

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
СИСТЕМИ РОЗРАХУНКУ ЗАРОБІТНОЇ ПЛАТИ
RESEARCH AND SOFTWARE IMPLEMENTATION
OF THE PAYROLL SYSTEM

Виконав(ла): студент(ка) 2 курсу, групи КНТ-212м

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

ЗМІЄВСЬКИЙ М.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ГОФМАН Є.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент КАЗУРОВА А.Є.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.

Сергій СУББОТІН
“ ” 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЗМІЄВСЬКОГО Максима Віталійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація системи розрахунку заробітної плати.

Research and Software Implementation of the Payroll System

керівник проєкту (роботи) к.т.н., доцент, ГОФМАН Євген Олександрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 14 ” листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 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. Огляд використовуваних технологій. 3. Дослідження типових особливостей архітектури. 4. Дослідження особливостей розробки вебдодатку з використанням angular. 5. Розробка та реалізація системи з рохрахунку заробітної плати.

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	ГОФМАН Є.О., доцент		
Нормоконтролер	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання “08” вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Вибір засобів розробки програмного продукту.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6-7 тижні	Розділ 3
5	Розробка програмного продукту.	8-9 тижні	Розділ 4, 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Максим ЗМІЄВСЬКИЙ
(Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Євген ГОФМАН
(Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
100 с., 1 табл., 34 рис., 2 дод., 46 джерел.

ШТУЧНИЙ ІНТЕЛЕКТ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, РОЗРОБКА
ВЕБДОДАТКУ, INTELLIJ IDEA, API OPENAI.

Об'єкт дослідження – програмні засоби для розрахунку заробітної плати.

Предмет дослідження – комп'ютерні програми для розрахунку заробітної плати.

Мета роботи – створення програмної системи для впровадження ефективної системи розрахунку заробітної плати для оптимізації процесів управління фінансовими ресурсами.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мова програмування Angular, середовище розробки IntelliJ IDEA, API OpenAI, комп'ютер з Intel Core i7.

Результати. Результатом проведеного дослідження є розроблена та успішно впроваджена програмна система, яка забезпечує точний та ефективний розрахунок заробітної плати.

Висновки. Розроблено програмну систему для розрахунку заробітної плати, яка оптимізує процеси управління фінансовими ресурсами.

Галузь використання – розроблена система може бути використана у різних галузях бізнесу та організаціях для автоматизації процесів розрахунку та обліку заробітної плати, сприяючи підвищенню ефективності управлінських рішень.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 100 pages, 1 table, 34 figures, 2 appendix, 46 sources.

ARTIFICIAL INTELLIGENCE, SOFTWARE, WEB APPLICATION DEVELOPMENT, INTELLIJ IDEA, OPENAI API.

The object of the research is software tools for calculating wages.

The subject of research is computer programs for calculating wages.

The purpose of the work is to create a software system for implementing an effective payroll system to optimize financial resource management processes.

Materials, methods and technical means: structural and object-oriented programming, Angular programming language, IntelliJ IDEA development environment, OpenAI API, computer with Intel Core i7.

The results. The result of the conducted research is a developed and successfully implemented software system that provides accurate and efficient calculation of wages.

Conclusions. A software system has been developed for calculating wages, which optimizes the processes of managing financial resources.

Field of use - the developed system can be used in various business sectors and organizations to automate the processes of calculating and accounting for wages, contributing to the improvement of the efficiency of management decisions.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Теоретичні аспекти розрахунку заробітної плати	10
1.1.1 Основні концепції розрахунку заробітної плати	10
1.1.2 Фактори, що впливають на розрахунок заробітної плати.....	11
1.1.3 Сучасні тенденції у розрахунку заробітної плати	11
1.2 Актуальні підходи до автоматизації розрахунків	12
1.3 Висновки до першого розділу.....	13
2 Огляд використовуваних технологій	15
2.1 Angular: особливості та використання у програмній реалізації.....	15
2.2 Tailwind: методи та переваги у розробці інтерфейсу	16
2.3 Jasmine: основні принципи тестування програмного забезпечення	18
2.4 Firebase Realtime Database: огляд можливостей та використання для систем розрахунку	19
2.5 Концепція сінгл пейдж додатку.....	20
2.6 Порівняння з традиційним вебсайтом	22
2.7 Порівняння з нативним додатком.....	23
2.8 Підсумкова порівняльна таблиця	23
2.9 Огляд особливостей обраної середи розробки.....	24
2.10 Висновки до другого розділу	25
3 Дослідження типових особливостей архітектури.....	27
3.1 Розподілення ролей	27
3.2 Інфраструктура	28
3.2.1 Клієнтська маршрутизація	28
3.2.2 Компоненти.....	28
3.2.3 Модульність.....	29

	7
3.2.4 API модуль	29
3.2.5 Інверсія управління	29
3.2.6 Графічний інтерфейс	30
3.2.7 Шаблони	31
3.2.8 Зв'язування	31
3.3 Шаблони MVC та MVVM	32
3.4 Висновки по третьому розділу	33
4 Дослідження особливостей розробки вебдодатку з використанням angular	35
4.1 Підготовка прикладу	35
4.2 Збірка та запуск	37
4.3 Модулі та TypeScript	39
4.4 Компоненти	41
4.5 Директиви	42
4.6 Сервіси	44
4.7 Клієнтська маршрутизація	45
4.8 Висновки по четвертому розділу	47
5 Розробка та реалізація системи з розрахунку заробітної плати	48
5.1 Контролю часу на виконання завдань	48
5.2 Календарь для додавання відсутностей	50
5.3 Управління заробітної плати	52
5.4 Управління користувачами	53
Висновки	54
Перелік джерел посилань	55
Додаток А Текст програми	59
Додаток Б Слайди презентації	96

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AI	– Artificial Intelligence;
API	– Application Programming Interface;
BERT	– Bidirectional Encoder Representations from Transformers;
CNN	– Convolutional Neural Network;
CSS	– Cascading Style Sheets;
CTRL	– Conditional Transformer Language Model;
DAN	– Do Anything Now;
DOM	– Document Object Model;
ES6	– ECMAScript 6;
GAN	– Generative Adversarial Networks;
GB	– GigaByte;
HTML	– HyperText Markup Language;
HTTP	– HyperText Transfer Protocol;
JS	– JavaScript;
NLP	– Natural Language Processing;
RAM	– Random Access Memory;
RNN	– Recurrent Neural Network;
SVG	– Scalable Vector Graphics;
TS	– TypeScript;
UI	– User interface;
URL	– Uniform Resource Locator;
VAE	– Variational Autoencoders;
ООП	– об'єктно-орієнтоване програмування;
ШІ	– штучний інтелект.

ВСТУП

В сучасному світі, що стрімко розвивається в галузі інформаційних технологій, автоматизація та оптимізація управлінських процесів у підприємствах стає необхідністю для забезпечення їх ефективності та конкурентоспроможності. Однією з ключових складових цього процесу є система розрахунку заробітної плати, яка відіграє важливу роль у взаємодії між працівниками та роботодавцями. Дотримання правил, регулювання законодавства та ефективна реалізація цієї системи є критично важливими для забезпечення справедливості та прозорості у відносинах між учасниками працівничого процесу.

Мета даної дипломної роботи полягає у вивченні та дослідженні сучасних підходів до розрахунку заробітної плати, а також в розробці програмної системи, спрямованої на автоматизацію цього процесу з метою підвищення ефективності та точності розрахунків. Враховуючи складність сучасних вимог до обліку робочого часу, податкових відшкодувань, страхових внесків та інших аспектів, важливо розробити інтегровану систему, яка враховує всі аспекти заробітної плати та забезпечує їхню відповідність вимогам законодавства.

У цьому контексті дипломна робота розглядається як спроба внести вагомий внесок у сферу управління персоналом та оптимізації бізнес-процесів, дозволяючи компаніям ефективніше використовувати свої людські ресурси та забезпечити якісний та точний розрахунок заробітної плати.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В першу чергу потрібно зрозуміти суть і особливості предметної області до системи, яка розробляється.

1.1 Теоретичні аспекти розрахунку заробітної плати

В розділі "Теоретичні аспекти розрахунку заробітної плати" ключовою метою є розкриття та дослідження фундаментальних понять, принципів та аспектів, які стосуються оплати праці. Такий розділ допомагає уточнити підходи до визначення заробітної плати, вивчити фактори, які впливають на її формування, а також оцінити сучасні тенденції у цій сфері [1].

1.1.1 Основні концепції розрахунку заробітної плати

Розрахунок заробітної плати – це складний процес, який базується на кількох ключових концепціях. Ці концепції визначають оплату за працю та враховують різні аспекти, пов'язані з визначенням заробітної плати працівників.

Перш за все, це вивчення систем оплати праці, таких як системи оплати за годину, за проектом, за виконану роботу чи за результатами. Різні компанії використовують різні підходи до оплати праці своїх співробітників [2].

Також важливо розглянути преміальні системи. Це може включати бонуси, премії за досягнення певних цілей, а також програми мотивації, спрямовані на підвищення продуктивності та мотивації працівників.

Визначення ставок оплати — ще один важливий аспект. Фактори, такі як досвід роботи, кваліфікація, освіта, рівень відповідальності,

конкурентоспроможність на ринку праці та інші показники, визначають конкретні ставки заробітної плати [3].

Також слід враховувати законодавче регулювання, таке як мінімальна заробітна плата та податкові ставки для оподаткування заробітної плати, а також законодавство щодо соціального страхування та пенсійних виплат.

Внутрішні та зовнішні чинники, такі як стратегія компанії, фінансова стійкість, економічні та ринкові умови, також впливають на процес визначення заробітної плати.

Ці концепції утворюють основу для реалізації оплати праці в організаціях та дозволяють враховувати різноманітні фактори для справедливого та конкурентоспроможного визначення оплати праці [4].

1.1.2 Фактори, що впливають на розрахунок заробітної плати

У цьому пункті розглядаються фактори, що враховуються при розрахунку заробітної плати. Вони включають досвід працівника, кваліфікацію, ринкові умови та нормативно-правове середовище. Аналізується взаємозв'язок цих факторів із системою оплати праці та їх вплив на стимулювання працівників [5].

1.1.3 Сучасні тенденції у розрахунку заробітної плати

Однією з ключових тенденцій є перехід до більш гнучких моделей оплати праці. Це може означати перехід від традиційних стандартних годинних або місячних ставок до систем, які базуються на результативності, досягненнях, або навіть виконанні конкретних завдань.

Ще однією важливою тенденцією є зростання значення врахування не тільки фінансових аспектів, а й нематеріальних факторів оплати праці. Це може включати оцінку балансу між роботою та особистим життям,

можливості професійного росту, корпоративну культуру та багато інших аспектів, які впливають на мотивацію працівників.

Для компаній стає дедалі важливішим врахування впливу технологій на розрахунок заробітної плати. Використання програмного забезпечення та різноманітних онлайн-інструментів для обліку робочого часу, виконання завдань та моніторингу продуктивності дозволяє більш точно оцінювати та оплачувати роботу працівників [6].

Також варто відзначити тенденцію до більшої прозорості та відкритості в питаннях оплати праці. Компанії все частіше надають працівникам доступ до інформації про заробітну плату, критерії оцінки та критерії надання бонусів. Це сприяє збереженню довіри та створенню більш прозорих відносин між роботодавцями та працівниками.

Успішні компанії намагаються бути в курсі цих тенденцій, оскільки вони впливають на спосіб, яким вони мотивують, утримують та розвивають свій персонал.

1.2 Актуальні підходи до автоматизації розрахунків

Актуальні підходи до автоматизації розрахунків заробітної плати є ключовим елементом оптимізації робочих процесів у сучасному бізнесі.

Одним з основних підходів є використання інтегрованих систем управління людськими ресурсами (HRIS), що дозволяють ефективно виконувати завдання з розрахунків заробітної плати. Ці системи об'єднують в собі функції автоматизації обліку робочого часу, оплати праці, а також інші аспекти управління персоналом.

Ще одним популярним підходом є використання хмарних технологій для зберігання та обробки даних заробітної плати. Це дає можливість компаніям ефективно зберігати великий обсяг інформації, забезпечуючи одночасно доступ до неї з будь-якого пристрою та зручний аналіз [7].

Також варто зазначити розвиток технологій штучного інтелекту (AI) та машинного навчання (ML) у сфері розрахунків заробітної плати. Впровадження цих технологій дозволяє знизити людську втручання в процеси розрахунків, автоматизувати обробку даних та підвищує точність розрахунків.

Крім цього, важливим аспектом є впровадження розумних аналітичних інструментів для прогнозування та аналізу заробітної плати. Вони дозволяють компаніям здійснювати більш точний прогноз витрат на оплату праці, а також аналізувати показники ефективності різних стратегій оплати праці.

Усі ці підходи спрямовані на впровадження ефективних та точних систем розрахунків заробітної плати, що дозволяють підвищити продуктивність та оптимізувати управління людськими ресурсами [8].

1.3 Висновки до першого розділу

Зважаючи на розглянуті аспекти теоретичних концепцій та актуальних підходів у розрахунку заробітної плати, важливо відзначити значущість впровадження інноваційних стратегій управління персоналом у сучасному бізнесі. Теоретичні аспекти дозволяють зрозуміти основні концепції та методи, на основі яких побудовані процеси розрахунків оплати праці. Вони формують теоретичну базу для розвитку та впровадження практичних аспектів у сфері управління заробітною платою.

Однак, оглянуті актуальні підходи до автоматизації розрахунків виявляються важливими для підвищення ефективності та точності обробки даних заробітної плати. Впровадження інтегрованих систем управління людськими ресурсами, хмарних технологій для зберігання даних, а також використання технологій штучного інтелекту та машинного навчання сприяють автоматизації процесів та покращенню управління заробітною платою.

Загалом, ураховуючи висвітлені теоретичні та практичні аспекти, можна визначити, що поєднання фундаментальних концепцій та інноваційних підходів визначає успішність і ефективність процесів розрахунків заробітної плати, сприяючи розвитку підприємства та оптимізації управління людськими ресурсами.

2 ОГЛЯД ВИКОРИСТОВУВАНИХ ТЕХНОЛОГІЙ

2.1 Angular: особливості та використання у програмній реалізації

Angular – це потужний фреймворк, спрямований на створення односторінкових динамічних вебзастосунків. У відповідності до контексту системи розрахунку заробітної плати, Angular може забезпечити ряд переваг:

- один компонент — одна функція. Angular базується на принципі компонентної архітектури, що дозволяє розбити вебзастосунок на окремі компоненти. В контексті системи розрахунку заробітної плати, це означає створення окремих компонентів для кожного фрагменту функціоналу: можливість введення даних, аналіз результатів, графіки звітів тощо. Компоненти можна легко перевикористовувати та підтримувати, що робить систему більш модульною та масштабованою [9];

- реактивне програмування. Angular пропонує реактивне програмування через використання RxJS - бібліотеки для реактивного програмування на JavaScript. Це дозволяє працювати з потоками даних та реагувати на зміни в реальному часі. У системі розрахунку зарплати, реактивність може забезпечити миттєве оновлення даних під час введення нових параметрів або змін;

- маршрутизація та керування станом додатку. Angular має вбудовану систему маршрутизації, яка дозволяє керувати різними сторінками або функціоналом системи. Це дозволяє створювати користувацький інтерфейс з розділеними сторінками для різних етапів роботи з системою. Користувачі можуть швидко переміщатися між різними функціональними блоками;

- тестування та робочий процес. Angular надає можливості для написання тестів, що дозволяє перевіряти якість коду та коректність роботи функціоналу. Це особливо важливо для системи розрахунку заробітної плати, де точність та надійність розрахунків мають велике значення;

- модульність та розширюваність. Angular надає можливість розбити програму на модулі. Це дозволяє розділити функціонал на логічні групи та дозволяє використовувати лише ті частини, які потрібні у певний момент. У системі розрахунку зарплати це дозволяє зберігати код чистим та підтримуваним;

- підтримка від Google та широкий ком'юнітет. Angular є фреймворком, який підтримується Google та має широкий ком'юнітет розробників. Це означає швидшу реакцію на знайдені помилки, безпечність та постійний розвиток фреймворку [10];

- Two-way data binding для миттєвого оновлення даних. Angular використовує two-way data binding, що дозволяє автоматично оновлювати дані в користувацькому інтерфейсі при їх зміні у програмній логіці. Це особливо важливо для системи розрахунків, де користувач може бачити зміни у даних миттєво;

- Dependency Injection для кращого управління залежностями. Angular має механізм Dependency Injection, що спрощує впровадження залежностей у компоненти. Це дозволяє створювати чистий та підтримуваний код для обробки даних та логіки розрахунків.

Інтеграція Angular у систему розрахунків заробітної плати може допомогти створити інтерактивний та ефективний інтерфейс для користувачів, що дозволить їм зручно взаємодіяти з функціоналом системи та отримувати потрібну інформацію.

2.2 Tailwind: методи та переваги у розробці інтерфейсу

Tailwind CSS - це інструмент, який надає розробникам гнучкий підхід до створення користувацьких інтерфейсів. Однією з головних переваг цього інструменту є можливість швидко та ефективно працювати з CSS без необхідності писати власний код. Використання готових класів Tailwind

дозволяє швидко застосовувати стилі до елементів без необхідності вивчати або розуміти всі деталі CSS [11].

Підхід, який пропонує Tailwind CSS, дозволяє створювати інтерфейси, які відповідають сучасним тенденціям у дизайні. Він надає широкий спектр можливостей для розробки зручних та естетичних інтерфейсів, що допомагає підвищити користувацький досвід.

Однією з ключових переваг Tailwind CSS є можливість швидкого розгортання нових компонентів та елементів інтерфейсу, що зменшує час, потрібний для розробки та підтримки проєктів. Використання готових стилів та компонентів прискорює робочий процес і дозволяє швидше втілювати новий функціонал чи зміни.

Крім того, Tailwind CSS сприяє однорідності в стилізації інтерфейсу проєкту, що важливо при роботі в команді. Використання однакових класів та підходів спрощує розуміння та підтримку коду для різних розробників.

Цей інструмент не обмежує в творчих можливостях: він надає достатній рівень гнучкості для кастомізації та налаштування стилів під конкретні потреби проєкту.

У підсумку, Tailwind CSS став популярним інструментом завдяки своїй простоті в застосуванні, гнучкості та швидкості розробки, що сприяє ефективній роботі розробників у створенні сучасних та інтуїтивно зрозумілих інтерфейсів.

Розглянувши ці аспекти, важливо врахувати, що Tailwind CSS може стати потужним інструментом для розробки інтерфейсу у проєктах, де потрібна швидка і ефективна стилізація елементів [12].

У цьому проєкті Tailwind CSS є необхідним інструментом, оскільки рух технологій вимагає прагнення створювати сучасний, легкий та швидкий у використанні інтерфейс.

2.3 Jasmine: основні принципи тестування програмного забезпечення

Jasmine, як фреймворк для тестування JavaScript, є надзвичайно важливим інструментом для забезпечення якості та стабільності нашого програмного забезпечення. Одним із основних факторів, який робить його незамінним, є можливість створення тестів, що дозволяють виявляти потенційні помилки та недоліки в коді ще на ранніх етапах розробки. Це означає, що ми можемо виявити та виправити помилки до того, як вони вплинуть на фінальний продукт. Такий підхід забезпечує вищу якість нашого ПЗ та допомагає уникнути значної кількості витрат на виправлення дефектів під час подальшого розвитку продукту. Окрім цього, виявлення проблем на ранніх етапах дозволяє нам планувати та впроваджувати виправлення в більш комфортний та ефективний спосіб, що збільшує загальну ефективність розробки [13].

Використання Jasmine не лише дозволяє перевіряти коректність окремих функцій та компонентів, але й стимулює активну співпрацю команди у процесі роботи над якістю коду. Це надзвичайно важливо для оптимізації процесу розробки, оскільки спільна робота команди у сфері тестування дозволяє уніфікувати підходи до тестування та підтримувати однорідний стиль написання тестів у всьому проєкті.

Залучення команди до тестування дозволяє не тільки ефективно виявляти та виправляти помилки, а й створювати внутрішні стандарти тестування, що підвищує якість коду та програмного продукту загалом. Окрім цього, спільна відповідальність за якість ПЗ стимулює кожного учасника команди вносити свій внесок у забезпечення стабільності та ефективності продукту.

Проведення тестів з використанням Jasmine не лише сприяє виявленню потенційних проблем, але й формує культуру якісної розробки, що є важливим елементом успішного проєкту. Використання цього

інструменту дозволяє нашій команді працювати як єдиний механізм, спрямований на досягнення високої якості програмного забезпечення [13].

Використання Jasmine надає можливість автоматизувати процес тестування, що відіграє ключову роль у збільшенні швидкості та ефективності розробки програмного забезпечення. Запуск тестів через цей фреймворк дозволяє нам оперативно виявляти та усувати помилки, зменшуючи час, потрібний для вирішення проблем.

Автоматизоване тестування має велике перевагу перед ручним, оскільки воно усуває ризик людських помилок і дозволяє швидше та точніше виявляти проблеми у програмному забезпеченні. Використання Jasmine для автоматизації тестування гарантує стабільність продукту шляхом надійної перевірки всіх його аспектів.

Автоматизовані тести забезпечують високу точність у виявленні проблем, а також швидке реагування на зміни та оновлення в коді. Це спрощує процес вдосконалення програми та збільшує її стабільність, зменшуючи кількість непередбачуваних помилок у програмному забезпеченні.

2.4 Firebase Realtime Database: огляд можливостей та використання для систем розрахунку

Firebase Realtime Database - це хмарна база даних від компанії Google, яка забезпечує реальний час оновлення та синхронізацію даних між різними користувачами або пристроями. Вона пропонує швидке та ефективне зберігання та обмін даними в реальному часі [14].

Можливості Firebase Realtime Database включають:

- реальний час оновлення: База даних працює у режимі реального часу, що означає, що будь-яка зміна даних негайно синхронізується і відображається для всіх користувачів, які мають до них доступ;

- гнучкість: Firebase Realtime Database працює на основі JSON-структури, що надає гнучкість у зберіганні даних та можливість організації їх у вигляді дерева з гнучкими ключами;
- автентифікація та безпека: Firebase надає інструменти для автентифікації користувачів та керування доступом до даних, що забезпечує безпеку та конфіденційність даних;
- інтеграція з іншими сервісами Firebase: Firebase Realtime Database може легко інтегруватися з іншими сервісами Firebase, такими як зберігання файлів, аутентифікація, аналітика та інші;
- щодо використання для систем розрахунку, Realtime Database може використовуватися для зберігання даних у реальному часі, таких як замовлення, транзакції, оновлення статусів тощо. Вона дозволяє різним пристроям та користувачам одночасно отримувати доступ до актуальних даних та проводити операції над ними. Такий підхід може полегшити системи розрахунків, дозволяючи швидше та ефективніше оновлювати та синхронізувати дані між різними пристроями та користувачами.

2.5 Концепція сігл пейдж додатку

Розглянемо типовий, звичайний сайт, на прикладі Wikipedia. При відкритті статті користувач очікує, що браузер встановить зв'язок із сервером та обробить його запит. Потім браузер завантажує різні ресурси для відображення сторінки, одночасно показуючи ті, що вже завантажилися. У цьому процесі елементи сторінки можуть змінювати місце, анімуючись та перераховуючись, що може робити відображення нестабільним. В залежності від швидкості мережі та робочого навантаження сервера, цей процес може займати значний час, що може дратувати користувача. Коли користувач бажає переглянути іншу статтю, відредагувати поточну або здійснити інші дії на сайті, відбувається повторне завантаження сторінки за новою URL-адресою. Частину ресурсів може кешувати браузер, що дозволяє швидше

завантажувати нову сторінку, але недоліки цього процесу все одно залишаються [15]-[17].

Більшість сайтів в інтернеті працюють за подібним принципом. Однак з часу створення Web пройшло багато років, і технології значно змінилися. JavaScript, AJAX та інші інструменти дозволили сайтам стати більш інтерактивними та функціональними, не лише обмежуючись простим відображенням сторінок. Зараз розробники мають достатньо технологій, щоб змінити поведінку схожим на нативний додаток операційної системи. Для досягнення цього вони використовують скрипти для здійснення всіх змін на сторінці вебсайту і використовують AJAX для взаємодії з сервером. При відкритті такого сайту користувач завантажує лише один раз сторінку, а всі подальші зміни, які не потребують нових даних, відбуваються швидше, оскільки не чекають відповіді від сервера. Цей підхід отримав назву "односторінковий додаток" (Single Page Application) [18]-[19]. І все більше нових сайтів створюються саме за такою концепцією, оскільки вона дозволяє покращити швидкість та ефективність вебдодатків (рис. 2.1 – 2.2).



Рисунок 2.1 – Життєвий цикл традиційного вебсайту



Рисунок 2.2 – Життєвий цикл односторінкового додатку

Отже, назва не відображає головної суті концепції односторінкового додатку. Його суть полягає не в обмеженні кількості сторінок на вебсайті, а в тому, щоб зробити сайт подібним до нативного додатка. Це досягається шляхом усунення необхідності постійних перезавантажень сторінок та переміщення логіки відтворення інформації з сервера на сторону користувача, таким чином знижуючи залежність функціонування сайту від сервера та Інтернет-з'єднання.

2.6 Порівняння з традиційним вебсайтом

Концепція односторінкового додатку виглядає дуже привабливо, однак має як певні переваги, так і недоліки.

Переваги односторінкових додатків:

- багатий функціонал інтерфейсу;
- швидка реакція інтерфейсу без необхідності постійних запитів до сервера;
- значне зменшення навантаження на сервер;
- спрощення логіки та складності серверної частини;
- близькість в дизайні до нативних додатків операційних систем.

Недоліки:

- збільшене навантаження на клієнтську частину через велику кількість JavaScript;
- складність розробки.

2.7 Порівняння з нативним додатком

Нативні додатки - ціль, до якої прагне подібність односторінкових вебдодатків. Однак навіть нативні додатки мають свої обмеження. Односторінковий вебдодаток намагається взяти найкраще з обох світів - вебсайтів і нативних додатків. Тим не менше, у деяких випадках використання звичайного нативного додатка може бути перевагою чи навіть необхідністю. Розглянемо їх переваги та недоліки.

Переваги односторінкових вебдодатків:

- підтримка кількох платформ;
- відсутність потреби встановлення;
- не займає місця на жорсткому диску;
- можливість використання на віддалених пристроях для обчислень.

Переваги нативних додатків:

- свобода вибору мов програмування та інтерфейсів;
- вища продуктивність нативного коду;
- використання унікальних можливостей платформи;
- доступ до будь-якого обладнання комп'ютера;
- незалежність від Інтернет-з'єднання.

2.8 Підсумкова порівняльна таблиця

У таблиці 2.1 наведено порівняння традиційного вебсайту, односторінкового вебдодатку та нативного додатку.

Таблиця 2.1 – Порівняння традиційного вебсайту, односторінкового вебдодатку та нативного додатку

	Вебсайт	Односторінковий вебдодаток	Нативний додаток
Чуйність інтерфейсу	- маленька	+ велика	+ велика
Встановлення	+ не потрібно	+ не потрібно	- потрібно
Місце на ПЗУ	+ не потрібно	+ не потрібно	- потрібно
Інтернет з'єднання	-- дуже потрібно	- потрібно	+ не потрібно
Наявність серверу	-- дуже потрібно	- потрібно	+ не потрібно
Навантаження на клієнт	+ середнє	- велике	+ середнє
Доступ до обладнання клієнта та можливостей платформи	- не має	- не має	+ є
Багатоплатформність	+ чудова	+ чудова	- погана

2.9 Огляд особливостей обраної середи розробки

Зважаючи на природу системи розрахунку зарплати, вона, складається з трьох основних компонентів: клієнтської частини, серверної інфраструктури та бази даних. Клієнтська частина повинна включати інтерфейси користувача, де працівники можуть взаємодіяти з системою для введення даних та перегляду звітів. Серверна частина відповідає за обробку цих даних, обчислення зарплати та зберігання результатів в базі даних.

У клієнтській частині системи можуть бути різні модулі для взаємодії з працівниками, де вони можуть вводити дані про робочий час, відпустки, додаткові винагороди тощо. Ці дані потім передаються до серверної частини

через API або інші засоби комунікації. Сервер обробляє ці дані, проводить необхідні розрахунки та зберігає їх у базі даних. База даних містить інформацію про співробітників, їхні умови праці, та інші дані, які потрібні для розрахунків.

Структура бази даних може бути організована таким чином, що містить таблиці для зберігання основних даних про працівників, їхні заробітні плати, графіки робочого часу, інформацію про відпустки та додаткові винагороди. Додатково, можуть бути створені таблиці для звітності, яка включає історію змін, статистику робочого часу, та інші дані для аналізу та звітності.

Залежно від специфіки системи, для реалізації цієї архітектури можуть бути використані різноманітні технології. Наприклад, для розробки клієнтської частини може бути обрано Angular для створення користувацьких інтерфейсів, Firebase Realtime Database для зберігання та маніпулювання даними в режимі реального часу, та Node.js для створення серверної частини.

Ці компоненти взаємодіють між собою, обмінюючи необхідну інформацію та забезпечуючи коректне функціонування системи розрахунку заробітної плати.

2.10 Висновки до другого розділу

Аналізуючи зазначені технології - Angular, Tailwind, Jasmine та Firebase Realtime Database - можна визначити їхню важливість у контексті розробки системи розрахунку заробітної плати. Angular, надійний фреймворк, надає інструменти для створення вебдодатків високої якості. Tailwind, спрощуючи розробку інтерфейсу, дозволяє швидко створювати стильовані компоненти. Jasmine допомагає поліпшити якість програмного забезпечення через тестування на початкових етапах розробки. Firebase Realtime Database, забезпечуючи оновлення даних у реальному часі, є ключовим для систем розрахунків, де потрібна актуальність та

синхронізованість даних. Ці технології в комплексі надають широкі можливості та інструменти для поліпшення процесу розробки програмного забезпечення та підвищення його якості, що відображає значущість їхнього використання у створенні ефективної системи розрахунку заробітної плати.

3 ДОСЛІДЖЕННЯ ТИПОВИХ ОСОБЛИВОСТЕЙ АРХІТЕКТУРИ

3.1 Розподілення ролей

В традиційних вебсайтах відображення надається в основному сервером, що підтримує значні обсяги роботи. У односторінкових додатках основну частину роботи виконують клієнти, що розвантажує сервер і дозволяє йому концентруватися на наданні сервісних послуг [20]. Проте, оскільки створення та оформлення всіх HTML-сторінок, за винятком головної, відбувається на клієнті, у процесі веброзміщення односторінкового додатку достатньо лише статичного вебсервера та API-сервера для сервісу, у випадку потреби.

Традиційний вебсайт:

а) клієнт (браузер):

- 1) відтворення HTML;
- 2) деякі незначні скрипти Сервер;
- 3) створення HTML;
- 4) маршрутизація;
- 5) логіка роботи графічного інтерфейсу;
- 6) API сервісу;

б) односторінковий додаток;

в) клієнт (браузер):

- 1) створення HTML;
- 2) маршрутизація;
- 3) логіка роботи графічного інтерфейсу Сервер;
- 4) статичний вебсервер;
- 5) API сервіс.

3.2 Інфраструктура

3.2.1 Клієнтська маршрутизація

Як ми знаємо, додатки з одною сторінкою мають один URL. Інші зміни та стани сторінки пов'язані з змінними JavaScript та цільниками даних такими як Cookie та LocalStorage [21]. Перезавантажуючи сторінку або відкриваючи її в іншому браузері або на іншому пристрої, ми отримуємо початковий стан сторінки. Щоб зберегти стан після перезавантаження сайту, можна використовувати сховища зверху, але це не дозволить нам перенести стан до іншого браузера, додатка чи пристрою. Для невеликих додатків таке може бути не потрібно, але в багатостаномичних додатках використання URL для передачі станів важлива функція. Як правило, в традиційних сайтах для цього використовується різні сторінки (кожна з яких має свій URL). Односторінкові додатки не обмежують кількість сторінок для перенаправлення, а навпаки, застосовують принципи неперезавантаження, що використовує частину URL для сторінки, зазвичай символ «#». Зміни URL реєструються в історії браузера, а це достатнє для браузерної навігації, а також для того, щоб скористатися кнопкою «Назад» у браузері.

Отже, реалізація інтегрованої маршрутизації (серверної і клієнтської) є необхідною для створення односторінкового додатка.

3.2.2 Компоненти

Односторінковий додаток містить велику кількість скриптів та HTML-тегів, що вимагає детальної структуризації, щоб забезпечити її підтримку та масштабування, а також для вдосконалення розробки. У традиційних вебсайтах одниничними блоками повідомлень є сторінки. У односторінкових додатках потрібно використовувати інший метод розмежування - компонування на компоненти.

Компонент має власні ресурси та поведінку; він може мати залежності від інших компонентів, або також містити їх. Компонування на компоненти - простий і достатньо розповсюджений спосіб структурної організації використовуваний в багатьох фреймворках графічного інтерфейсу [22].

3.2.3 Модульність

Обґрунтовуючись на тому, що було вказано раніше, односторінковим додаткам потрібно велика кількість ресурсів, скриптів та HTML-коду, то перше завантаження додатку може занадто витратити багато часу. Тому рекомендується здійснити динамічне підвантаження ресурсів лише в тому випадку, коли вони є потрібними.

3.2.4 API модуль

Щоб здійснити складні та супроводжувані базою даних дії додатку, рекомендовано створити окремий компонент, який буде відповідати за виконання дій на сервері. Більшість операцій потребує авторизації, що також можна вирішити, реалізуючи у компоненті.

3.2.5 Інверсія управління

По мірі зростання кількості бібліотек та фреймворків, які використовуються для створення односторінкових додатків, реалізація принципу інверсії управління, відокремлюючи створення залежностей від логіки клієнта і дозволяючи компонентам бути слабко зв'язаними, має все більше сенсу [23].

За даним принципом інверсії управління, звертаючись до сервісу, клієнт не повинен займатися цим автономно, а через посередника (рис. 3.1).



Рисунок 3.1 – Принцип інверсії управління

Цей принцип має 3 реалізації: впровадження залежностей, фабрики та локатори сервісів.

Ангуляр 2 - поточний єдиний фреймворк, що вимагає використання даного шаблону при розробці.

3.2.6 Графічний інтерфейс

HTML задає вигляд вебсайту. У звичайних сайтах, HTML-розмітка створюється сервером під час обробки запита. Однак, для односторінкових додатків необхідно уникати зайвих запитів до сервера, отже окрім основного HTML сторінки, що ми отримуємо під час відкриття додатка, решта HTML-розмітки генерується на стороні клієнта скриптами [24].

З появою WEB, появилось багато фреймворків та технологій які полегшують створення HTML на сервері. Ми постійно зустрічаємо нові вимоги з моделювання класів і архітектури клієнтських додатків, так що необхідно використовувати бібліотеки для генерації графічного інтерфейсу клієнта із скриптів. Ці технології необхідні для створення потужного односторінкового додатка що швидко підтримується та масштабується.

3.2.7 Шаблони

Одною з проблем, з якою ми стикаємося при розробці графічного інтерфейсу з використанням різних фреймворків, є передбачення повторного використання та структуризації коду. Одним з ефективних способів досягти цього - використання шаблонів. Це полягає в єдиному HTML-шаблоні для змінення певних частин у відповідності до особливостей кожного певного екземпляру [25]. Можливості шаблонів залежать від вибраного фреймворку для розробки, таких як Angular/Angular2, Knockout, Ember тощо.

3.2.8 Зв'язування

Наступним кроком у створенні графічного інтерфейсу є забезпечення синхронізації або зв'язку між даними та їх відображенням. Існують два типи зв'язків:

- односторонній зв'язок - коли зміни у джерелі призводять до змін у відображенні;
- двосторонній зв'язок - коли зміни у джерелі впливають на відображення, а також зміни у відображенні впливають на джерело.

При великій кількості елементів, які потребують зв'язку, рекомендується використовувати готові фреймворки. Вони дозволяють зменшити кількість коду і вирішити складні задачі зв'язку, такі як управління залежностями між зв'язками та автоматичне поширення змін по всьому дереву зв'язків.

Існує багато фреймворків, що підтримують зв'язування, включаючи:

- Angular/Angular2;
- Knockout;
- Ember.

Ці фреймворки надають можливості для ефективного управління зв'язками даних та полегшують роботу зі складними структурами зв'язків.

3.3 Шаблони MVC та MVVM

При розробці графічного інтерфейсу часто застосовують популярні шаблони: MVC (Model-View-Controller), (рис. 3.2) та його більш сучасний варіант MVVM (Model-View-ViewModel), (рис. 3.3).

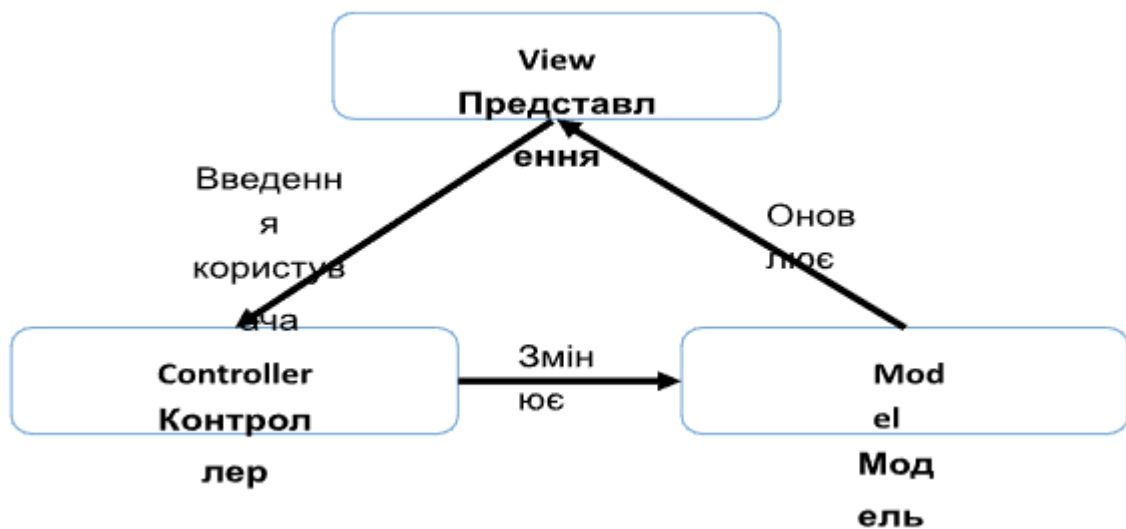


Рисунок 3.2 – Концепція MVC

Концепція MVC була створена ще у 80-ті роки минулого століття та спочатку була пов'язана з графічним інтерфейсом того періоду [26].

Основна мета MVC - розділити код на три рівні відповідно до їх функціональності:

- модель (Model) — представляє дані та бізнес-логіку предметної області;
- представлення (View) — відображає модель;
- контролер (Controller) — обробляє введення користувача та оновлює модель.

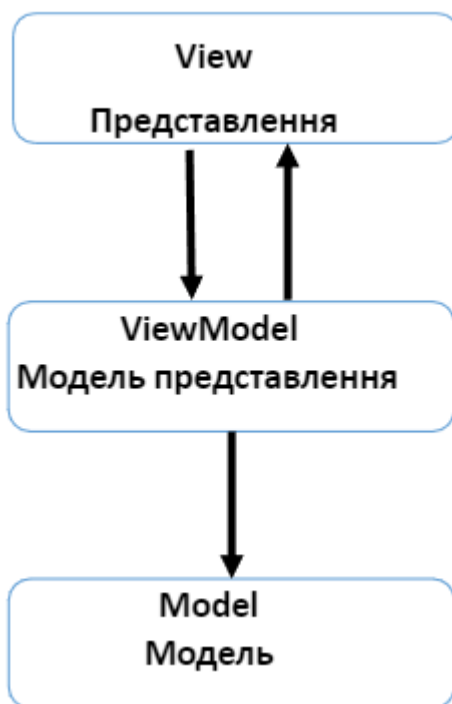


Рисунок 3.3 – Концепція MVVM

MVVM – це сучасна модифікація MVC, яка більш адаптована до сучасних вимог розробки графічних інтерфейсів [27]. У MVVM контролер відповідає як за введення користувача, так і за оновлення представлення та має назву моделі представлення (ViewModel):

- модель (Model) – представляє дані та бізнес-логіку предметної області;
- модель представлення (ViewModel) – абстрактне відображення представлення;
- представлення (View) – відображає модель представлення та пересилає користувацьке введення моделі представлення.

3.4 Висновки по третьому розділу

Розглянувши типові особливості архітектури односторінкових додатків, можна зробити висновок, що ця архітектура успадковує певні особливості архітектури графічних інтерфейсів нативних додатків. Багато

шаблонів проектування, розроблених для нативних додатків, з невеликими адаптаціями знаходять своє застосування при розробці односторінкових вебдодатків.

4 ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ РОЗРОБКИ ВЕБДОДАТКУ З ВИКОРИСТАННЯМ ANGULAR

4.1 Підготовка прикладу

Для початку роботи з фреймворком Angular рекомендується встановити Node.js (рис. 4.1). Цей фреймворк працює на платформі Node.js, тому потрібно мати встановлену його версію 5.x.x або новішу, а також npm (Node Package Manager) версії 3.x.x або новішу [28].

Встановити Node.js можна завантаживши інсталятор з офіційного сайту Node.js: <https://nodejs.org/>. Після встановлення перевірте версії Node.js та npm, введіть команди у консолі:

- `node -v`;
- `npm -v`.

Це дозволить вам переконатися, що ви працюєте з відповідними версіями. Після цього ви зможете продовжити розгортання прикладу вебдодатку за допомогою фреймворку Angular, як описано в офіційній документації чи інструкціях, які використовувалися для підготовки прикладу.

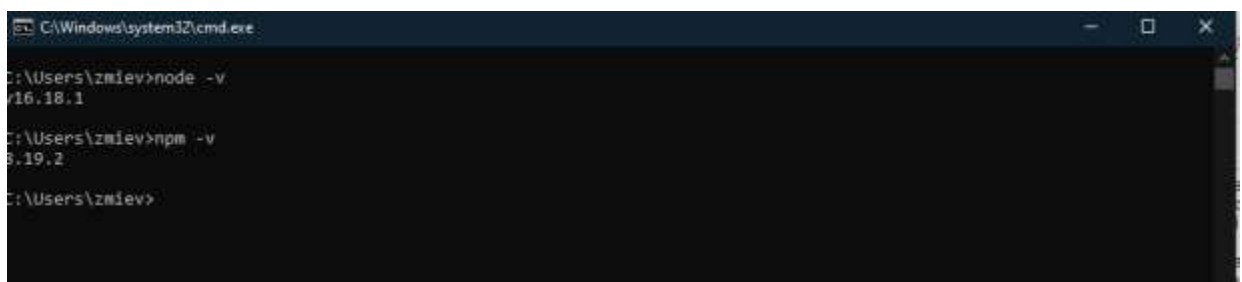


Рисунок 4.1 – Команди перевірки встановленої версії Node.js та npm

Інші залежності можна буде автоматично встановлювати завдяки вбудованому менеджеру пакетів npm (рис. 4.2).

.angular	07.09.2023 21:19	Папка с файлами	
.firebase	07.08.2023 23:51	Папка с файлами	
.git	10.10.2023 18:04	Папка с файлами	
dist	03.11.2023 20:23	Папка с файлами	
e2e	07.12.2022 11:58	Папка с файлами	
node_modules	05.10.2023 19:44	Папка с файлами	
src	11.08.2023 12:24	Папка с файлами	
.firebaserc	07.08.2023 23:41	Файл "FIREBASERC"	1 КБ
.gitignore	07.09.2023 21:10	Текстовый докум...	2 КБ
angular.json	05.10.2023 21:59	Исходный файл J...	4 КБ
firebase.json	07.08.2023 23:42	Исходный файл J...	1 КБ
firestore.indexes.json	11.06.2022 11:09	Исходный файл J...	1 КБ
firestore.rules	07.08.2023 21:25	Файл "RULES"	1 КБ
karma.conf.js	11.06.2022 10:43	Исходный файл J...	2 КБ
package.json	05.10.2023 19:44	Исходный файл J...	2 КБ
package-lock.json	05.10.2023 19:44	Исходный файл J...	1 187 КБ
README.md	11.06.2022 10:43	Исходный файл ...	1 КБ
tailwind.config.js	01.09.2023 15:10	Исходный файл J...	3 КБ
tsconfig.app.json	11.06.2022 10:43	Исходный файл J...	1 КБ
tsconfig.json	01.09.2023 11:48	Исходный файл J...	1 КБ
tsconfig.spec.json	11.06.2022 10:43	Исходный файл J...	1 КБ
tslint.json	07.12.2022 11:58	Исходный файл J...	4 КБ

Рисунок 4.2 – Файлова структура прикладу

Лістинг всіх файлів прикладу наведений в додатку А. Файли відповідні за роботу серверної частини:

- «package.json» – файл пакетного менеджера npm, який містить перелік та версії залежностей які необхідні для роботи прикладу, а також деякі команди для збірки та запуску серверу [29]-[32];

- «tsconfig.json» – конфігурація компілятора TypeScript;

- «firebase.json» – конфігурація Firebase;

- «tailwind.config.json» – конфігурація Tailwind;

- «typings.json» – ідентифікує TypeScript визначення;

- «index.html» – головна сторінка вебдодатку яка містить лише основні теги та скрипти які асинхронно завантажать потрібні ресурси для повного відображення додатку;

- «main.ts» – містить код який завантажує інфраструктуру фреймворка Angular, сервіси та запускає додаток з головний компонентом «AppComponent». За це відповідає функція «bootstrap»;

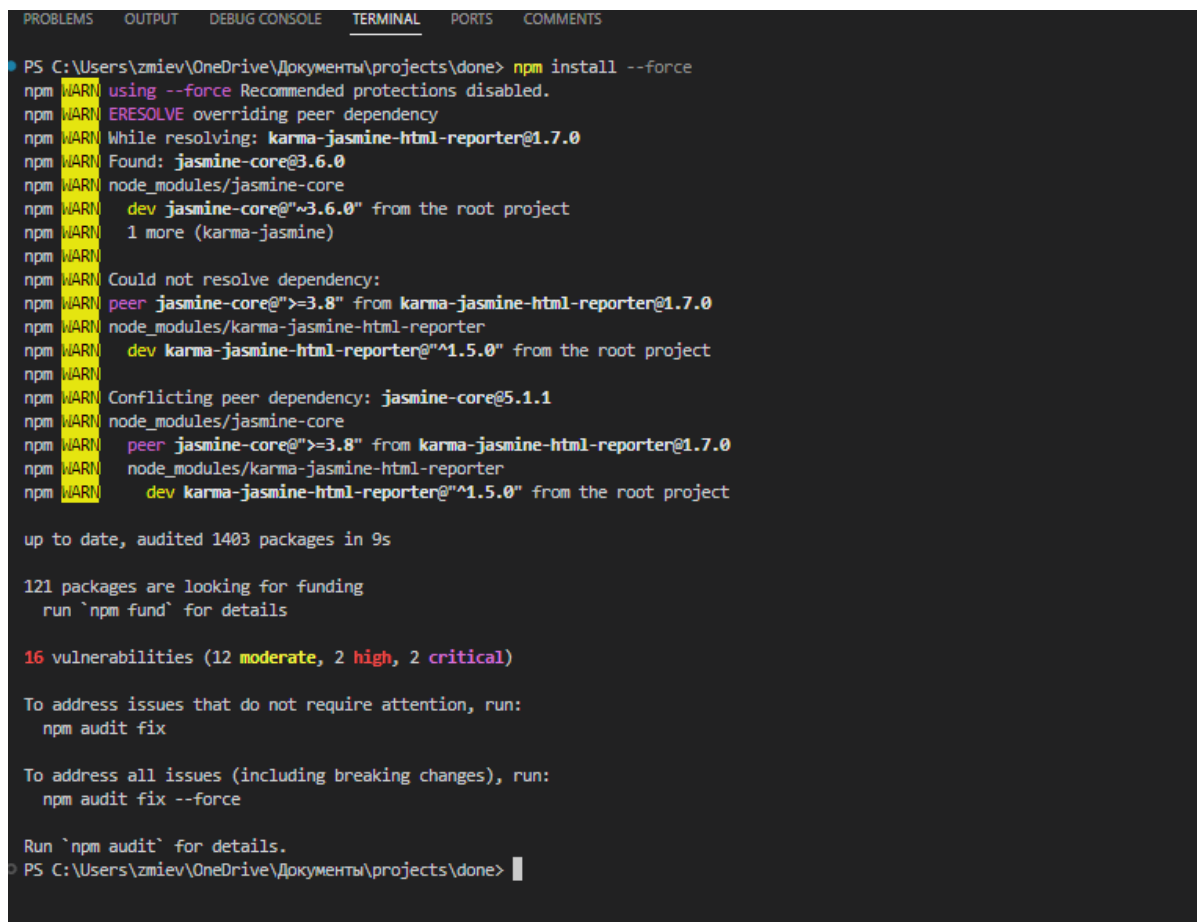
- «styles.css» – містить загальні спільні стилі необхідні для роботи додатку.

Кожен компонент має «*.ts» файл який містить його TypeScript код, та може мати «*.html» та «*.scss/css/less» файли що містять HTML шаблон та CSS стилі компонента відповідно. За рекомендованими нормами кодування Angular компоненти мають містити у назві «component».

4.2 Збірка та запуск

Управління Node.js та npm вимагає використання консолі. Тому, спочатку потрібно відкрити консоль і перейти до теки з вашим проєктом.

При першому запуску програми або після внесення змін у файл «package.json», необхідно виконати процедуру відновлення залежностей. Це дозволить завантажити потрібні версії всіх необхідних компонентів. Для здійснення цієї процедури потрібно ввести команду «npm install» у консолі (рис. 4.3).



```

PS C:\Users\zmiev\OneDrive\Документи\projects\done> npm install --force
npm WARN using --force Recommended protections disabled.
npm WARN ERESOLVE overriding peer dependency
npm WARN While resolving: karma-jasmine-html-reporter@1.7.0
npm WARN Found: jasmine-core@3.6.0
npm WARN   node_modules/jasmine-core
npm WARN   dev jasmine-core@"~3.6.0" from the root project
npm WARN   1 more (karma-jasmine)
npm WARN Could not resolve dependency:
npm WARN peer jasmine-core@">=3.8" from karma-jasmine-html-reporter@1.7.0
npm WARN   node_modules/karma-jasmine-html-reporter
npm WARN   dev karma-jasmine-html-reporter@"^1.5.0" from the root project
npm WARN Conflicting peer dependency: jasmine-core@5.1.1
npm WARN   node_modules/jasmine-core
npm WARN peer jasmine-core@">=3.8" from karma-jasmine-html-reporter@1.7.0
npm WARN   node_modules/karma-jasmine-html-reporter
npm WARN   dev karma-jasmine-html-reporter@"^1.5.0" from the root project

up to date, audited 1403 packages in 9s

121 packages are looking for funding
  run `npm fund` for details

16 vulnerabilities (12 moderate, 2 high, 2 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
PS C:\Users\zmiev\OneDrive\Документи\projects\done>

```

Рисунок 4.3 – Результат роботи команди «npm install»

Для запуску серверу необхідно в консолі ввести команду «ng serve» (рисунок 4.4).

```

npm WARN ERESOLVE overriding peer dependency
npm WARN While resolving: karma-jasmine-html-reporter@1.7.0
npm WARN found: jasmine-core@3.6.0
npm WARN node_modules/jasmine-core
npm WARN dev jasmine-core@3.6.0 from the root project
npm WARN 1 more (karma-jasmine)
npm WARN Could not resolve dependency:
npm WARN peer jasmine-core@>3.8 from karma-jasmine-html-reporter@1.7.0
npm WARN node_modules/karma-jasmine-html-reporter
npm WARN dev karma-jasmine-html-reporter@1.3.0 from the root project
npm WARN Conflicting peer dependency: jasmine-core@5.1.1
npm WARN node_modules/jasmine-core
npm WARN peer jasmine-core@>3.8 from karma-jasmine-html-reporter@1.7.0
npm WARN node_modules/karma-jasmine-html-reporter
npm WARN dev karma-jasmine-html-reporter@1.5.0 from the root project

up to date, audited 1493 packages in 5s

121 packages are looking for funding
  run `npm fund` for details

5 vulnerabilities (12 moderate, 2 high, 2 critical)
To address issues that do not require attention, run:
  npm audit fix
To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
PS C:\Users\valer\OneDrive\Documents\projects\docs> ng serve
  Browser application bundle generation complete.

Initial Chunk Files | Names | Raw Size
---|---|---
vendor.js           | vendor | 8.76 MB
main.js             | main   | 1.19 MB
styles.css, styles.js | styles | 493.22 kB
polyfills.js        | polyfills | 115.96 kB
runtime.js          | runtime | 12.96 kB

Initial Total | 10.73 MB

Lazy Chunk Files | Names | Raw Size
---|---|---
src_app_layout_layout_module_ts.js | layout-layout-module | 382.07 kB

Build at: 2023-12-04T13:32:43.402Z - Hash: 24623f9dc9017c3e - Time: 5256ms

Warning: C:\Users\valer\OneDrive\Documents\projects\docs\src\app\salary-page\salary-page.component.ts depends on 'lodash/cloneDeep'. CommonJS or AMD dependencies can cause optimization bailouts.
For more info see: https://angular.io/guide/build#configuring-commonjs-dependencies

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

Compiled successfully.

```

Рисунок 4.4 – Вивід консолі при успішному запуску серверу

Після успішного запуску сервера вебсайт буде доступним за адресами які виведені у консолі. Також за додатковими адресами, які теж виведені у консолі, буде доступний графічний інтерфейс керування сервером (рис. 4.5).

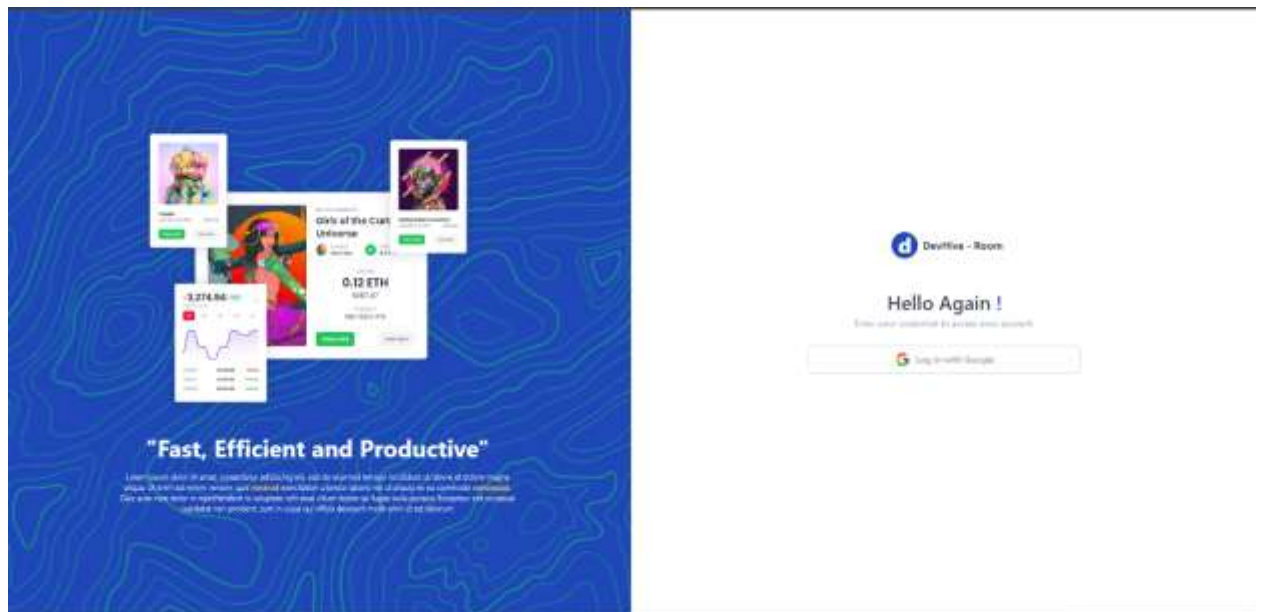


Рисунок 4.5 - Графічний інтерфейс

4.3 Модулі та TypeScript

Angular рекомендує використання модульного підходу, особливо модулів ES2015, які TypeScript підтримує добре, хоча це не обов'язково. TypeScript можна замінити на звичайний JavaScript, оскільки будь-який код TypeScript може бути компільований у JavaScript [32]-[36]. Використання модулів та TypeScript значно полегшує написання, структурування та підтримку коду, тому відмовлятися від них не рекомендується.

Модулі є аналогом простору імен у мовах програмування, таких як C# або C++. Вони включають в себе внутрішній та зовнішній код (тобто класи, функції, значення), які можуть бути використані іншими модулями. Для позначення зовнішнього коду, що експортується модулем, використовується ключове слово «export», наприклад:

```
```typescript
export class AppComponent { }
```
```

Щоб імпортувати частину з іншого модулю, використовується наступний синтаксис:

```

```typescript
import { AppComponent } from './app.component';
```

```

Структура (рис. 4.6) вебдодатка Angular базується на компонентах, директивах та сервісах [37]. Компоненти відповідають за контроль та управління конкретною частиною DOM-елементів, включаючи логіку їхнього відображення та поведінку. Директиви, на відміну від компонентів, визначають лише певну поведінку. Сервіси призначені для зручного впровадження єдиного екземпляра класу у компонентах та директивах додатку, які від них залежать. Отже, сервіси втілюють шаблон проєктування «Одинак».

Для визначення класу як компонента, директиви або сервісу в Angular використовуються метадані у вигляді функцій-декораторів, які розміщуються перед класом із необхідними параметрами.



Рисунок 4.6 – Архітектура Angular 2 додатку

Компоненти в Angular можуть використовувати інші компоненти у вигляді дочірніх, що створює зручну ієрархію елементів в додатку. Зв'язування між батьківськими та дочірніми компонентами дозволяє

взаємодіяти між ними та передавати дані від одного до іншого, що спрощує роботу з даними та управління елементами в додатку (рис. 4.7).

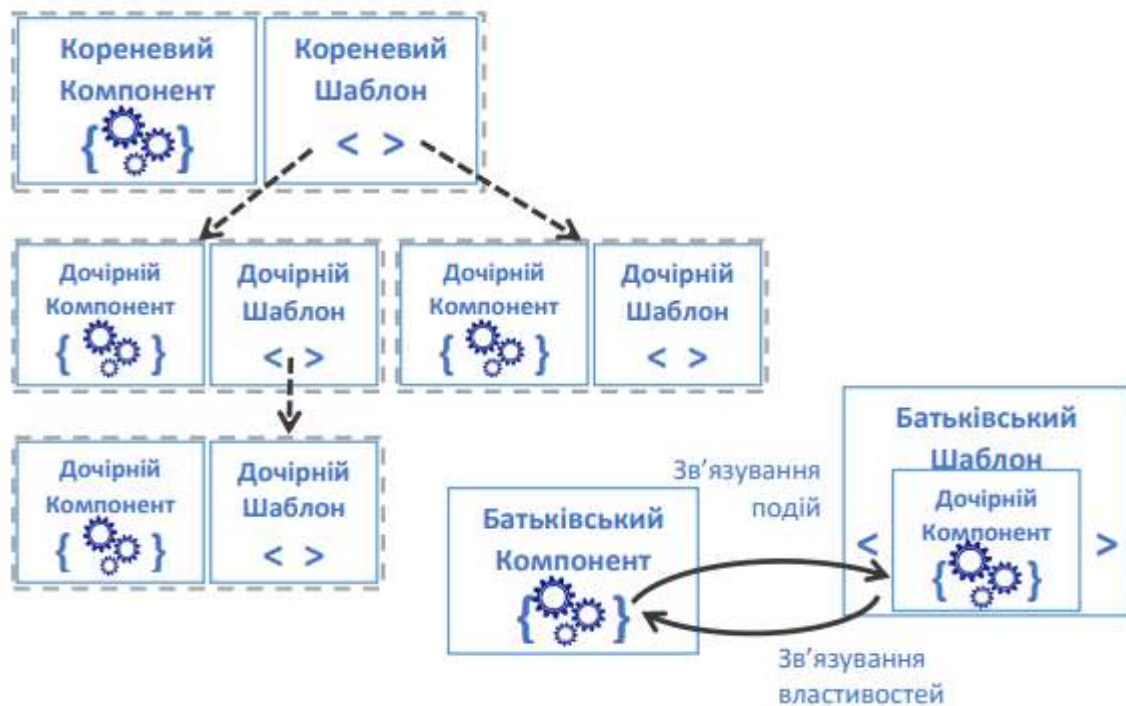


Рисунок 4.7 – Ієрархія та зв'язок компонентів Angular 2 додатку

4.4 Компоненти

Для того щоб створити компонент треба обернути клас який буде містити поведінку компонента у функцію-декоратор «@Component». Розглянемо наступний приклад компоненту «NavbarComponent» (рис. 4.8):

- «selector» - визначає CSS-селектор, за яким фреймворк вибирає HTML-елементи, які відтворюватимуть компонент. Ці елементи стануть контейнерами для компонента. Селектор у прикладі вказує фреймворку відтворити кожен тег «<app-navbar></app-navbar>» як компонент «NavbarComponent»;

- «templateUrl» – вказує посилання на HTML-шаблон, який буде відтворений компонентом. Для невеликих шаблонів зручно використовувати альтернативний параметр «template»;

- «directives» – визначає модулі, які використовуються в компоненті;
- «providers» – вказує необхідні для компонента сервіси. Екземпляри реалізації цих сервісів фреймворк передасть у конструктор компонента;
- «standalone» – використовується для позначення того, що компонент може існувати незалежно, без прив'язки до інших компонентів;
- «imports» – вказує на імпорт інших модулів або компонентів, які використовуються в даному компоненті для підтримки його функціональності.

```
@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.scss'],
  standalone: true,
  imports: [
    AngularSvgIconModule,
    NavbarMenuComponent,
    ProfileMenuComponent,
    NavbarMobileComponent,
  ],
})
```

Рисунок 4.8 – Компонент Angular 2 додатку

4.5 Директиви

Angular містить розширений набір вбудованих директив, які дозволяють зручно керувати виглядом компонентів через шаблони [38]-[42]. Проте, якщо виникає потреба, користувач може створити свої власні директиви для специфічних вимог чи функціональності.

Angular дозволяє зв'язувати властивості, атрибути та події елементів у шаблоні компонента з властивостями та методами компонента, а також його дочірніх компонентів. Це відбувається декларативно, використовуючи спеціальний синтаксис:

- інтерполяція: Дозволяє односторонньо оновлювати властивості та вміст DOM-елементів при зміні значення компонента;
- зв'язування властивостей: Одностороннє зв'язування властивостей DOM-елементів або дочірніх компонентів з властивостями компонента;
- зв'язування подій: Зв'язування подій DOM-елементів з методами компонента;
- двостороннє зв'язування: Дозволяє обидвосторонньо зв'язати властивості DOM-елементів з властивостями компонента;
- зв'язування атрибутів: Задає значення атрибуту DOM-елементу при його створенні;
- зв'язування класу та стилю: Дозволяє динамічно міняти класи та стилі DOM-елементу;

Структурні директиви Angular:

- умовна директива (`*ngIf`): Додає або видаляє елемент DOM в залежності від значення логічного виразу (рисунк 4.9);
- директива `*ngSwitch`: Змінює елемент DOM в залежності від значення виразу;
- директива `*ngFor`: Створює елемент DOM для кожного елемента списку (рис. 4.10).

Ці можливості Angular дозволяють зручно та ефективно керувати компонентами та їх взаємодією з DOM-елементами.

```
<button (click)="toggleMenu()" type="button" clickOutside (clickOutside)="isMenuOpen = false" aria-expanded="false"
  aria-haspopup="true"
  class="flex bg-primary-500 relative w-8 h-8 justify-center items-center m-1 mr-2 text-xl rounded-full text-white">
  <img *ngIf="user?.avatar" class="rounded-full" [src]="user.avatar">
  <span *ngIf="!user?.avatar">{{ user?.fullName?.[0] }}</span>
</button>
```

Рисунок 4.9 – На рисунку частина коду з «`ngIf`»

```

<div class="dropdown relative inline-block" *ngFor="let menu of menuService.pagesMenu">
  <!-- Button -->
  <button
    [ngClass]="
      menu.selected || menu.active
      ? 'bg-primary-500 text-white'
      : 'text-gray-600 hover:bg-gray-100 hover:text-primary-500 dark:text-night-200 dark:hover:bg-night-600 '
    "
    class="mr-2 inline-flex rounded-md px-3 py-2 text-sm font-medium">
    <span>{{ menu.group }}</span>
  </button>
  <!-- Dropdown -->
  <div
    class="dropdown-content absolute top-[100%] min-w-[200px] origin-top-left"
    navbar-submenu
    [submenu]="menu.items"></div>
</div>

```

Рисунок 4.10 – На рисунку частина коду з «ngFor»

Приклад атрибутивної директиви зображено на рисунку 4.11.

```

<span class="text-sm text-gray-400 dark:text-night-300">
  <div class="relative test-gray-600 #focus-within:test-gray-400">
    <input
      class="bg-white appearance-none border-1 border-gray-200 rounded py-2 px-4 text-gray-700 leading-tight focus:outline-none"
      type="number" [(ngModel)]="currency">
    </div>
  </span>

```

Рисунок 4.11 – На рисунку частина коду з «[(...)]="..."»

Для створення власної директиви потрібно створити клас, в якому міститься поведінка цієї директиви, і огорнути цей клас декоратором «@Directive» [43].

4.6 Сервіси

Для того щоб створити реалізацію сервісу треба обернути клас який буде містити поведінку сервісу у функцію-декоратор «@Injectable» (рис. 4.12).

Приклад реалізації сервісу авторизації.

```

import { UserService } from '../src/app/users/users.service';
import { getAuth } from '@angular/fire/auth';
import { Injectable } from '@angular/core';
import { Auth, OAuthProvider, signInWithPopup, signOut, UserCredential } from 'firebase/auth';
import { Firestore } from '@angular/fire/firestore';
import { first, mergeMap } from 'rxjs/operators';
import { CrudService } from 'src/app/db/shared/crud.service';
import { User } from 'src/app/db/users/user.model';
import { of } from 'rxjs';

@Injectable({
  providedIn: 'root'
})

export class AuthService extends CrudService<User> {
  collectionName = 'users';
  users: User;
  userId: string;
  testId: string;

  constructor(
    public firestore: Firestore,
    private userService: UserService,
  ) {
    super(firestore);
  }

  async loginWithGoogle(): Promise<UserCredential> {
    const auth: Auth = getAuth();
    const provider: OAuthProvider = new OAuthProvider(providers.google);
    return await signInWithPopup(auth, provider).then((authResult: UserCredential) => this.loginSuccessHandler(authResult));
  }

  async signOut(): Promise<any> {
    const auth: Auth = getAuth();
    await signOut(auth);
    if (localStorage.getItem('cache')) {
      localStorage.removeItem('cache');
      window.location.reload();
    }
  }
}

```

Рисунок 4.12 – На рисунку частина коду з сервісу авторизації

4.7 Клієнтська маршрутизація

Маршрутизація в Angular надає можливість керувати навігацією в додатку. Для цього використовується модуль `RouterModule`, який дозволяє визначити маршрути та пов'язати їх з компонентами. Ключовий компонент для роботи з маршрутизацією - `Router`, який відстежує стан маршрутування в додатку [44].

Маршрути в Angular описуються за допомогою масиву об'єктів, де кожен об'єкт представляє окремий маршрут. Кожен маршрут має властивості, такі як шлях (path), який вказує на URL, компонент, який буде відображатися при цьому маршруті, а також додаткові параметри.

Щоб використовувати маршрути в додатку, потрібно імпортувати `RouterModule` у головний модуль додатку та вказати масив маршрутів у функції `forRoot()`. Для навігації між різними маршрутами використовується

директива ``routerLink`` у шаблонах компонентів, яка створює посилання на відповідні маршрути.

Параметри маршрутів дозволяють передавати дані в URL, що дозволяє робити динамічну навігацію. Для отримання параметрів використовується об'єкт ``ActivatedRoute``, який дозволяє отримувати доступ до параметрів та стану маршруту.

Маршрутизація в Angular - потужний інструмент для створення інтуїтивних та динамічних вебдодатків з можливістю переходу між різними частинами додатку за допомогою URL-адреси.

Приклад налаштування зображено на рисунку 4.13.

```
{
  path: 'users',
  canActivate: [IsLoggedInGuard, IsAdminGuard],
  children: [ ...
]
},
{
  path: 'reyle-now',
  canActivate: [IsAdminGuard],
  component: ReyleNowComponent,
  children: [ ...
]
},
{
  path: 'timeline',
  canActivate: [IsLoggedInGuard],
  component: TimelineComponent,
  children: [ ...
]
},
{
  path: 'salary',
  canActivate: [IsLoggedInGuard, IsAdminGuard],
  component: SalaryPageComponent,
  children: [ ...
]
},
},
```

Рисунок 4.13 – Частина коду з маршрутизації

4.8 Висновки по четвертому розділу

На основі аналізу фреймворку Angular та робочого прикладу, розробленого за його допомогою, можна зробити декілька висновків. Фреймворк виявився вкрай гнучким та зручним для створення односторінкових вебдодатків, пропонуючи багато можливостей інтеграції та функціоналу.

Під час дослідження були розглянуті ключові особливості фреймворку, включаючи структуру додатку, програмне забезпечення для його роботи та процес створення та запуску. Офіційні приклади використання дозволили поглиблено зрозуміти функціонал Angular та способи його застосування.

Активна розробка та регулярні виправлення можуть свідчити про перспективи розвитку фреймворку та його удосконалення. Таким чином, Angular є потужним інструментом для розробки вебдодатків.

5 РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ З РОХРАХУНКУ ЗАРОБІТНОЇ ПЛАТИ

Реалізація додатку, розробленого на Angular, Firebase та використання Tailwind, є інтегрованою платформою, яка забезпечує різноманітні можливості для користувачів. Ця аплікація включає багатофункціональний інтерфейс, який створено для зручності та ефективності користувача [44]-[46].

Після входу до додатку користувач зустрічає сторінку авторизації через Google, реалізовану завдяки Firebase. У цьому місці було використано функціонал Firebase для створення захисту через гарди та інтерсептори для перевірки авторизації, щоб гарантувати, що користувачеві буде доступ тільки після успішної автентифікації.

Після авторизації користувачі мають доступ до чотирьох основних сторінок:

5.1 Контролю часу на виконання завдань

Це розділ, де користувачі можуть реєструвати свій робочий час. На цій сторінці вони можуть зберігати час, який вони проводять на роботі. Для кращої зручності та певної впевненості, тут можна використовувати асинхронні запити до Firebase для зберігання цих даних.

Графічний інтерфейс (рис. 5.1) було створено на основі елементів з tailwind.

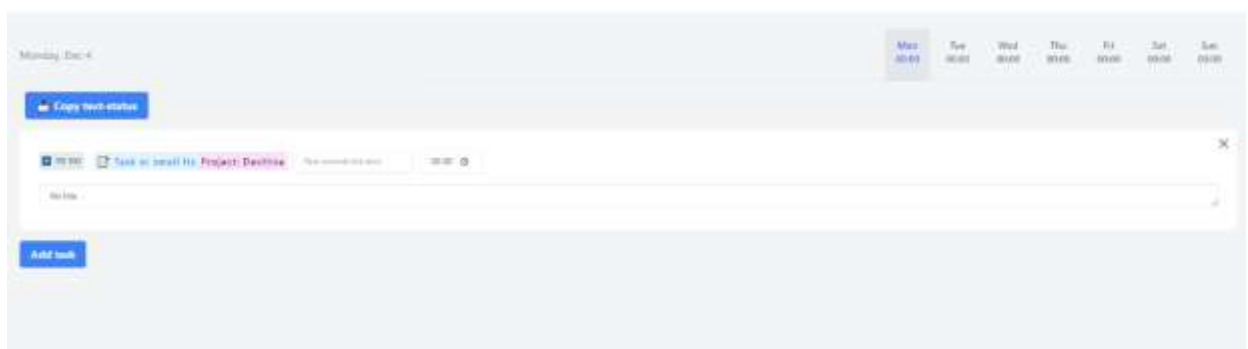


Рисунок 5.1 – Інтерфейс сторінки Time

При створенні нової задачі (рис. 5.2) для логування використовується компонент «TimeLine», де генерується об'єкт та зберігається у Firebase Database.

```
add() {
  const newCheck = new Check({
    day: this.list.dayString,
    focused: true,
    assigneeId: this.user.id,
    status: TaskStatus.ToDo
  });
  this.list.tasks.push(newCheck);

  from(this.checksService.create(newCheck))
    .pipe(
      this.toast.observe(
        {
          loading: 'Creating...',
          success: (s) => 'Task created! ',
          error: (e) => 'Something did not work',
        }
      )
    )
    .subscribe((data) => {
      newCheck.id = data.id;
      this.log({ entityId: newCheck.title, type: LogType.Add });
    });
}
```

Рисунок 5.2 – Створення задачі

Після успішного створення з'являється інформаційне вікно.

5.2 Календарь для додавання відсутностей

На цій сторінці користувачі можуть встановлювати дати своїх відпусток або вихідних. Це може бути відображено за допомогою календаря (рис. 5.3), де вони можуть зробити відповідні записи, що зберігаються в Firebase Realtime Database.

При створенні запиту генерується об'єкт та зберігається в БД. Також відбувається перерахунок бонусу та заробітної плати і одразу записується до БД. З урахуванням всіх доповнень (рисунок 5.4).

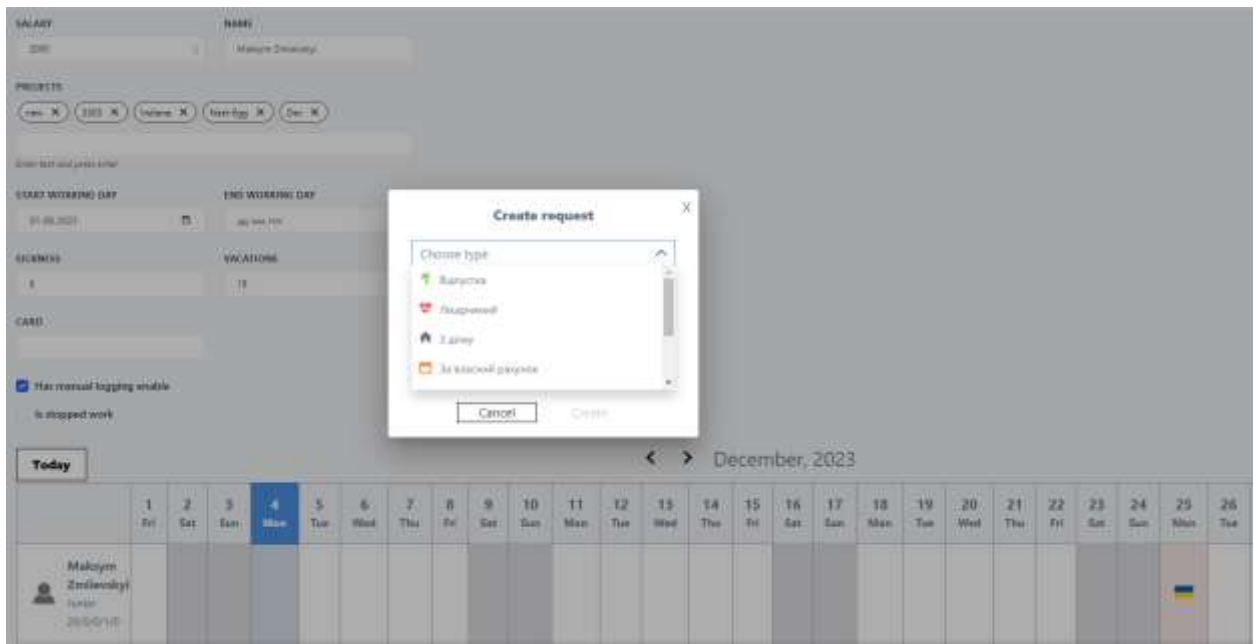


Рисунок 5.3 – Інтерфейс сторінки Calendar

```

sendHolidayRequest(chosenType: HolidayType, dateFrom: string, dateTo: string, comment: string, user: User) {
  const holiday = {
    type: chosenType,
    userId: user.id,
    dateFrom,
    dateTo,
    date: [dateFrom.slice(0, 7), dateTo.slice(0, 7)],
    year: [dateFrom.slice(0, 4), dateTo.slice(0, 4)],
    comment,
  }
  this.handleSameHolidaysDates(holiday);
}

handleSameHolidaysDates(holiday: any) {
  const userHolidaysObservables = this.holidaysService.getByArrayField('date', holiday.date);
  combineLatest([userHolidaysObservables]).pipe(take(1)).subscribe((userHolidays: any) => {
    let isSameHolidays = false;
    let sameHolidays = [];
    userHolidays[0] = userHolidays[0].filter(userHoliday => userHoliday.userId === holiday.userId)
    userHolidays[0].forEach(userHoliday => {
      const requestHolidayIntervalLength = datesDifference(new Date(holiday.dateFrom), new Date(holiday.dateTo));
      const sameHolidayIntervalLength = datesDifference(new Date(userHoliday.dateFrom), new Date(userHoliday.dateTo));
      const requestHolidayInterval = getDatesInterval(holiday.dateFrom, requestHolidayIntervalLength);
      const sameHolidayInterval = getDatesInterval(userHoliday.dateFrom, sameHolidayIntervalLength);
      requestHolidayInterval.forEach(date => {
        sameHolidayInterval.forEach(sameDate => {
          if (date === sameDate) {
            sameHolidays.push(true);
          }
        })
      })
      isSameHolidays = sameHolidays.length > 0 ? true : false;
    });
    if (isSameHolidays) {
      alert('Request wasn't sent. You already have a vacation for this time');
      return;
    }
    this.holidaysService.create(new Holiday(holiday)).then(() => {
      alert('Your request was sent ★');
      this.changeDetectorRef.detectChanges();
    });
  });
}

```

Рисунок 5.4 – Зберігання робочих днів

5.3 Управління заробітної плати

Тут адміністратор може переглянути розрахунок зарплати користувачів, враховуючи відпустки, вихідні тощо. Окрім цього, система враховує час, який користувач провів на роботі, для розрахунку можливого бонусу. Розрахунок може проводитись за допомогою штучного інтелекту (рис. 5.5).

| EMPLOYEE | POSITION | SALARY | BONUS | TOTAL | CURRICULUM | CHIEF |
|---------------------|----------|--------------------|-------|--------------------|------------|-------|
| Maksym Zivodnyk | Junior | 1000 | 0 | 1000.00 | | 100% |
| Viktoriya Ivanenko | Junior | 800 | 0 | 800.00 | | 100% |
| Tetiana Koval | Senior | 2000 | 0 | 2000.00 | | 100% |
| Test User 2 | Senior | 2000 | 0 | 2000.00 | | 100% |
| Paula Perekhoda | Junior | 800 | 0 | 800.00 | | 100% |
| Test | Junior | 1000 | 0 | 1000.00 | | 100% |
| Total salary | | \$14,698.00 | | \$14,698.00 | | |

Рисунок 5.5 – Інтерфейс сторінки Salary

При зборі даних з БД, обробляється бонус та «Calculated Salary» (рис. 5.6), на основі відсутності та витраченого робочого часу.

```
getSalaryReports(holidays: Holiday[], users: User[], month: Date) {
  month.setHours(10);
  const date = month.toISOString().slice(0, 7);
  const salaryReportsObservables = users.map(user => this.reportsService.getbyTwoFields('userId', user.id, 'date', date))
  combineLatest(salaryReportsObservables).pipe(take(1)).subscribe((reports: Report[]) => {
    reports.forEach((report) => {
      if (!report[0]) {
        return;
      }
      const userIndex = users.findIndex(user => user.id === report[0].userId);
      this.usersReports[userIndex].salaryReport = report[0];
    });

    let emptyUsersReports = this.usersReports.map((userReport) => {
      if (!userReport.salaryReport.userId) {
        return userReport;
      }
    }).filter(report => !report);
    if (emptyUsersReports.length) {
      let createReportsPromises = emptyUsersReports.map(emptyUserReport => this.createSalaryReport(emptyUserReport.userId, date));
      Promise.all(createReportsPromises).then(() => this.getSalaryReports(holidays, users, month));
    } else {
      this.getUsersLabels(holidays, users, month);
    }
  });
}
```

Рисунок 5.6 – Код «Calculated Salary»

Також ШІ вираховує бонус який ми можемо надати користувачу, поле «Bonus» можна редагувати натиснувши на нього (рис. 5.7).

Salary page

December, 2022

| | WORKDAYS | SICKNESS | VACATIONS | WARTIME | DISCOUNT | PLANNED SALARY | BONUS | UNDETERMINED SALARY | COMMENT | CARD |
|-----------------------|----------|----------|-----------|---------|----------|----------------|-------|---------------------|---------|------|
| M Maksym Zimbalskyi | 2102 | 0 | 0 | 0 | 0 | 1000 | 0 | \$2,000.00 | | Copy |
| V Valeria Sazachanska | 2102 | 0 | 0 | 0 | 0 | 100 | 0 | \$100.00 | | Copy |
| B Test User 1 | 2102 | 0 | 0 | 0 | 0 | 1000 | 0 | \$2,000.00 | | Copy |
| I Test User 2 | 2102 | 0 | 0 | 0 | 0 | 1000 | 0 | \$2,000.00 | | Copy |
| P Pasha Petelivets | 2102 | 0 | 0 | 0 | 0 | 100 | 0 | \$100.00 | | Copy |
| T Test | 2102 | 0 | 0 | 0 | 0 | 100 | 0 | \$100.00 | | Copy |

\$14,699.00 Total salary
 \$14,699.00 Planned salary
 USD to UAH 27.1

Рисунок 5.7 – Інтерфейс редагування бонусу

5.4 Управління користувачами

Це сторінка, де адміністратор може створювати нових користувачів або видаляти старих. Це місце, де робляться зміни в користувацькій базі даних, що також зберігається в Firebase Realtime Database (рис. 5.8).

Create new user

Full name *

Email *

Phone number

Level

Salary, \$

Vacation days

Sick days

Daytime

Birth day

Start working day

End working day

Cancel

Users

| NAME | Salary \$ | Remove |
|---------------------|-----------|--------|
| Maksym Zimbalskyi | 1000 | |
| Valeria Sazachanska | 100 | |
| Test User 1 | 1000 | |
| Test User 2 | 1000 | |
| Pasha Petelivets | 100 | |
| Test | 100 | |

Рисунок 5.8 – Інтерфейс створення користувача

ВИСНОВКИ

У процесі виконання даної дипломної кваліфікаційної роботи були вивчені та проаналізовані сучасні технології та інструменти, спрямовані на оптимізацію систем розрахунку заробітної плати. Виявлено, що використання Angular, Tailwind, Jasmine та Firebase Realtime Database стають ключовими факторами для створення високоефективного та надійного програмного забезпечення для розрахунку заробітної плати.

Архітектура системи була детально розглянута, включаючи розподіл компонентів та їх взаємодію. Побудована структура системи відображає оптимальну організацію функцій та даних для зручного та ефективного управління розрахунками.

Проаналізовано можливості та переваги використання кожної з технологій, зокрема Angular для створення потужного користувацького інтерфейсу, Tailwind для швидкої та ефективної розробки дизайну, Jasmine для виявлення та виправлення помилок у ранніх етапах розробки та Firebase Realtime Database для забезпечення актуальних даних у режимі реального часу.

Основні функціональні вимоги та інтерфейс користувача були докладно розглянуті та виконані, забезпечуючи зручний доступ до основних опцій розрахунку, управління робочим часом та відпустками, а також можливість адміністрування користувачів.

У результаті розробки програмного забезпечення для системи розрахунку заробітної плати використання вищезгаданих технологій забезпечило не тільки високу якість продукту, а й актуальність та ефективність його реалізації. Розроблені рішення відображають потужний потенціал цих технологій у вдосконаленні процесу розробки програмного забезпечення для систем розрахунку, та можуть слугувати важливим внеском у розвиток цієї галузі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. І. М. Петренко, Т. В. Сидоренко. (2015). "Автоматизація процесів обліку та розрахунків заробітної плати в підприємстві". Київ: КНЕУ. – 152 с.
2. Г. С. Яремчук, В. І. Жук. (2010). "Основи теорії та практики обліку заробітної плати". Київ: Центр учбової літератури. – 177 с.
3. Л. О. Дудинець, Л. М. Шапіро. (2016). "Управління заробітною платою та соціальним захистом на підприємстві". Київ: Центр учбової літератури. – 147 с.
4. В. І. Паламарчук. (2012). "Організація обліку та розрахунків заробітної плати в умовах підприємства". Київ: КНТЕУ. – 78 с.
5. А. М. Ковалев, В. М. Шульга. (2017). "Сучасні аспекти розрахунків заробітної плати та оподаткування доходів населення". Львів: ЛНУ імені Івана Франка. – 18 с.
6. Андрійчук В. О. "Оптимізація роботи Angular додатків для забезпечення високої продуктивності." - Херсон: ХНУ, 2018. – 21 с.
7. Інтернет-ресурси Angular Material [Electronic resource]. – Access mode: <https://material.angular.io/>
8. Смирнов В. Ф. "Веброзробка за допомогою Angular: посібник для початківців." - Київ: КВІЦ, 2019. Джонсон, К. "Angular в дії." - Manning Publications, 2017. – 180 с.
9. Семенов О. "Використання REST API у розробці Angular додатків." - Дніпро: ДНУ, 2019. – 189 с.
10. Соловійов В. "Реалізація CRUD операцій у вебзастосунках на Angular." - Харків: ХНУ, 2021. – 100 с.
11. Сандерс, К. "Angular: Розробка великих застосунків." - O'Reilly Media, 2018. – 50 с.
12. Марченко І. О. "Створення вебзастосунків з використанням Angular: навчальний посібник." - Львів: НУЛП, 2018. – 102 с.

13. Докладчик А. І. "Розробка клієнтського застосунку на основі фреймворку Angular." - Донецьк: ДонНУ, 2017. – 159 с.
14. Бабенко О. В. "Використання Angular для розробки вебзастосунків: технології та прийоми." - Харків: ХНУ, 2016. – 107 с.
15. Лебідь А. С. "Основи веброзробки з використанням Angular: навчальний посібник." - Одеса: ОНПУ, 2019. – 52 с.
16. Федоров І. П. "Аспекти використання Angular у розробці вебдодатків." - Київ: КПІ, 2018. – 69 с.
17. Гребенік Д. В. "Розробка користувацького інтерфейсу з використанням Angular Material." - Дніпро: ДНУ, 2020. – 252 с.
18. Кузнецов П. І. "Особливості створення адаптивних вебзастосунків на Angular." - Луганськ: ЛНУ, 2017. – 72 с.
19. Руденко І. В. "Використання Angular у створенні високоефективних вебдодатків." - Запоріжжя: ЗНТУ, 2019. – 89 с.
20. Smith, J. "Mastering Angular: A Hands-On Guide to Angular Framework." - O'Reilly Media, 2017. – 142 с.
21. Freeman, A. "Pro Angular: Develop modern web applications." - Apress, 2017. – 92 с.
22. Green, J. "Angular Development with TypeScript." - Manning Publications, 2018. – 148 с.
23. Kurz, T. "Angular 5: From Theory to Practice." - Leanpub, 2017. – 122 с.
24. Schwartz, O. "Testing Angular Applications." - O'Reilly Media, 2018. – 12 с.
25. Інтернет-ресурси Angular [Electronic resource]. – Access mode: <https://angular.io/>.
26. Інтернет-ресурси TypeScript [Electronic resource]. – Access mode: <https://www.typescriptlang.org/>.
27. Інтернет-ресурси RxJS [Electronic resource]. – Access mode: <https://rxjs.dev/>.

28. Інтернет-ресурси Web-розробка [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Web>.
29. Самсонов А. "Реалізація аутентифікації та авторизації в Angular." - Харків: ХНУРЕ, 2018. – 120 с.
30. Максимов Д. "Розробка масштабованих Angular додатків." - Київ: КПІ, 2019. – 127 с.
31. Джонсон А. "Розробка корпоративних застосунків на Angular." - Одеса: ОНПУ, 2021. – 182 с.
32. Нігро А. "Angular: практика створення вебзастосунків." - Київ: КВІЦ, 2018. - – 107 с.
33. Фрімен, Адам. "Angular 8: Повний практичний посібник." - Київ: БІНОМ. Лабораторія знань, 2019. – 152 с.
34. Грінберг, Й. "Angular: великий практичний посібник." - ДМК Пресс, 2018. – 142 с.
35. Хоскінс, А. "Angular: від новачка до професіонала." - Київ: КВІЦ, 2019. – 178 с.
36. Лебідь А. "Впровадження Angular у розробці ентєрпрайз-застосунків." - Одеса: ОНПУ, 2017. – 278 с.
37. Кузнецов П. "Створення SPA застосунків з використанням Angular." - Луганськ: ЛНУ, 2018. – 171 с.
38. Лазарєв А. "Архітектура вебзастосунків на Angular." - Київ: КПІ, 2020. – 179 с.
39. Макфарленд А. "Angular: від новачка до експерта." - Київ: КВІЦ, 2019. – 149 с.
40. Берман К. "Підготовка великих даних для відображення в Angular." - Дніпро: ДНУ, 2018. – 111 с.
41. Хоровиц, А. "Angular: від теорії до практики." - Київ: БІНОМ. Лабораторія знань, 2019. – 171 с.
42. Фернандес, Ф. "Angular: Розробка ефективних вебзастосунків." - Packt Publishing, 2019. – 144 с.

43. Сміт, М. "Angular і TypeScript для професіоналів." - Apress, 2016. – 164 с.
44. Бреттел, М. "AngularJS: За визначенням і на практиці." - O'Reilly Media, 2014. – 2 с.
45. Фланаган, Д. "JavaScript: Підробне розгляд." - O'Reilly Media, 2011. – 44 с.
46. Шапіров, Й. "Програмування на TypeScript: підробний огляд." - O'Reilly Media, 2016. – 82 с.

ДОДАТОК А
Текст програми

A.1 Текст файла salary-page.component.html

```

<div class="flex-col rounded-lg bg-white px-8 py-8 dark:bg-night-700 sm:flex-row">
  <div class="mb-4 flex items-center justify-between" *ngIf="users?.length">
    <div class="flex-col items-center">
      <h3 class="text-md font-bold text-gray-900 dark:text-night-50">Salary page</h3>
      <span class="text-sm text-gray-400 dark:text-night-300">Count employees salary</span>
    </div>
    <div class="items-center flex">
      <button
        class="bg-white hover:bg-gray-300 text-white font-bold p-2 rounded mx-2 flex items-center
        justify-center"
        (click)="switchMonth(true)">
        <svg-icon src="assets/icons/heroicons/solid/chevron-right.svg"
          [svgClass]="h-5 w-5 text-gray-400 dark:text-night-400 rotate-180">
        </svg-icon>
      </button>
      <h2 *ngIf="routeDate">{{ users[0]?.days[0]?.date | date: 'MMMM, YYYY' }}</h2>
      <button
        class="bg-white hover:bg-gray-300 text-white font-bold p-2 rounded mx-2 flex items-center
        justify-center"
        (click)="switchMonth(false)">
        <svg-icon src="assets/icons/heroicons/solid/chevron-right.svg"
          [svgClass]="h-5 w-5 text-gray-400 dark:text-night-400">
        </svg-icon>
      </button>
    </div>
  </div>
  <button
    class="flex-none rounded-md bg-gray-100 px-4 py-2.5 text-xs font-semibold text-gray-500
    hover:bg-gray-200 dark:bg-night-600 dark:text-night-200 dark:hover:bg-night-500"
    (click)="goCurrentMonth()">Today

```

```

</button>

</div>

<div class="relative overflow-x-auto">

  <table class="w-full table-auto">

    <thead class="text-xs uppercase text-gray-400 dark:text-night-300">

      <tr>

        <th class="p-3 text-center"></th>

        <th class="p-3 text-center"

          *ngFor="let label of salaryInfo[0] | keyvalue: originalOrder; let index = index">

            <span>{{label.key}}</span>

          </th>

        <th class="p-3 text-center"></th>

      </tr>

    </thead>

    <tbody>

      <tr *ngFor="let user of users; let userIndex = index">

        <td class="p-4" (click)="openUserProfile(user.id)">

          <div class="flex items-center">

            <div class="flex bg-primary-500 relative w-8 h-8 justify-center items-center m-1 mr-2
text-xl rounded-full text-white">

              <img *ngIf="user?.avatar" class="rounded-full" [src]="user.avatar">

              <span *ngIf="!user?.avatar">{{ user?.fullName?.[0] }}</span>

            </div>

            <div class="flex flex-col justify-items-start">

              <a

                class="mb-1 text-sm font-semibold text-gray-900 dark:text-night-100
dark: hover:text-primary-500">{{

                user?.fullName

              }}</a>

              <span class="text-xs font-semibold text-gray-400 dark:text-night-300">{{ user?.level

```

```

    }}</span>

</div>

</div>

</td>

<ng-container *ngFor="let label of salaryInfo[userIndex] | keyvalue: originalOrder">

  <td *ngIf="label.key !== 'Planned salary' && label.key !== 'Bonus' &&
  label.key !== 'Comment' && label.key !== 'Calculated salary'" class="py-2 pl-2 text-
center">

    <span class="text-sm font-semibold text-gray-400 dark:text-night-200">

      {{label.value || '-'}}

    </span>

  </td>

  <td *ngIf="label.key === 'Planned salary' || label.key === 'Bonus'"
  class="py-2 pl-2 text-center">

    <span class="text-sm font-semibold text-gray-400 dark:text-night-200">

      <app-currency-input [value]="label.value"

        (valueChanged)="changeSalaryInfo($event, 'number', label, userIndex)">

      </app-currency-input>

    </span>

  </td>

  <td *ngIf="label.key === 'Calculated salary'" class="py-2 pl-2 text-center">

    <span class="text-sm font-semibold text-gray-400 dark:text-night-200">

      {{label.value | currency}}

    </span>

  </td>

  <td *ngIf="label.key === 'Comment'" class="py-2 pl-2 text-center">

    <span class="text-sm font-semibold text-gray-400 dark:text-night-200">

      <textarea type="text" [value]="label.value"

        (keyup.enter)="changeSalaryInfo($event, 'text', label, userIndex)"></textarea>

    </span>

```

```

        </td>

    </ng-container>

    <td class="py-2 pl-2 text-center cursor-pointer" (click)="copySalaryReport(userIndex)">

        <span

            class="text-sm font-semibold text-gray-400 dark:text-night-200 flex items-center
justify-center">

                <svg-icon src="assets/icons/heroicons/outline/cursor-click.svg"

                    [svgClass]="h-5 w-5 text-gray-400 dark:text-night-400 mr-2">

                </svg-icon>

                Copy

            </span>

        </td>

    </tbody>

</table>

<div class="flex items-center divide-x mt-8" *ngIf="totalSalary">

    <div class="flex-col items-center mr-4">

        <h3 class="text-md font-bold text-gray-900 dark:text-night-50">{{totalSalary[1] |
currency}}</h3>

        <span class="text-sm text-gray-400 dark:text-night-300">Total salary</span>

    </div>

    <div class="flex-col items-center pl-4 mr-4">

        <h3 class="text-md font-bold text-gray-900 dark:text-night-50">{{totalSalary[0] |
currency}}</h3>

        <span class="text-sm text-gray-400 dark:text-night-300">Planned salary</span>

    </div>

    <div class="flex-col flex-end items-center pl-4">

        <h3 class="text-md font-bold text-gray-900 dark:text-night-50">USD to UAH</h3>

        <span class="text-sm text-gray-400 dark:text-night-300">

            <div class="relative text-gray-600 focus-within:text-gray-400">

                <input

```

```
class="bg-white appearance-none border-1 border-gray-200 rounded py-2 px-4 text-
gray-700 leading-tight focus:outline-none focus:bg-white focus:border-primary-500"
```

```
type="number" [(ngModel)]="currency">
```

```
</div>
```

```
</span>
```

```
</div>
```

```
</div>
```

```
</div>
```

```
</div>
```

A.2 Текст файла salary-page.component.ts

```
import { Component, OnInit, ChangeDetectionStrategy, ChangeDetectorRef, Inject } from
'@angular/core';

import { User } from '../db/users/user.model';

import { UsersService } from '../db/users/users.service';

import { filter, map, mergeMap, take, switchMap } from 'rxjs/operators';

import { HolidaysService } from '../db/holidays/holidays.service';

import { Holiday } from '../db/holidays/holiday.model';

import { ActivatedRoute, Router } from '@angular/router';

import { combineLatest, Observable } from 'rxjs';

import { NationalHolidays } from '../shared/data/national-holidays';

import { UserDay } from '../db/user-days/user-day.model';

import { generateUserDaysByMonth } from '../shared/utills/generate-user-days-by-month';

import * as cloneDeep from 'lodash/cloneDeep';

import { dateDay } from '../shared/utills/date-day';

import { CurrencyPipe, KeyValue } from '@angular/common';

import { ReportsService } from '../db/reports/report.service';

import { Report, ToReportType } from '../db/reports/report.model';

import { copyToClipboard } from '../shared/utills/copy-to-clipboard';

import { UaMonthsGenitivePipe } from '../shared/utills/ua-months-genitive';
```

```

import { isWeekday } from '../shared/utils/is-weekday';
import { isNationalHoliday } from '../shared/utils/is-national-holiday';
import { HotToastService } from '@ngneat/hot-toast';

@Component({
  selector: 'app-salary-page',
  templateUrl: './salary-page.component.html',
  styleUrls: ['./salary-page.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush
})

export class SalaryPageComponent implements OnInit {
  users: User[];
  salaryInfo: any[] = [];
  usersReports: any[] = [];
  totalSalary: number[];
  routeDate: Date;
  currency: number = 37.3;

  constructor(
    private router: Router,
    private activatedRoute: ActivatedRoute,
    private changeDetectorRef: ChangeDetectorRef,
    private usersService: UsersService,
    private holidaysService: HolidaysService,
    private reportsService: ReportsService,
    private currencyPipe: CurrencyPipe,
    private uaMonthsGenitivePipe: UaMonthsGenitivePipe,
    private toast: HotToastService
  ) {}

  ngOnInit(): void {

```

```

    this.getRouteParams();
}

getRouteParams() {
    const month$: Observable<Date> = this.activatedRoute.firstChild.params
        .pipe(filter(params => params.date), map(params => new Date(params.date)));
    combineLatest([month$.pipe(switchMap((date) => this.getHolidays(date))), this.getUsers(), month$])
        .pipe(take(1)).subscribe(([holidays, users, month]) => this.generateMonthTimeline(holidays, users,
month));
}

getUsers(): Observable<any> {
    return this.userService.getAll({ field: 'createdAt', value: 'desc' });
}

getHolidays(date: Date): Observable<any> {
    date.setHours(10);
    return this.holidaysService.getByArrayField('date', date.toISOString().slice(0, 7));
}

generateMonthTimeline(holidays: Holiday[], users: User[], month: Date) {
    this.getWorkingUsers(users);
    this.generateTimeline(users, month);
    this.fillUsersTimelineByHolidays(holidays, users);
    this.fillFieldsOfUserDays(users);
    this.fillUsersReports(users);
    this.getSalaryReports(holidays, users, month);
}

getWorkingUsers(users: User[]) {
    this.users = users.filter(user => !user.isStoppedWork);
}

generateTimeline(users: User[], month: Date) {
    this.routeDate = month;

```

```

const emptyUserDays: UserDay[] = generateUserDaysByMonth(month);

users.forEach(user => {
  user.days = cloneDeep(emptyUserDays);
})
}

fillUsersTimelineByHolidays(holidays: Holiday[], users: User[]) {
  holidays.forEach((holiday) => {
    const userIndex = users.findIndex(user => user.id === holiday.userId);

    let dayIndexes = users[userIndex]?.days.map((day, index) => {
      if (dateDay(day.date) >= holiday.dateFrom && dateDay(day.date) <= holiday.dateTo) {
        return index;
      }
    }).filter(index => typeof index !== 'undefined');

    dayIndexes.forEach(dayIndex => users[userIndex].days[dayIndex].holidays.push(holiday))
  })
}

fillFieldsOfUserDays(users: User[]) {
  users.forEach(user => {
    user.days.forEach(day => {
      day.isToday = this.isToday(day.date)
      day.isWeekend = !isWeekday(day.date);
      day.isNationalHoliday = isNationalHoliday(day.date);
      day.isFirstDay = day.holidays[0]?.dateFrom === dateDay(day.date);
      day.isLastDay = day.holidays[0]?.dateTo === dateDay(day.date);
    })
  })
}

fillUsersReports(users: User[]) {
  this.usersReports = [];

  users.forEach(user => {

```

```

        this.usersReports.push({ user, salaryReport: {} })
    })
}

getUsersLabels(holidays: Holiday[], users: User[], month: Date) {
    this.salaryInfo = [];
    users.forEach((user, userIndex) => {
        this.salaryInfo.push(this.getUserSalaryInfo(userIndex, holidays, users, month));
        this.calculateSalary(userIndex, false);
    })
    this.calculateTotalSalary();
    this.changeDetectorRef.detectChanges();
}

calculateTotalSalary() {
    let totalPlannedSalary = 0;
    let totalCalculatedSalary = 0;
    this.users.forEach((user, userIndex) => {
        if (this.salaryInfo[userIndex]) {
            totalPlannedSalary += this.salaryInfo[userIndex]['Planned salary'];
            totalCalculatedSalary += this.salaryInfo[userIndex]['Calculated salary'];
        }
    })
    this.totalSalary = [totalPlannedSalary, totalCalculatedSalary];
}

getSalaryReports(holidays: Holiday[], users: User[], month: Date) {
    month.setHours(10);
    const date = month.toISOString().slice(0, 7);

    const salaryReportsObservables = users.map(user => this.reportsService.getByTwoFields('userId',
user.id, 'date', date))

    combineLatest(salaryReportsObservables).pipe(take(1)).subscribe((reports: Report[]) => {
        reports.forEach((report) => {

```

```

    if (!report[0]) {
        return;
    }

    const userIndex = users.findIndex(user => user.id === report[0]?.userId);

    this.usersReports[userIndex].salaryReport = report[0];
})

let emptyUsersReports = this.usersReports.map((userReport) => {
    if (!userReport.salaryReport.userId) {
        return userReport;
    }
}).filter(report => !!report);

if (emptyUsersReports.length) {
    let createReportsPromises = emptyUsersReports.map(emptyUserReport =>
this.createSalaryReport(emptyUserReport.user, date));

    Promise.all(createReportsPromises).then((res) => this.getSalaryReports(holidays, users, month));
}

else {
    this.getUsersLabels(holidays, users, month);
}

})
}

createSalaryReport(user: User, date: string) {
    return this.reportsService.create(new Report({
        userId: user.id,
        date,
        plannedSalary: +user.salary,
    }));
}

getUserSalaryInfo(userIndex: number, holidays: Holiday[], users: User[], month: Date) {
    const workingDaysAndSalary = this.getWorkDaysLabel(userIndex, holidays, users, month);

```

```

const salaryReport = this.usersReports[userIndex].salaryReport;

const salaryInfo = {
  ...workingDaysAndSalary,
  'Planned salary': salaryReport.plannedSalary,
  'Bonus': salaryReport.bonus,
  'Calculated salary': 0,
  'Comment': salaryReport.comment,
  'Card': this.usersReports[userIndex]?.user?.card
}

return salaryInfo;
}

changeSalaryInfo(event, inputType: string, label, userIndex: number) {
  let newValue = event.target.value;

  if (inputType === 'number' && (newValue < 0 || newValue > 999999)) {
    newValue = label.value;
    return;
  }

  if (inputType === 'number') {
    this.usersReports[userIndex].salaryReport[ToReportType[label.key]] = +newValue;
    this.salaryInfo[userIndex][label.key] = +newValue;
    this.calculateSalary(userIndex, true);
  }

  else {
    this.usersReports[userIndex].salaryReport[ToReportType[label.key]] = newValue;
  }

  this.changeDetectorRef.detectChanges();

  this.reportsService.update(this.usersReports[userIndex].salaryReport.id,
this.usersReports[userIndex].salaryReport);
}

```

```

calculateSalary(userIndex: number, withTotal: boolean) {
  let salaryLabel = this.salaryInfo[userIndex];

  let calculatedSalary =
    ((+salaryLabel['Workdays'].slice(0, 2) + salaryLabel['Vacations'] +
      salaryLabel['Sickness'] + salaryLabel['National']) /
      +salaryLabel['Workdays'].slice(3, 5) * salaryLabel['Planned salary']) + +salaryLabel['Bonus'];

  this.salaryInfo[userIndex]['Calculated salary'] = calculatedSalary;

  if (withTotal) {
    this.calculateTotalSalary();
  }
}

getWorkDaysLabel(userIndex: number, holidays: Holiday[], users: User[], month: Date) { //: any[] {
  const vacations = this.getAbsenceDaysNumber(holidays, users, userIndex, 'vacation');
  const sickness = this.getAbsenceDaysNumber(holidays, users, userIndex, 'sickness');
  const daysOff = this.getAbsenceDaysNumber(holidays, users, userIndex, 'day_off');

  const nationalHolidays = users[userIndex].days.map((day) => day.isNationalHoliday).filter((day) =>
    !!day);

  const allWorkingDays = this.getWeekdaysInMonth(month);

  const workingDays = allWorkingDays - vacations - sickness - daysOff - nationalHolidays.length;

  const workingDaysLabel = {
    'Workdays': `${workingDays}/${allWorkingDays}`,
    'Sickness': sickness,
    'Vacations': vacations,
    'National': nationalHolidays.length,
    'DaysOff': daysOff,
  };

  users[userIndex].workingDaysLabel = workingDaysLabel;

  return workingDaysLabel;
}

getAbsenceDaysNumber(holidays: Holiday[], users: User[], userIndex: number, holidayType: string) {

```

```

let daysNumber = 0;

if (holidayType !== 'day_off') {

    holidays = holidays.filter((holiday) => holiday.type === holidayType && holiday.approved &&
holiday.userId === users[userIndex].id);

}

else {

    holidays = holidays.filter((holiday) => (holiday.type === holidayType ||

        (!holiday.approved && (holiday.type === 'vacation' || holiday.type === 'sickness')))) &&
holiday.userId === users[userIndex].id);

}

holidays.forEach((holiday) => {

    const firstDay = users[userIndex].days[0].date;

    const lastDay = users[userIndex].days[users[userIndex].days.length - 1].date;

    const lessFirst = holiday.dateFrom < dateDay(firstDay);

    const moreLast = holiday.dateTo > dateDay(lastDay);

    if (lessFirst && moreLast) {

        daysNumber = users[userIndex].days.length;

    } else if (lessFirst && !moreLast) {

        daysNumber += this.getBusinessDaysNumber(firstDay, new Date(holiday.dateTo), userIndex);

    } else if (!lessFirst && moreLast) {

        daysNumber += this.getBusinessDaysNumber(new Date(holiday.dateFrom), lastDay, userIndex) - 1;

    }

    else {

        daysNumber += this.getBusinessDaysNumber(new Date(holiday.dateFrom), new
Date(holiday.dateTo), userIndex);

    }

})

return daysNumber;

}

getBusinessDaysNumber(startDate, endDate, userIndex: number) {

    const nationalHolidaysDays = this.getNationalHolidaysDaysNumber(startDate, endDate, userIndex);

```

```

let elapsed, daysBeforeFirstSunday, daysAfterLastSunday;

const ifThen = (a, b, c) => {
  return a === b ? c : a;
};

elapsed = endDate - startDate;

elapsed /= 86400000;

daysBeforeFirstSunday = (7 - startDate.getDay()) % 7;

daysAfterLastSunday = endDate.getDay();

elapsed -= (daysBeforeFirstSunday + daysAfterLastSunday);

elapsed = (elapsed / 7) * 5;

elapsed += ifThen(daysBeforeFirstSunday - 1, -1, 0) + ifThen(daysAfterLastSunday, 6, 5);

elapsed -= nationalHolidaysDays;

return Math.round(elapsed);
}

getNationalHolidaysDaysNumber(startDate, endDate, userIndex: number) {
  let nationalHolidaysDays = [];

  for (let i = startDate.getDate(); i <= endDate.getDate(); i++) {
    nationalHolidaysDays.push(this.users[userIndex].days[i - 1].isNationalHoliday &&
!this.users[userIndex].days[i - 1].isWeekend);
  }

  nationalHolidaysDays = nationalHolidaysDays.filter(day => !!day);

  return nationalHolidaysDays.length;
}

getWeekdaysInMonth(month: Date) {
  const daysInMonth = new Date(month.getFullYear(), month.getMonth() + 1, 0).getDate();

  let weekdays = 0;

  for (let i = 0; i < daysInMonth; i++) {
    let day = new Date(month.getFullYear(), month.getMonth(), i);

    if (isWeekday(day)) {
      weekdays++;
    }
  }
}

```

```

    }
  }
  return weekdays;
}

isWeekday(day: Date | string) {
  const dayNumber = new Date(day).getDay();
  return dayNumber !== 0 && dayNumber !== 6;
}

isToday(day: Date) {
  return new Date(day).toString() === new Date().toString();
}

goCurrentMonth() {
  this.routeDate = new Date();
  const year = this.routeDate.getFullYear();
  const month = this.routeDate.getMonth() + 1;
  this.router.navigate(['./', `${year}-${month}`], { relativeTo: this.activatedRoute });
  this.getRouteParams();
}

switchMonth(prevMonth: boolean) {
  if (prevMonth) {
    this.routeDate.setMonth(this.routeDate.getMonth() - 1);
  }
  else {
    this.routeDate.setMonth(this.routeDate.getMonth() + 1);
  }
  const year = this.routeDate.getFullYear();
  const month = this.routeDate.getMonth() + 1;
  this.router.navigate(['./', `${year}-${month}`], { relativeTo: this.activatedRoute });
  this.getRouteParams();
}

```

```

copySalaryReport(userIndex: number) {

  const month = this.uaMonthsGenitivePipe.transform(this.routeDate);

  const salaryLabel = this.salaryInfo[userIndex];

  let report =

    `Відпрацьовано ${salaryLabel['Workdays'].slice(0, 2)} із ${salaryLabel['Workdays'].slice(3, 5)}
    робочих днів у ${month}\n`

    + `Використано:\n- ${salaryLabel['Sickness']} sickness days\n- ${salaryLabel['Vacations']}
    vacations\n- ${salaryLabel['DaysOff']} days off\n`;

  if (salaryLabel['National'] > 0) {

    report += `У ${month} - ${salaryLabel['National']} national ${salaryLabel['National']} == 1 ?
    'holiday': 'holidays'\n`

  }

  if (salaryLabel['Bonus'] > 0) {

    report += `Нараховано бонус: ${this.currencyPipe.transform(salaryLabel['Bonus'])}\n`

  }

  report += `USD to UAH: ${this.currency}\n`

  report += `До виплати: ${this.currencyPipe.transform(salaryLabel['Calculated
  salary'])} (${this.currencyPipe.transform(salaryLabel['Calculated salary'] * this.currency, 'UAH '))\n`

  report += `Реквізити: ${salaryLabel['Card']}\n`

  report += `Цей лист був згенерований системою, яка ще перебуває у тестуванні, тому будьте
  уважні, перевірте всі дані, щоб вони були правильними.\n`

  copyToClipboard(report);

  this.toast.success('Copied to clipboard!')

}

openUserProfile(userId: string) {

  this.router.navigate(['users/${userId}/profile']);

}

originalOrder = (a: KeyValue<number, string>, b: KeyValue<number, string>): number => {

  return 0;

}

}

```

A.3 Текст файла check.model.ts

```
import { dateDay } from 'src/app/shared/utls/date-day';
import { DbEntity } from '../shared/db-entity.model';
export class Check extends DbEntity {
  // -----
  title: string;
  description: string;
  day: string;
  type: TaskType;
  projectId: string;
  status: TaskStatus;
  proove: string;
  timestampsPauseRun?: number[];
  // -----
  assigneeId?: string;
  // order?: number;
  focused?: boolean;
  // -----
  constructor({ id, status, title, description, slug, createdAt, assigneeId, timestampsPauseRun, day, proove,
    type, focused, projectId }: Partial<Check> = {}) {
    super({id, slug, createdAt});
    this.status = status || TaskStatus.ToDo;
    this.title = title || 'No title';
    this.description = description || '';
    this.assigneeId = assigneeId || '';
    this.day = day || dateDay();
    this.timestampsPauseRun = timestampsPauseRun || [];
    this.proove = proove || '';
    // this.prooveType = prooveType || ProoveType.CommitLink;
```

```

this.type = type || TaskType.Task_or_Fix;

this.focused = focused || false;

this.projectId = projectId || 'DevHive';

// -----
}

get done() {

    return this.status === TaskStatus.Completed

}

get isPaused() {

    return !(this.timestampsPauseRun.length % 2);

}

public get durationMs () {

    const history = [ ... this.timestampsPauseRun || [] ];

    if (history.length % 2) {

        history.push(new Date().getTime());

    }

    const ms = history.reduce(

        (sum, value, index) => {

            const isOdd = !(index % 2);

            if (isOdd) {

                return sum;

            }

            const previousValue = history[index - 1];

            const delta = Math.abs(previousValue - value);

            return sum + delta;

        }, 0);

    return ms;

}

logActivityChange() {

    const now = new Date().getTime();

```

```

        this.timestampsPauseRun.push(now);
    }

    pause() {
        if (this.isPaused){
            return;
        }

        const now = new Date().getTime();

        this.timestampsPauseRun.push(now);
    }

    runAgain() {
        if (!this.isPaused){
            return;
        }

        const now = new Date().getTime();

        this.timestampsPauseRun.push(now);
    }

    complete() {
        const now = new Date().getTime();

        if (!this.isPaused){
            this.timestampsPauseRun.push(now);
        }

        this.status = TaskStatus.Completed;
    }
}

export enum TaskStatus {
    ToDo = 'to_do',
    InProgress = 'in_progress',
    Paused = 'paused',
    Completed = 'completed'
}

```

```

export enum TaskType {
  Task_or_Fix = 'Commit',
  Feature_or_Refactoring = 'Pull-request',
  Learning_or_Investigation = 'Learning_or_Investigation',
  Meeting = 'Meeting',
}

export enum ProjectType {
  faciliatease = 'Commit',
  Feature_or_Refactoring = 'Pull-request',
  Learning_or_Investigation = 'Learning_or_Investigation',
  Meeting = 'Meeting',
}

export enum ProoveTypeString {
  commit_link = 'commit',
  pull_request_link = 'pull request',
}

export const status2emojii = {
  [TaskStatus.ToDo]: '☐',
  [TaskStatus.InProgress]: '⏸',
  [TaskStatus.Paused]: '⏸',
  [TaskStatus.Completed]: '☑',
}

```

A.4 Текст файла checks.service.ts

```

import { dateDay } from '../shared/utils/date-day';
import { map } from 'rxjs/operators';
import { Check } from '../db/checks/check.model';
import { Injectable } from '@angular/core';

```

```

import { Firestore } from '@angular/fire/firestore';
import { CrudService } from '../shared/crud.service';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ChecksService extends CrudService<Check> {
  collectionName: string = 'checks';

  constructor(public firestore: Firestore) {
    super(firestore)
  }

  getByDay(day: Date, userId: string): Observable<Check[]> {
    const dateString = dateDay(day);
    return this.getByTwoFields('assigneeId', userId, 'day', dateString)
      .pipe(
        map((tasks: Check[]) => (tasks || []).map(t => {
          return new Check({
            status: t.status,
            title: t.title,
            description: t.description,
            createdAt: t.createdAt,
            assigneeId: t.assigneeId,
            projectId: t.projectId,
            id: t.id,
            day: t.day,
            timestampsPauseRun: t.timestampsPauseRun,
            proove: t.proove,
            type: t.type,
            focused: t.focused,
          });
        }));
  }

```

```

    )))
  );
}
}

```

A.5 Текст файла `holiday.model.ts`

```

import { DbEntity } from '../shared/db-entity.model';

export class Holiday extends DbEntity {

  type: HolidayType;

  userId: string;

  dateFrom: string | Date;

  dateTo: string | Date;

  date: string[];

  year: string[];

  approved?: boolean;

  comment?: string;

  constructor({ userId, type, dateFrom, dateTo, date, year, approved, comment }: Partial<Holiday> = {})
  {
    super({});

    this.type = type || HolidayType.DayOff;

    this.userId = userId || null;

    this.dateFrom = dateFrom || null;

    this.dateTo = dateTo || null;

    this.approved = approved || false;

    this.comment = comment || "";

    this.date = date || [];

    this.year = year || [];

  }
}

```

```
export enum HolidayType {
    Vacation = 'vacation',
    Sickness = 'sickness',
    FromHome = 'from_home',
    DayOff = 'day_off',
    Overtime = 'overtime',
    WeekendWork = 'weekend_work',
    Transfer = 'transfer',
}
```

A.6 Текст файла holidays.service.ts

```
import { Injectable } from '@angular/core';
import { collectionData, Firestore } from '@angular/fire/firestore';
import { query, where } from 'firebase/firestore';
import { map, Observable } from 'rxjs';
import { CrudService } from '../shared/crud.service';
import { Holiday } from '../holiday.model';

@Injectable({
    providedIn: 'root'
})
export class HolidaysService extends CrudService<Holiday> {
    collectionName: string = 'holidays';

    constructor(public firestore: Firestore) {
        super(firestore)
    }

    getUserHolidaysByYear(userId: any, year: any, unique: boolean = false): Observable<Holiday | Holiday[]> {
        let docsRef = query(this.collectionRef, where('userId', "==", userId), where('year', 'array-contains', year));

        return collectionData(docsRef, { idField: 'id' }).pipe(
```

```

map((lists: Holiday[]) => {
  if (unique) {
    return lists.length ? lists[0] : null;
  }
  return lists || undefined;
})
);
}
}

```

A.7 Текст файла **national-holiday.model.ts**

```

export class NationalHoliday {
  title: string;
  date: string;
  constructor({title, date}: Partial<NationalHoliday> = {}) {
    this.title = title || "";
    this.date = date || null;
  }
}

```

A.8 Текст файла **list.model.ts**

```

import { dateDay } from '../shared/utils/date-day';
import { DbEntity } from '../shared/db-entity.model';
export class List extends DbEntity {
  // -----
  title: string;
  day: string;
  // -----

```

```

assigneeId?: string;

// -----

constructor({id, title, day, assigneeId}: Partial<List> = {}) {
  super({});

  this.id = id || null;

  this.day = day || dateDay();

  this.title = title || new Date().toDateString().slice(0,10);

  this.assigneeId = assigneeId || null;
}

```

A.9 Текст файла log.model.ts

```

import { DbEntity } from '../shared/db-entity.model';

export class Log extends DbEntity {
  // -----

  entity: LogEntity;

  entityId: string;

  parentId: string;

  type: LogType;

  userName: string;

  value: {
    field?: string,

    old?: string;

    new?: string;
  };

  constructor({ id, type, entity, value, userName, parentId, entityId }: Partial<Log> = {}) {
    super({});

    this.id = id || null;

    this.type = type || null;

    this.entity = entity || null;

```

```

    this.parentId = parentId || null;

    this.entityId = entityId || null;

    this.value = value || null;

    this.userName = userName || null;

  }
}

export enum LogType {

  Edit = 'edit',

  Add = 'add',

  Delete = 'delete',

  Get = 'get',

}

export enum LogEntity {

  Users = 'users',

  Lists = 'lists',

  Tasks = 'tasks',

}

```

A.10 Текст файла logs.service.ts

```

import { Injectable } from '@angular/core';
import { Firestore } from '@angular/fire/firestore';
import { CrudService } from '../shared/crud.service';
import { Log } from '../log.model';

@Injectable({
  providedIn: 'root'
})

export class LogsService extends CrudService<Log> {

  collectionName: string = 'logs';

  constructor(public firestore: Firestore) {

```

```

    super(firestore)
  }
}

```

A.11 Текст файла report.model.ts

```

import { DbEntity } from '../shared/db-entity.model';

export class Report extends DbEntity {

  userId: string;

  date: string;

  plannedSalary: number;

  bonus?: number;

  comment?: string;

  constructor({ userId, date, plannedSalary, bonus, comment }: Partial<Report> = {}) {
    super({});

    this.userId = userId || null;

    this.date = date || "";

    this.plannedSalary = plannedSalary || 0;

    this.bonus = bonus || 0;

    this.comment = comment || "";
  }
}

export enum ToReportType {

  'Planned salary' = 'plannedSalary',

  'Bonus' = 'bonus',

  'Calculated salary' = 'calculatedSalary',

  'Comment' = 'comment',

}

```

A.12 Текст файла report.service.ts

```
import { Injectable } from '@angular/core';
import { Firestore } from '@angular/fire/firestore';
import { CrudService } from '../shared/crud.service';
import { Report } from './report.model';

@Injectable({
  providedIn: 'root'
})
export class ReportsService extends CrudService<Report> {
  collectionName: string = 'reports';
  constructor(public firestore: Firestore) {
    super(firestore)
  }
}
```

A.13 Текст файла crud.service.ts

```
import {
  CollectionReference,
  Firestore,
  Query,
  addDoc,
  collection,
  deleteDoc,
  doc,
  query,
  updateDoc,
  where,
  docData,
```

```

} from '@angular/fire/firestore';

import { DocumentData } from 'rxfire/firestore/interfaces';

import { DocumentReference } from 'firebase/firestore';

import { Injectable } from '@angular/core';

import { Observable, of } from 'rxjs';

import { map, mergeMap } from 'rxjs/operators';

import { collectionData } from 'rxfire/firestore';

import { DbEntity } from './../../db/shared/db-entity.model';

@Injectable({
  providedIn: 'root'
})

export abstract class CrudService<T extends DbEntity> {
  abstract collectionName: string;

  constructor(public firestore: Firestore) { }

  protected get collectionRef(): CollectionReference {
    return collection(this.firestore, this.collectionName);
  }

  create(doc: T): Promise<DocumentReference<DocumentData>> {
    return addDoc(this.collectionRef, { ...doc });
  }

  update(id: string, entity: T): Promise<void> {
    const docRef = doc(this.firestore, this.collectionName, id);

    const newObject: any = { ...entity };

    return updateDoc(docRef, newObject);
  }

  delete(id: string): Promise<void> {
    const docRef = doc(this.firestore, this.collectionName, id);

    return deleteDoc(docRef);
  }

  getAll(

```

```

order = { field: 'createdAt', value: 'asc' as any },
search = null,
): Observable<DocumentData[]> {
  const allQuery: Query<DocumentData> = !search ?
    query(this.collectionRef) :
    query(this.collectionRef, where(search.field, '==', search.value));
  return collectionData(allQuery, { idField: 'id' }).pipe(
    map((data: T[]) => {
      const newData = (data || []).map((item: T) => {
        return {
          ...item,
          createdAt: (item.createdAt as any)?.toDate() || new Date()
        }
      });
      const koef = order.value === 'desc' ? 1 : -1;
      newData.sort((a, b) => {
        return a.createdAt.getTime() > b.createdAt.getTime() ? koef * 1 : koef * -1;
      })
      return newData;
    })
  );
}

getId(id: string): Observable<T> {
  const docRef = doc(this.firestore, `${this.collectionName}/${id}`);
  return docData(docRef, { idField: 'id' }).pipe(
    mergeMap((list: T) => {
      if (list) {
        return of(list);
      }
    })
  );
  return this.getBySlug(id);
}

```

```

    })
  );
}

getBySlug(slug: string): Observable<T> {
  if (!slug) {
    return of(undefined);
  }
  return this.getByField('slug', slug, true) as Observable<T>;
}

getByField(field: string, value: any, first: boolean = false): Observable<T | T[]> {
  if (!field) {
    return of(undefined);
  }
  const docsRef = query(this.collectionRef, where(field, '==', value));
  return collectionData(docsRef, { idField: 'id' }).pipe(
    map((lists: T[]) => {
      if (first) {
        return lists.length ? lists[0] : null;
      }
      return lists || undefined;
    })
  );
}

getByTwoFields(firstField: any, firstValue: any, secondField: any, secondValue: any,
  order = { field: 'createdAt', value: 'desc' as any },
): Observable<T | T[]> {
  let docsRef = query(this.collectionRef, where(firstField, "==", firstValue), where(secondField, "==",
secondValue));

  return collectionData(docsRef, { idField: 'id' }).pipe(
    map((data: T[]) => {

```

```

const newData = (data || []).map((item: T) => {
  return {
    ...item,
    createdAt: (item.createdAt as any)?.toDate() || new Date()
  }
});

const koef = order.value === 'desc' ? 1 : -1;

newData.sort((a, b) => {
  return a.createdAt.getTime() > b.createdAt.getTime() ? koef * 1 : koef * -1;
})

return newData;
})

);
}

getByArrayField(field: string, value: any, first: boolean = false): Observable<T | T[]> {
  if (!field) {
    return of(undefined);
  }

  const docsRef = query(this.collectionRef, where(field, 'array-contains', value));

  return collectionData(docsRef, { idField: 'id' }).pipe(
    map((lists: T[]) => {
      if (first) {
        return lists.length ? lists[0] : null;
      }

      return lists || undefined;
    })
  );
}
}

```

A.14 Текст файла db-entity.model.ts

```
export class DbEntity{
  id: string;
  slug: string;
  createdAt: Date;
  constructor(entity: Partial<DbEntity>) {
    this.createdAt = entity.createdAt || new Date();
    this.id = entity.id || null;
    this.slug = entity.slug || null;
  }
}
```

A.15 Текст файла user-day.model.ts

```
export class UserDay {
  date: Date;
  holidays: any[]
  notWorkingDay: boolean;
  isToday: boolean;
  isWeekend: boolean;
  isNationalHoliday: boolean;
  nationalHolidayName?: string;
  isBirthday: boolean;
  isFirstDay: boolean;
  isLastDay: boolean;
  constructor({ date, holidays, notWorkingDay, isToday, isWeekend, isNationalHoliday, isFirstDay, isLastDay }: Partial<UserDay> = {}) {
    this.date = date || null;
    this.holidays = holidays || [];
```

```

    this.notWorkingDay = notWorkingDay || false;

    this.isToday = isToday || false;

    this.isWeekend = isWeekend || false;

    this.isNationalHoliday = isNationalHoliday || false;

    this.isFirstDay = isFirstDay || false;

    this.isLastDay = isLastDay || false;

  }
}

```

A.16 Текст файла `user.model.ts`

```

import { DbEntity } from '../shared/db-entity.model';
import { UserDay } from '../user-days/user-day.model';

export class User extends DbEntity {
  // -----

  fullName: string;

  email: string;

  level: UserLevel;

  salary: number;

  avatar: string;

  phoneNumber: string;

  birthday: any;

  hasManualLoggingEnable: boolean;

  days?: UserDay[];

  workingDaysLabel?: string | Object;

  salaryReport?: any;

  projects?: string[];

  availableSick: number;

```

```

availableVacation: number;

constructor({ id, createdAt, avatar, fullName, email, phoneNumber, birthday, level,
  salary, hasManualLoggingEnable, isStoppedWork, startWorkingDay, endWorkingDay, projects,
  availableSick, availableVacation }: Partial<User> = {}) {

  super({id, createdAt});

  this.fullName = fullName || null;

  this.email = email || null;

  this.level = level || UserLevel.Junior;

  this.salary = salary || 9999;

  this.avatar = avatar || null;

  this.phoneNumber = phoneNumber || null;

  this.birthday = birthday || null;

  this.hasManualLoggingEnable = hasManualLoggingEnable || false;

  this.isStoppedWork = isStoppedWork || false;

  this.startWorkingDay = startWorkingDay || null;

  this.endWorkingDay = endWorkingDay || null;

  this.projects = projects || [];

  this.availableSick = availableSick;

  this.availableVacation = availableVacation;
}
}

export enum UserLevel {

  Trainee = 'Trainee',

  JuniorLow = 'Junior Low',

  Junior = 'Junior',

  JuniorStrong = 'Junior Strong',

  MiddleLow = 'Middle Low',

  Middle = 'Middle',

  MiddleStrong = 'Middle Strong',

  SeniorLow = 'Senior Low',

```

```
Senior = 'Senior',  
SeniorStrong = 'Senior Strong'  
}
```

A.17 Текст файла users.service.ts

```
import { Firestore } from '@angular/fire/firestore';  
import { CrudService } from '../shared/crud.service';  
import { User } from './user.model';  
  
@Injectable({  
  providedIn: 'root'  
})  
  
export class UsersService extends CrudService<User> {  
  collectionName: string = 'users';  
  constructor(public firestore: Firestore) {  
    super(firestore)  
  }  
}
```

ДОДАТОК Б
Слайди презентації



Рисунок Б.1 – Слайд № 1



Рисунок Б.2 – Слайд № 2

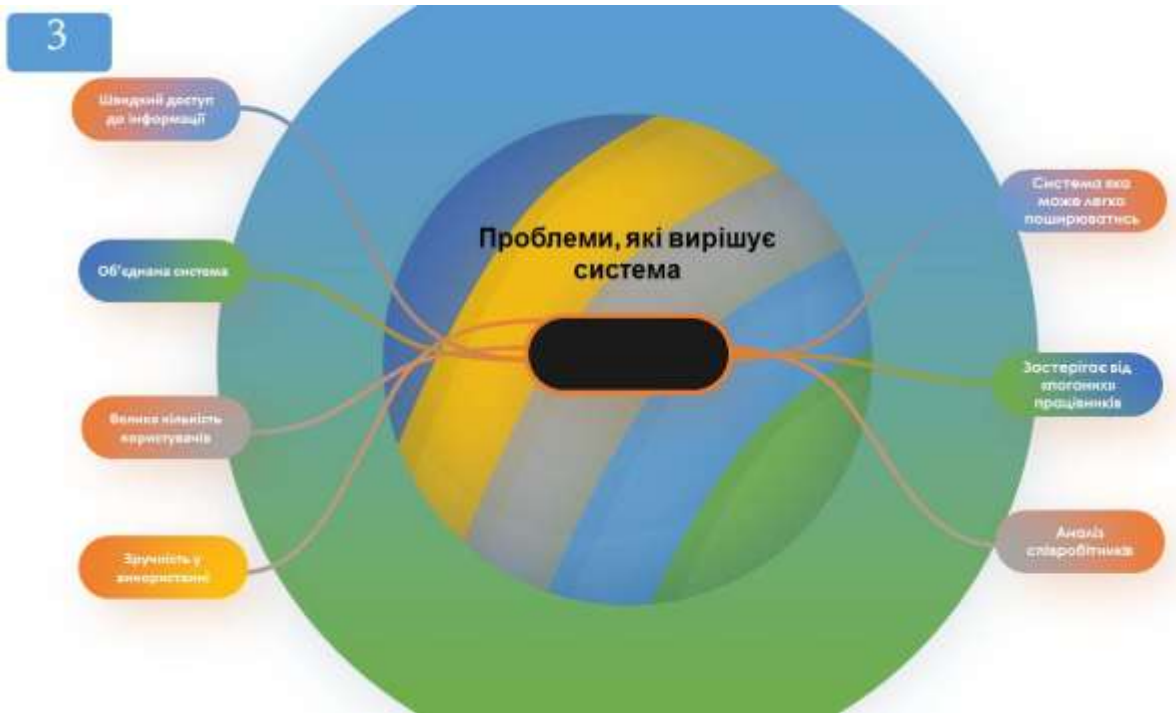


Рисунок Б.3 – Слайд № 3



Рисунок Б.4 – Слайд № 4

5 Бібліотеки машинного навчання

Під час проєктування системи був вибір між двома СУБД



Рисунок Б.5 – Слайд № 5

6 Архітектура проєкту

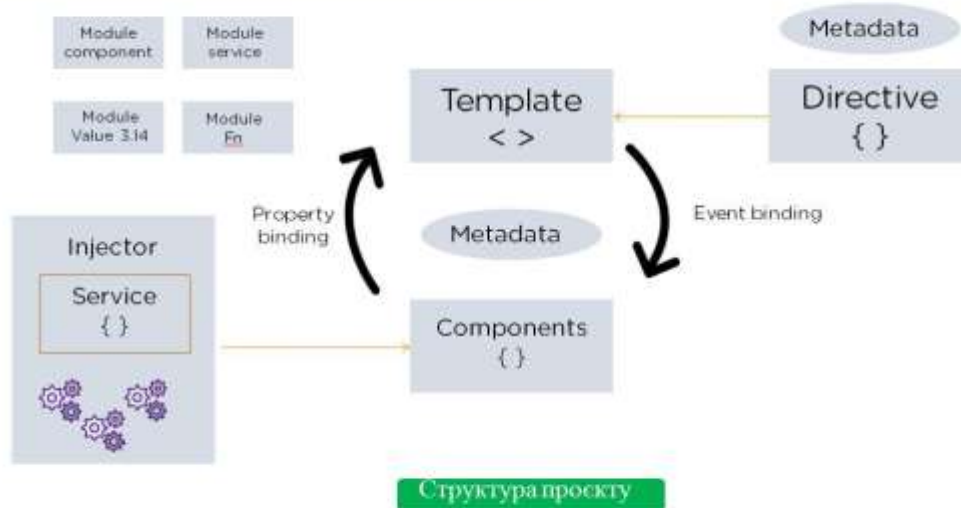
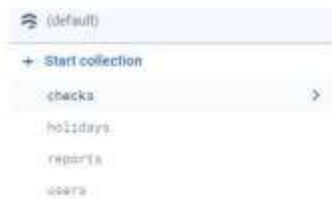


Рисунок Б.6 – Слайд № 6

7 База даних



Сутність DrugEntity



Структурна схема роботи БД

Рисунок Б.7 – Слайд № 7

8 MVVM

Angular використовує концепцію MVVM (Model-View-ViewModel) для покращення структури та управління користувацьким інтерфейсом додатків. MVVM спрощує роботу з даними, розділяючи їх на модель (Model), представлення (View) та ViewModel.

У цій архітектурі, Model відповідає за управління даними, View відображає інформацію користувачу, а ViewModel є проміжним зв'язком між ними, який забезпечує зв'язок між View та Model.

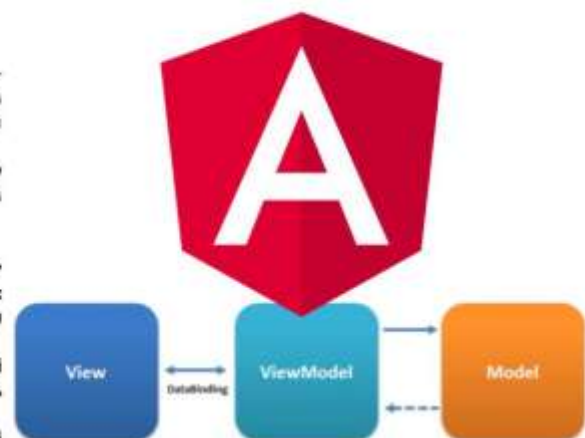
Angular використовує цю архітектурну модель для:

Розділення відповідальностей: Це дозволяє розбити програму на компоненти, які відповідають за відображення даних (View), їх обробку та маніпулювання (ViewModel), а також за управління даними (Model).

Збереження чистоти інтерфейсу: View незалежний від моделі даних і логіки додатку. ViewModel служить як прослойка, яка містить логіку обробки даних та забезпечує комунікацію між Model і View.

Зручне тестування: Через розділення логіки додатку на окремі частини, тести можуть бути написані окремо для кожного шару: для Model, ViewModel і View, що полегшує тестування і підтримку коду.

Отже, Angular використовує підхід MVVM для поліпшення структури програм та робить їх більш масштабованими, зручними для тестування та обслуговування.



Структурна MVVM

Рисунок Б.8 – Слайд № 8



Висновки

У процесі виконання даної дипломної роботи були вивчені та проаналізовані сучасні технології та інструменти, спрямовані на оптимізацію систем розрахунку заробітної плати. Виявлено, що використання Angular, Tailwind, Jasmine та Firebase Realtime Database становить ключовими факторами для створення високоєфективного та надійного програмного забезпечення для розрахунку заробітної плати.

Архітектура системи була детально розглянута, виключаючи розподіл компонентів та їх взаємодію. Побудована структура системи відображає оптимізовану організацію функцій та даних для зручного та ефективного управління розрахунками.

Проаналізовано можливості та переваги використання кожної з технологій, зокрема Angular для створення потужного користувацького інтерфейсу, Tailwind для швидкої та ефективної розробки дизайну, Jasmine для виявлення та виправлення помилок у ранніх етапах розробки та Firebase Realtime Database для забезпечення актуальних даних у режимі реального часу.

Основні функціональні вимоги та інтерфейс користувача були докладно розглянуті та виконані, забезпечуючи зручний доступ до основних опцій розрахунку, управління робочим часом та відпустками, а також можливість адміністрування користувачів.

У результаті розробки програмного забезпечення для системи розрахунку заробітної плати використання вищезгаданих технологій забезпечило не тільки високу якість продукту, а й актуальність та ефективність його реалізації. Розроблені рішення відображають потужний потенціал цих технологій у вдосконаленні процесу розробки програмного забезпечення для систем розрахунку, та можуть слугувати важливим внеском у розвиток цієї галузі.

Рисунок Б.9 – Слайд № 9