )}
  >
    <tab.component />
  </Tabs.TabPane>
), []]);
```

```
const tabs = useMemo(() => {
  const keys = Object.keys(MAP);
  for (let i = 0; i < keys.length; i +=
1) {
    if
(signedUser?.roles?.includes(keys[i]))
return MAP[keys[i]].map(renderTab);
  }
  return null;
}, [signedUser, renderTab]);
```

```
const content = useMemo(() => (
  <Row gutter={[16, 16]}>
    <Col span={8}>
      <Row gutter={[16, 16]}>
        <Col span={24}>
          <Card
            className={styles.block}
            title='Profile Details'
            {...CARD_STYLE}
            bodyStyle={{
...CARD_STYLE.bodyStyle, flexDirection:
'row' }}
          >
            <Skeleton
loading={loading} avatar active>
              <div
className={styles.avatarWrap}>
                <Avatar size={50}
src={user.avatar?.url}>U</Avatar>
                <Typography.Text
strong className={styles.name}>
                  { user.name }
                <br />
              </div>
            </Skeleton>
          </Col>
        </Row>
      </Col>
    </Row>
  )
```

```

        {
user.roles?.map((role, idx) => (
            <Tag
key={idx.toString()}
color={USER_ROLE_COLORS[role] ||
'cyan'}>{ role }</Tag>
        )) }
        <Tag
color={USER_STATUS_COLORS[user.status]
|| 'default'}>{ user.status }</Tag>
            </Typography.Text>
        </div>
        </Skeleton>
    </Card>
</Col>
<Col span={24}>
    <Card
        className={styles.block}
        title='About'
        {...CARD_STYLE}
    >
        <Skeleton
loading={loading} active>
            <Can

perform={wildcard(PERMISSION.VIEW_USER_R
ATE, user.id)}

            yes={ (
                <Typography.Text
className={styles.description} copyable>
                    <Tooltip
title='Rate'><DollarOutlined
/></Tooltip>

                    {
formatCurrency(user.rate) }
                </Typography.Text>
            )}
        />
        <Typography.Link
href={`mailto:${user.email}`}
className={styles.description} copyable>

```

```

        <Tooltip
title='Email'><MailOutlined /></Tooltip>
            { user.email }
        </Typography.Link>
    </Skeleton>
</Card>
</Col>
<Col span={24}>
    <Can

perform={wildcard(PERMISSION.VIEW_USER_W
ORKTIME, user.id)}

            yes={ (
                <Card

className={styles.block}
                    {...CARD_STYLE}
                >
                    <Skeleton
loading={loading} active>
                        <UserCalendar />
                    </Skeleton>
                </Card>
            )}
            loading={<Loader
spanSize={24} />}
        />
    </Col>
</Row>
</Col>
<Col span={16}>
    <Row gutter={[16, 16]}>
        <Can

perform={wildcard(PERMISSION.VIEW_USER_P
ROFITS, user.id)}

            yes={<Profits />}
            loading={<Loader />}
        />
        <Can

```

```

perform={wildcard(PERMISSION.VIEW_USER_W
ORKTIME, user.id)}
    yes={<Time />}
    loading={<Loader />}
  />
  <Col span={24}>
    <Card
className={styles.block}
{...CARD_STYLE}>
      <Skeleton
loading={loading} active>
        <Tabs>
          {tabs}
        </Tabs>
      </Skeleton>
    </Card>
  </Col>
</Row>
</Col>
</Row>
), [user, tabs, loading]);

return (
  <>
    <PageHeader
      className={styles.pageHeader}
      title='User Profile'
      subTitle={user.name}
      onBack={back}
      extra={
        <SuspenseLoader
loading={loading} key={0}>
          <GuardedLink
            gate={{
              any:
[wildcard(PERMISSION.EDIT_USERS,
user.id)],
              except:
[wildcard(`!${PERMISSION.EDIT_USERS}`,
user.id)],
            }}
            href='/users/[id]/edit'
          as={` /users/${user.id}/edit`}
            loading={loading}
            hideIfFailed
          >
            <Button
              type='primary'>Edit</Button>
          </GuardedLink>
        </SuspenseLoader>,
      ]}
    </EntityAccessMiddleware
      entityName='user'      loading={loading}
      response={response}>
      { content }
    </EntityAccessMiddleware>
  </>
);
};

export default UserProfile;

```

A.2.8 Компонент Users

```
import { useEffect, useState, useCallback
} from 'react';
import { useDispatch, useSelector } from
'react-redux';
import {
  Table, Tag, Button, Space,
} from 'antd';
import Link from 'next/link';

import styles from './Users.module.scss';
import      usePagination      from
'../../utils/hooks/usePagination';
import      useFilters          from
'../../utils/hooks/useFilters';
import      GuardedLink         from
'../common/GuardedLink';
import      InviteModal         from
'../InviteModal/InviteModal';
import Can from '../common/Can';
import      SuspenseLoader      from
'../common/SuspenseLoader';
import {    fetchUsers    }    from
'../../store/users/actions';
import {    usersSelector  }    from
'../../store/users/selectors';
import {
  PERMISSION,              RESPONSE_MODE,
  USER_ROLE_COLORS, USER_STATUS_COLORS,
} from '../../utils/constants';

const COLUMNS = [
  {
    title: 'Name',
    dataIndex: 'name',
    render: (text, { id }) => <Link
href='/users/[id]'
as={`/${users}/${id}`}><a>{text}</a></Link>
  },

```

```
    {
      title: 'Email',
      dataIndex: 'email',
    },
    {
      title: 'Role',
      dataIndex: 'roles',
      render: (roles) => roles.map(
        (role, idx) => (<Tag
key={idx.toString()}
color={USER_ROLE_COLORS[role]      ||
'default'}>{role}</Tag>),
      ),
    },
    {
      title: 'Status',
      dataIndex: 'status',
      render: (text) => <Tag
color={USER_STATUS_COLORS[text]      ||
'default'}>{text}</Tag>,
    },
    {
      title: 'Created At',
      dataIndex: 'created_at',
    },
  ];

const Users = () => {
  const dispatch = useDispatch();
  const      [inviteModalVisible,
setInviteModalVisibility]      =
useState(false);
  const [users, total, loading,
filtersData]      =
useSelector(usersSelector);
  const [pagination, paginationOptions] =
usePagination();
  const [filters, sorting, filterOptions]
= useFilters(COLUMNS, filtersData);
```

```

    const openInviteModal = useCallback(()
=> setInviteModalVisibility(true), []);

    const closeInviteModal = useCallback(()
=> setInviteModalVisibility(false), []);

    useEffect(() => {
      dispatch(fetchUsers({
        ...pagination, filters, sorting,
mode: RESPONSE_MODE.SIMPLIFIED,
      }));
    }, [dispatch, pagination, filters,
sorting]);

    return (
      <>
        <SuspenseLoader loading={loading}
className={styles.buttons}>
          <Space
className={styles.buttons}>
            <GuardedLink
              gate={PERMISSION.ADD_USERS}
              href='/users/new'
              loading={loading}
              hideIfFailed
            >
              <Button
type='primary'>Add</Button>
            </GuardedLink>

```

```

      <Can
        perform={PERMISSION.ADD_INVITATIONS}
        yes={<Button type='primary'
onClick={openInviteModal}>Invite</Button
>}
      />
    </Space>
  </SuspenseLoader>
  <Table
    loading={loading}
    dataSource={users}
    pagination={{
      total,
      ...paginationOptions,
    }}
    {...filterOptions}
  />
  <InviteModal
    visible={inviteModalVisible}
    onCancel={closeInviteModal}
  />
</>
);
};

export default Users;

```

ДОДАТОК Б
Слайди презентації

Міністерство освіти та науки України
Національний університет «Запорізька політехніка»

ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
МЕТОДІВ ПЛАНУВАННЯ РЕСУРСІВ ІТ-КОМПАНІЇ

Виконав: Бессонов О. О.
ст. групи КНТ-119м
Керівник: Сердюк С. М.
к.т.н., доцент ЗНТУ

Рисунок Б.1 – Слайд №1

Об’єкт, предмет та мета роботи

Об’єкт дослідження: процеси балансування, оптимізації та автоматизації ресурсів підприємства.

Предмет дослідження: методи підвищення якості фінансового аудиту, підвищення ефективності менеджменту ресурсів та підвищення продуктивності управлінського персоналу.

Мета роботи: підвищення ефективності менеджменту ресурсів та оптимізація робочого процесу управлінського персоналу на ІТ підприємстві.

2

Рисунок Б.2 – Слайд №2

Огляд ERP-системи

ERP – це організаційна стратегія інтеграції виробництва та операцій, управління трудовими ресурсами, фінансового менеджменту та управління активами, орієнтована на безперервне балансування та оптимізацію ресурсів підприємства за допомогою спеціалізованого інтегрованого пакета прикладного програмного забезпечення, що забезпечує загальну модель даних і процесів для всіх сфер діяльності.

В якості характерної особливості ERP-стратегії наголошується принциповий підхід до використання єдиної транзакційної системи для переважної більшості операцій і бізнес-процесів організації, незалежно від функціональної і територіальної роз'єднаності місць їх виникнення і проходження, обов'язковість відомості всіх операцій в єдину базу для подальшої обробки та отримання в реальному часі збалансованих планів.

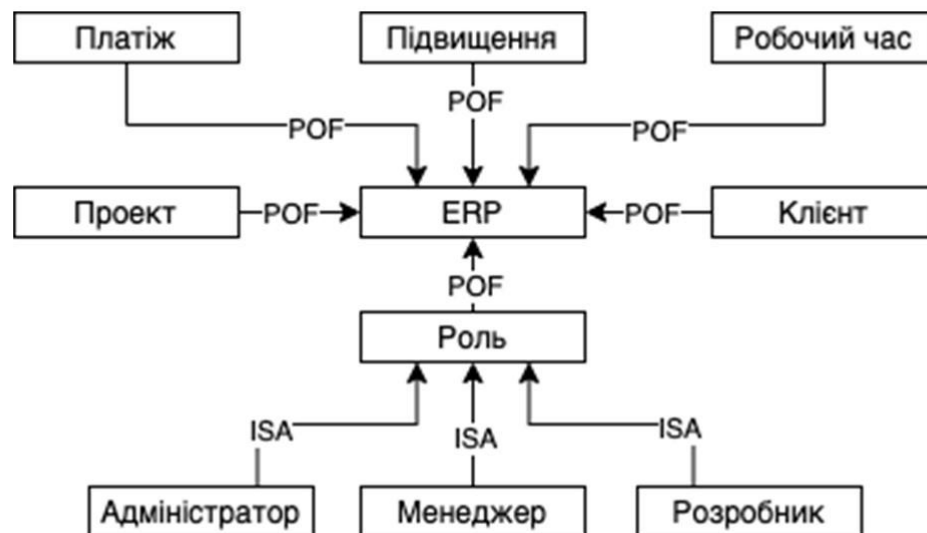
При підборі системи для оптимізації бізнесу дуже важливу роль відіграє архітектура програмного продукту. Від неї залежать терміни впровадження ERP-системи на підприємстві. За цим параметром можна виділити усього два типи програмних продуктів: з єдиною архітектурою та модульні.

Бізнес-програми з єдиною архітектурою відрізняються простотою установки і надійністю. Вони являють собою готовий неподільний інструмент, компоненти якого пов'язані між собою. Така архітектура служить своєрідним захистом від помилок і збоїв системи. Але подальша адаптація або розширення програмного продукту може викликати певні складнощі.

3

Рисунок Б.3 – Слайд №3

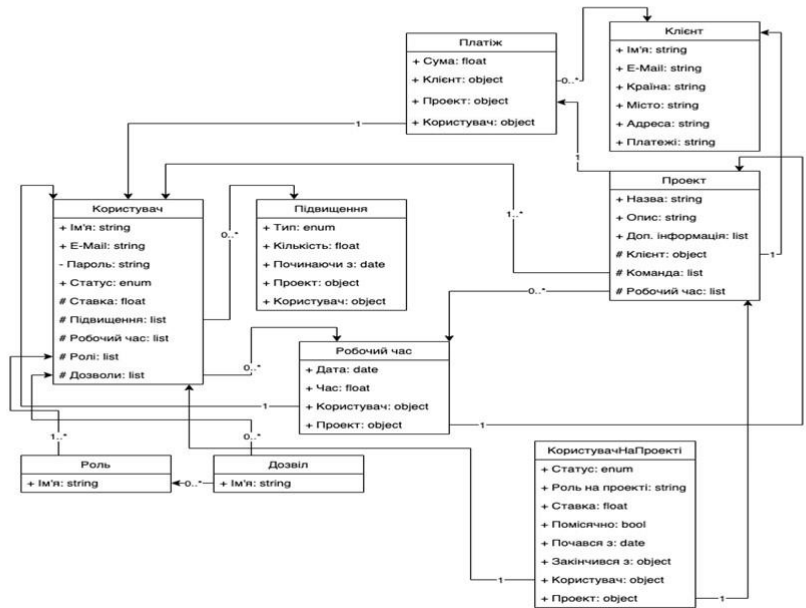
Відношення ERP-системи з зовнішніми акторами



4

Рисунок Б.4 – Слайд №4

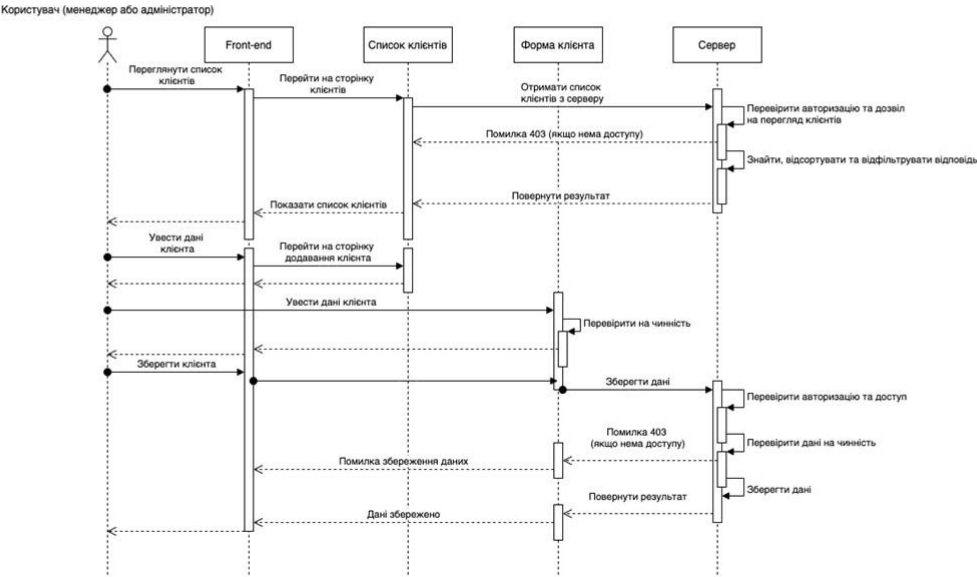
Діаграма класів в нотації UML 2.0



7

Рисунок Б.7 – Слайд №7

Діаграма послідовності в нотації UML 2.0



8

Рисунок Б.8 – Слайд №8

Діаграма діяльності

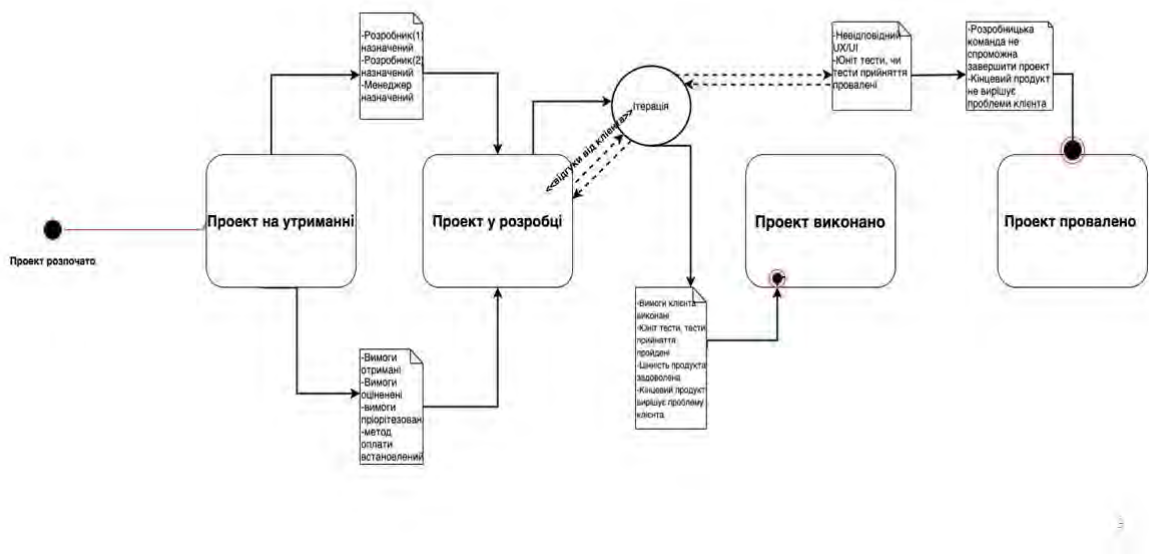


Рисунок Б.9 – Слайд №9

Порівняльна характеристика мов програмування для розробки веб-додатку

Критерій порівняння	Мова програмування				
	JS	Java	PHP	C#	Python
Швидкість виконання коду	++	+	+	+	++
Масштабованість	++	±	++	±	+
Популярність	++	+	++	±	++
Універсальність	++	++	++	+	++
Простота розроблення веб-інтерфейсу	++	±	+	±	++

Рисунок Б.10 – Слайд №10

Порівняльна характеристика мов програмування для розробки серверного додатку

Критерій порівняння	Мова програмування				
	JS (Node.js)	Java	PHP	C#	Python
Швидкість виконання коду	+	++	+	++	+
Масштабованість	++	+	++	+	+
Популярність	++	+	++	+	++
Універсальність	++	++	++	+	++
Простота розроблення API додатків	++	+	++	+	++

11

Рисунок Б.11 – Слайд №11

Порівняльна характеристика середовищ розробки

Критерій порівняння	Середовище розробки			
	VS Code	IntelliJ IDEA	NetBeans	Eclipse
Регулярність оновлень та модернізацій	++	++	±	±
Інтеграція системи контролю версій	++	++	+	+
Популярність	++	++	±	±
Модульність	++	++	-	-
Доступність	++	-	++	++
Багатофункціональність	++	++	++	+
Наявність додаткового інструментарію	+	++	+	+

12

Рисунок Б.12 – Слайд 12

Порівняльна характеристика JavaScript фреймворків

Критерій порівняння	JavaScript фреймворк				
	Vue	React	Ember	Svelte	Angular 2
Паралельне програмування	++	+	+	++	++
Масштабованість	++	++	++	+	+
Популярність	+	+++	+	+	++
Швидкість розробки кодової бази	+	++	+	++	+
Простота вивчення	++	++	+	+	+
Підтримуваність браузером	+	+	++	+	++

13

Рисунок Б.13 – Слайд №13

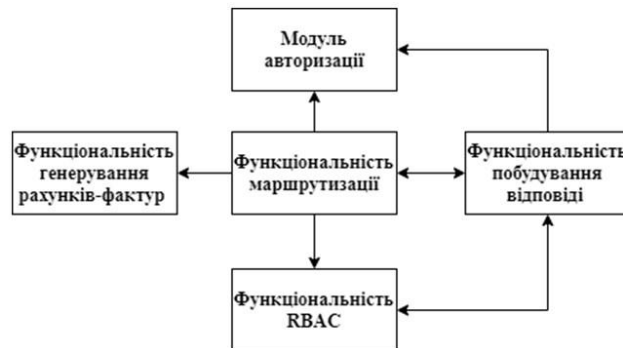
Порівняльна характеристика PHP фреймворків

Критерій порівняння	PHP фреймворк				
	Laravel	CodeIgniter	Symfony	CakePHP	Yii 2
Підтримка патернів	++	+	++	+	+
Масштабованість	++	+	++	+	+
Популярність	+++	+	++	+	++
Швидкість розробки кодової бази	++	++	++	+	+
Простота вивчення	++	++	+	+	+
Підтримка різних баз даних	+++	++	++	++	++

14

Рисунок Б.14 – Слайд №14

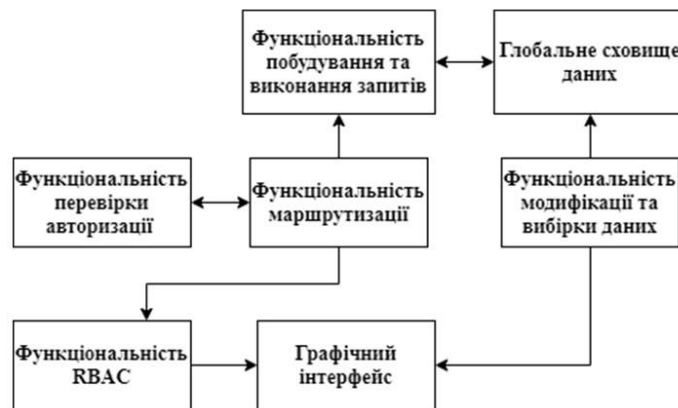
Схема основні структурних компонентів серверного додатку



15

Рисунок Б.15 – Слайд №15

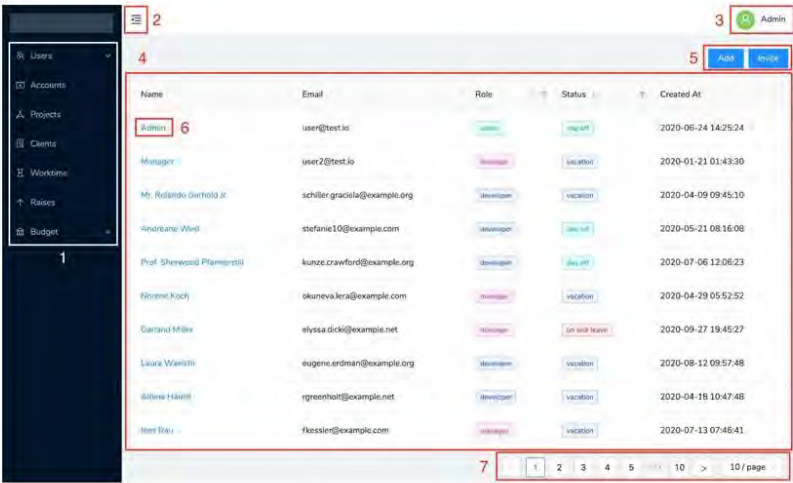
Схема основних структурних компонентів веб-додатку



16

Рисунок Б.16 – Слайд №16

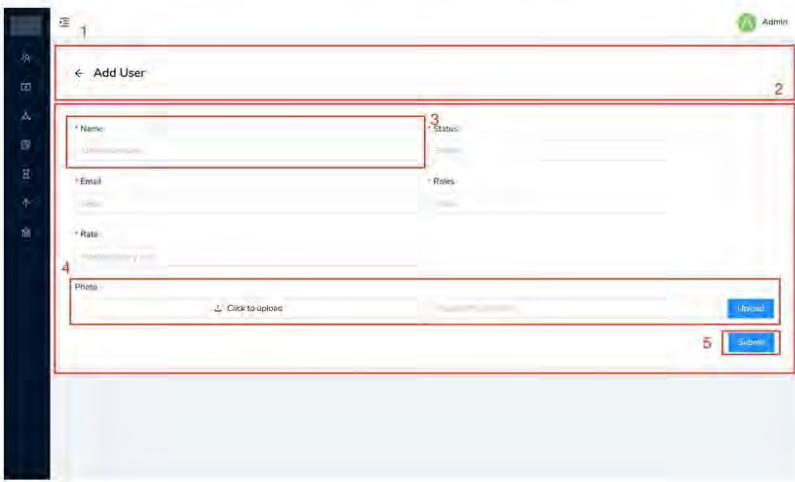
Приклад графічного інтерфейсу веб-додатку



- 1 – посилання до сторінок зі списками сутностей;
- 2 – кнопка приховування бокової панелі;
- 3 – блок з аватаром та ім'ям авторизованого користувача;
- 4 – таблиця з користувачами;
- 5 – кнопки дій;
- 6 – ім'я користувача, що є посиланням на його профіль;
- 7 – кнопки управління пагінацією;

Рисунок Б.17 – Слайд №17

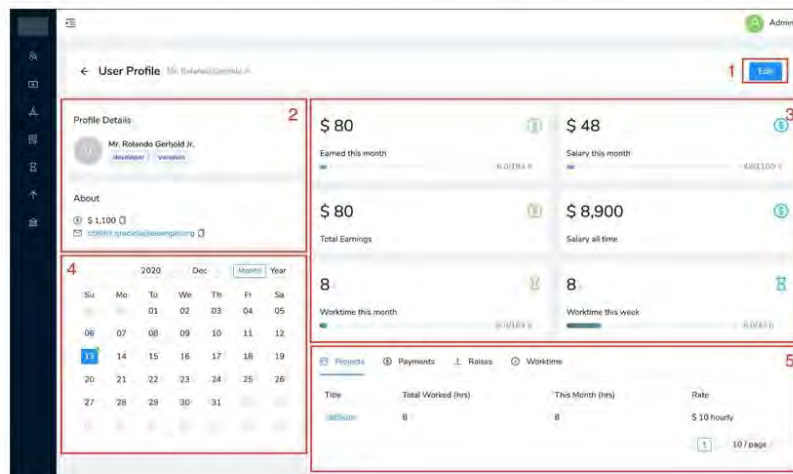
Приклад графічного інтерфейсу веб-додатку



- 1 – блок з назвою сторінки та посиланням для переходу на попередню сторінку;
- 2 – форма додавання користувача;
- 3 – обов'язкове поле для заповнення;
- 4 – необов'язкове поле для заповнення;
- 5 – кнопка підтвердження форми та відправки запиту на додавання запису до БД;

Рисунок Б.18 – Слайд №18

Приклад графічного інтерфейсу веб-додатку



1 — посилання на сторінку редагування запису;

2 — загальна інформація про користувача;

3 — скалькульована інформація про фінансовий оборот користувача;

4 — календар робочого часу;

5 — зв'язана з записом інформація у таблицях (проекти, платежі, підвищення, робочий час).

19

Рисунок Б.19 – Слайд №19

Основні варіанти використання програмного комплексу

- Організація, впорядкування та побудування відношень між сутностями, якими оперує ІТ підприємство (клієнти, розробники, менеджери, робочий час, проекти, платежі, підвищення)
- Перегляд підрахованої інформації за проектом або працівником: робочий час, прибутки, зарплати.
- Генерування рахунків-фактур для передачі клієнту

20

Рисунок Б.20 – Слайд №20

Висновки

Під час виконання магістерської роботи, було реалізовано програмний комплекс для підвищення якості фінансового аудиту, підвищення ефективності менеджменту ресурсів та оптимізації робочого процесу управлінського персоналу на ІТ підприємстві.

Розроблений програмний комплекс має велике практичне значення та рекомендується до використання в ІТ підприємствах для балансування та оптимізації ресурсів, а також підвищення загального показника продуктивності.

Наукова новизна результатів дослідження полягає у формалізації знань про предметну область робочого процесу на ІТ підприємстві, на підставі яких було зпроектовано архітектуру програмного комплексу ERP-системи та змодельовано комплекс моделей програмного забезпечення, що поліпшує процес розробки.

Розроблений програмний комплекс дозволяє підвищити загальний виробничий показник у ІТ підприємстві шляхом оптимізації робочого процесу та підвищення продуктивності управлінського персоналу.

21

Рисунок Б.21 – Слайд №21

ДЯКУЮ ЗА УВАГУ!

22

Рисунок Б.22 – Слайд №22