

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Автоматизація робочого графіку»

Виконав студент групи ІПЗ-19-3
ЛЯШУК Максим Володимирович

Керівник роботи:
КРАВЧЕНКО Світлана Миколаївна

Рецензент:
ЄФРЕМОВ Юрій Миколайович

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення

Андрій МОРОЗОВ

«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЛЯШУК Максим Володимирович**

Керівник роботи: **КРАВЧЕНКО Світлана Миколаївна**

Тема роботи: **«Автоматизація робочого графіку»**

затверджена Наказом закладу вищої освіти від «14» лютого 2023 р., №70с

Вихідні дані для роботи: **дослідження методів та технологій використання в розробці веб-додатку процесу спрощення/автоматизації певних процесів всередині компаній.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Кравченко С.М.	16.02.2023р.	16.02.2023р.
2	Кравченко С.М.	16.02.2023р.	16.02.2023р.
3	Кравченко С.М.	16.02.2023р.	16.02.2023р.

Календарний план

№ з/П	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі	16 лютого 2023 – 22 лютого 2023	Виконано
2	Пошук, огляд та аналіз аналогічних розробок	22 лютого 2023 – 10 березня 2023	Виконано
3	Формулювання технічного завдання та опрацювання літературних джерел	10 березня 2023 – 18 березня 2023	Виконано
4	Проектування структури системи	18 березня 2023 – 28 березня 2023	Виконано
5	Написання програмного коду	28 березня 2023 – 10 червня 2023	Виконано
6	Тестування роботи додатку	10 червня 2023 – 20 червня 2023	Виконано
7	Попередній захист	3 травня 2023	Виконано

РЕФЕРАТ

Випускна робота бакалавра складається з програмного продукту системи «Автоматизація робочого графіку» та пояснювальної записки. Пояснювальна записка до випускної роботи містить 72 сторінки, 43 ілюстрації та 1 таблиця.

Метою роботи є розробка веб-додатку «Автоматизація робочого графіку», що полегшить взаємодію організацій з їх співробітниками, а також надасть змогу користувачам власноруч здійснювати пошук організацій та подавати заяви на працевлаштування, якщо організація має відкриті вакансії.

В роботі визначено основні завдання на розробку системи та проаналізовано аналоги розробленої системи. Наведено сценарії роботи програмного продукту, загальна структура програмного продукту, опис алгоритмів взаємодії окремих, опис об'єктної структури системи (діаграма класів). Також описана програмна реалізація та наводяться результати тестування системи.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, АВТОМАТИЗАЦІЯ, ОДНОСТОРІНКОВИЙ ДОДАТОК, ГРАФІК, ОРГАНІЗАЦІЯ.

					ІПЗ.КР.Б – 121 – 23 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Автоматизація робочого графіку Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		М. Ляшук						
Керівник		С. Кравченко					2	
Н. Контр.		О. Тичина				Житомирська політехніка, група ІПЗ-19-3		
Зав. каф.		А. Морозов						

ABSTRACT

The bachelor's thesis consists of the software product of the Work Schedule Automation system and an explanatory note. The explanatory note to the thesis contains 72 pages, 43 illustrations and 1 table.

The purpose of the work is to develop a web application "Automation of the work schedule", which will facilitate the interaction of organizations with their employees, as well as enable users to search for organizations themselves and submit applications for employment, if the organization has open vacancies.

The work defines the main tasks for the development of the system and analyzes the analogues of the developed system. The scenarios of the software product, the general structure of the software product, the description of individual interaction algorithms, the description of the object structure of the system (class diagram) are given. The software implementation is also described and the results of system testing are given.

KEY WORDS: WEB APP, AUTOMATION, SINGLE PAGE APP, SCHEDULE, ORGANIZATION.

					ПЗ.КР.Б – 121 – 23 – ПЗ	Арк.
						3

ЗМІСТ

РЕФЕРАТ	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	8
ВСТУП	9
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ АВТОМАТИЗАЦІЙ РОБОЧИХ ГРАФІКІВ АБО ПРОЦЕСІВ	11
1.1. Постановка задачі	11
1.2 Аналіз аналогів програмного продукту	10
1.3 Визначення архітектури веб-додатку	15
1.3.1 Визначення архітектури серверної частини веб-додатку	16
1.3.2 Визначення структури клієнтської частини веб-додатку	20
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	19
Висновки до розділу 1	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ «АВТОМАТИЗАЦІЯ РОБОЧОГО ГРАФІКУ»	22
2.1. Визначення варіантів використання та структури системи	24
2.2. Розробка бази даних	29
2.3 Проектування та реалізація алгоритмів роботи додатку	30
2.4. Реалізація функціоналу додатку	31
2.4.1 Реалізація основної архітектури додатку	33
2.4.2 Реалізація компонента для роботи із файлами	37
2.4.3 Реалізація та налаштування конфігураційного файлу для підключення до бази даних	39
2.4.4 Реалізація контролера та створення ендпоінту на сервері	40
Висновки до розділу 2	40

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ «АВТОМАТИЗАЦІЯ РОБОЧОГО ГРАФІКУ»	41
3.1. Порядок встановлення та налаштування параметрів додатку	41
3.2. Структура інтерфейсу веб-додатку та порядок роботи	45
3.2.1. Інтерфейс та порядок роботи з панеллю адміністратора	47
3.2.2. Інтерфейс та порядок роботи з основною частиною додатка	54
3.3. Тестування роботи веб-додатку	60
Висновки до розділу 3	62
ВИСНОВКИ	63
ЛІТЕРАТУРА	65
ДОДАТКИ	68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CRUD – стандартний набір операцій взаємодії із базою даних

REDUX – бібліотека для управління станом веб-додатку яка базується на патерні Flux.

AXIOS – бібліотека JavaScript яка надає зручний спосіб для надсилання запитів браузеру або серверу.

API – програмний інтерфейс додатка

STRAPI - фреймворк для створення гнучких API

DOM – об'єктна модель документа

SPA – одно сторінковий додаток

БД – база даних

ПК – персональний комп'ютер

СУБД – система управління базами даних

ЕНДПОІНТ – посилання на сервер для зв'язку з базою даних

ВСТУП

Актуальність теми. В наш час все більше з'являється різноманітних додатків, які полегшують ті чи інші процеси в нашому житті. Будь-який сучасний підприємець прагне осучаснити свій бізнес та полегшити процес його керування. Тому додаток «Автоматизація робочого графіку» є одним із тих, що покращує та полегшує взаємодію власника організації з його робітниками.

Часто робітники будь-якої організації можуть не знати де отримати ту чи іншу інформацію щодо робочого процесу. Вони починають шукати відповідну інформацію у колег, керівників або адміністрації організації, що буває не легкою задачею. Для цього потрібно знайти контактну інформацію відповідних осіб, які не завжди мають час допомогти кожному, що призводить до певних проблем. Звісно в кожній організації є певний механізм повідомлення робітників тією чи іншою інформації, але він не завжди працює коректно, тому у робітників виникають питання.

Якщо взяти до уваги підприємця який має декілька організацій або філіалів однієї організації, то з кожною такою установою збільшується кількість інформації та робочих моментів. Це все стає не структурованим та не згрупованим, що призводить до певних проблем. Виникає проблема систематизації цього об'єму інформації, її управління та донесення до кожного робітника. Додаток «Автоматизація робочого графіку» може вирішити в повній мірі ці проблеми. Система дозволить підприємцю створювати сторінки своїх організацій та зручно керувати ними. Також за допомогою додатку користувачі зможуть знаходити будь-яку доступну організацію, ознайомлюватися з нею та подавати заявки на роботу, якщо відповідна установа має відкриті вакансії.

Метою випускної роботи є проектування та розробка веб-додатку «Автоматизація робочого графіку» призначений для полегшення управління організаціями та взаємодії з робітниками.

З мети можна визначити наступні завдання:

					ІПЗ.КР.Б – 121 – 23 – ПЗ	Арк.
						7

- визначення оптимального способу подання функціоналу для вирішення вище описаних проблем;
- визначення архітектури та структури системи для реалізації веб-додатку;
- обґрунтування та вибір засобів реалізації системи;
- реалізація веб-додатку «Автоматизація робочого графіку»;
- оптимізація додатку, задля прискорення взаємодії користувача з веб-додатком;
- тестування веб-додатку та аналіз отриманих результатів у вигляді висновків.

Об’єктом дослідження є процес спрощення/автоматизації певних процесів всередині компаній.

Предметом дослідження є методи та технології для використання у розробці веб-додатку «Автоматизація робочого графіку».

В роботі використано мову програмування JavaScript, фреймворк NextJS (базується на ReactJS), середовище виконання NodeJS та його фреймворк Strapi для створення гнучких API.

Запропонований додаток спроектовано, протестовано та готовий до експлуатації, може бути використаний для управління організацією або пошуку роботи для звичайного користувача. Додаток не є «суворо» прив’язаним до однієї організації та може бути використаний для будь якої організації (або декількох) в межах свого функціоналу.

Випускна робота складається з списку використаних джерел із 30 найменувань, 1 додатку, 1 таблиці та 43 ілюстрації по тексту, з них 0 оформлено на окремій сторінці. Повний обсяг роботи становить 72 сторінки, з них 67 сторінок основного тексту.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ТЕХНОЛОГІЙ АВТОМАТИЗАЦІЙ РОБОЧИХ ГРАФІКІВ АБО ПРОЦЕСІВ

1.1. Постановка задачі

Основною метою створення системи «Автоматизації робочого графіку» є систематизація інформації організації, відповідно створення її та наповнення всім необхідним, що передбачено функціоналом. Реалізація веб-додатку допоможе робітникам взаємодіяти з організацією в якій робітник працює та вчасно отримувати необхідну інформацію щодо робочого процесу.

В випускній роботі необхідно розробити веб-додаток, який зможе виконувати задачі автоматизації робочого графіку та процесу, структурований виклад інформації та полегшити користувачам взаємодію з установою де користувач працює, а також додаток повинен бути простим і зрозумілим для будь-якого користувача який би хотів скористатися ним.

Для реалізації поставленого завдання, основними етапами є:

1. Визначення вимог та функціоналу додатку
2. Проектування дизайну інтерфейсу користувача
 - 2.1. Створення дизайну сторінок для користувачів
 - 2.2. Створення дизайну адміністративної частини додатку
 - 2.3. Адаптація інтерфейсу під мобільні пристрої
3. Проектування та розробка основної архітектури додатку
 - 3.1. Розробка серверної частини
 - 3.2. Розробка клієнтської частини
4. Розробка функціоналу та сторінок додатку
 - 4.1. Підключення до бази даних
 - 4.2. Функціонал авторизації та реєстрації користувачів
 - 4.3. Зміна усіх особистих даних у особистому кабінет
 - 4.4. Компоненти для обробки зображень перед їх завантаженням на сервер
 - 4.5. Сховища яке контролює стан додатку

- 4.6. Модулів які необхідні для запитів до бази даних
- 4.7. Пошук користувачів та організацій
- 4.8. Адміністративної частини з усім необхідним функціоналом
- 4.9. Впровадження необхідних компонентів та утиліт

5. Розробка скриптів для налаштування конфігурації

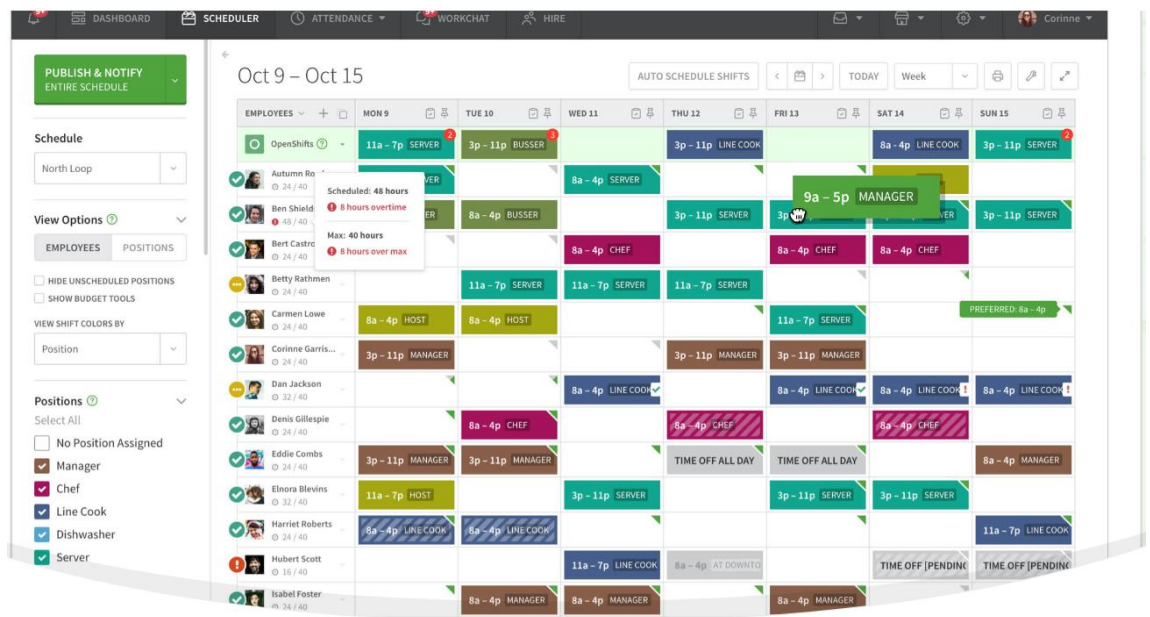
Результатом реалізації поставленого завдання є веб-додаток «Автоматизація робочого графіку», який систематизує інформацію організації, а також полегшить комунікацію власника з його робітниками.

1.2 Аналіз аналогів програмного продукту

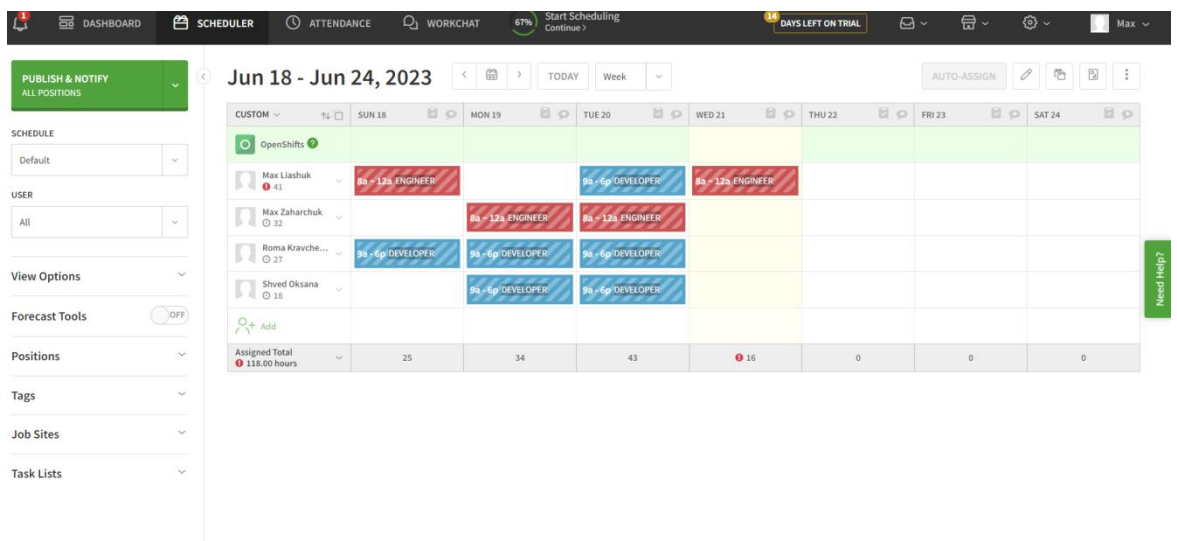
Було проведено аналіз аналогів програмного продукту, серед вже існуючих веб-додатків та визначено їх переваги та недоліки.

Потреба в пошуку організацій для працевлаштування та комунікації керівництва з робітниками виникає досить часто. Кожен хто зіштовхувався з цим власноруч, знає як важко буває знайти інформацію в мережі або інших ресурсах, а також знайти саме актуальну та достовірну інформацію.

Одним з додатків, який надає такий функціонал є «wheniwork.com». Основною функцією даного веб-додатку є повноцінний контроль робочих графіків та графіків відпусток, а також управління змінами працівників (рис. 1.1). Додаток є іноземним тому основна мова інтерфейсу є англійська. Наявний зручний функціонал для створення, редагування та видалення графіків, а також управління робітниками.



В додатку на головній сторінці відразу можна створити графік для відстеження власного часу навіть не будучи власником компанії. Також ви можете створити графік як власник компанії. Заповнивши усю необхідну інформацію додаток створить графік.



Інформація щодо графіків надається майже в повному обсязі: присутній час роботи на кожен день для кожного робітника, вихідні, посади робітників. Функціонал досить зручний та зрозумілий у використанні.

Інтерфейс додатку є адаптивним та зручним як на мобільному пристрої так і на великих екранах. Інтерфейс додатку та всі події доступні лише на

українській мові, іноземні туристи не зможуть скористатись даним додатком для пошуку інформації.

Іншим досить популярним аналогом є веб-додаток «www.deputy.com» (рис.1.3). Додаток являє собою схожий з попереднім, також надає можливість створення графіків, контролювання їх та динамічне змінення. Цей додаток також є іноземним, тому мова інтерфейсу є англійською, але він є максимально зрозумілим та не важкий в освоєнні.

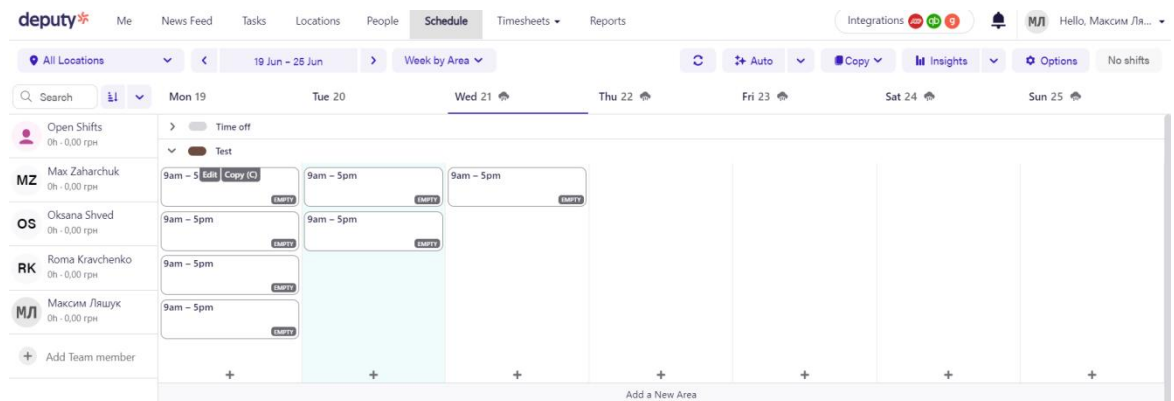


Рисунок 1.3 – Робочий графік у веб-додатку «www.deputy.com»

Інформація в графіку наведена досить детально: усі необхідні позиції, до кожного користувача можливо додати окремий час роботи та контролювати його робочий час. Також можливо створювати декілька філіалів організації.

При створенні нового філіалу організації вам відкривається повноцінне керування кожною із точок. Так ви маєте змогу розділяти та структурувати інформацію кожного філіалу організації.

В додатку присутній увесь необхідний функціонал для взаємодії між робітником та керівником, але не вистачає функціоналу по працевлаштуванню, адже це також є невід'ємною частиною будь-якої організації. Інтерфейс є простим та зрозумілим у використанні.

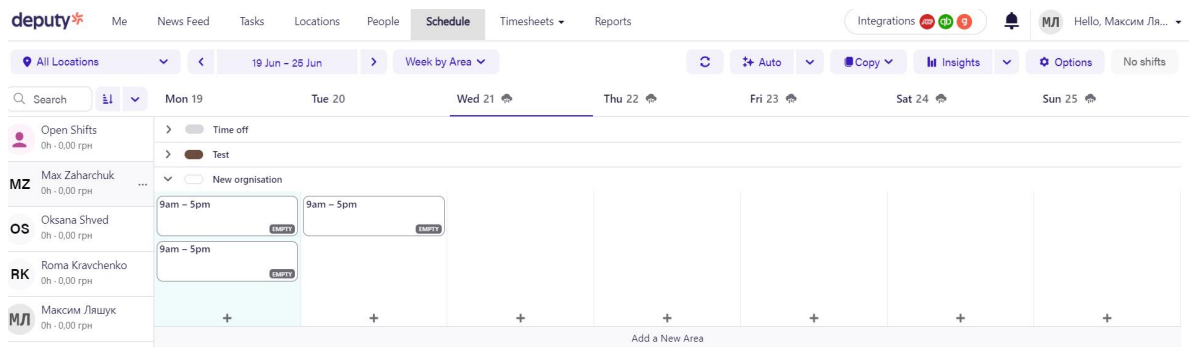


Рисунок 1.4 – Створення нового філіалу організації та графіку до нього

«app.shiftbase.com» додаток який пропонує достатній функціонал для роботи з графіками (рис. 1.5).

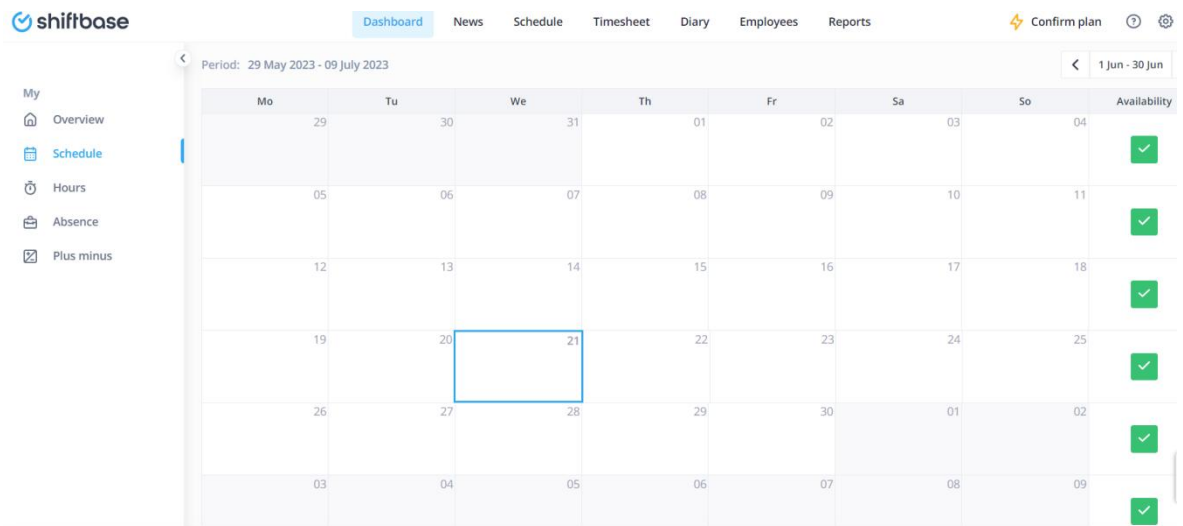


Рисунок 1.5 – Робочий графік у веб-додатку «app.shiftbase.com»

Додаток надає змогу зареєструватися як власник компанії, створити компанію, додати робітників та їх посади. Далі ви можете створити графік де можете керувати змінами робітників та динамічно змінювати усю необхідну інформацію. Додаток як і попередній не має функціоналу по працевлаштуванню, що безперечно є мінусом:

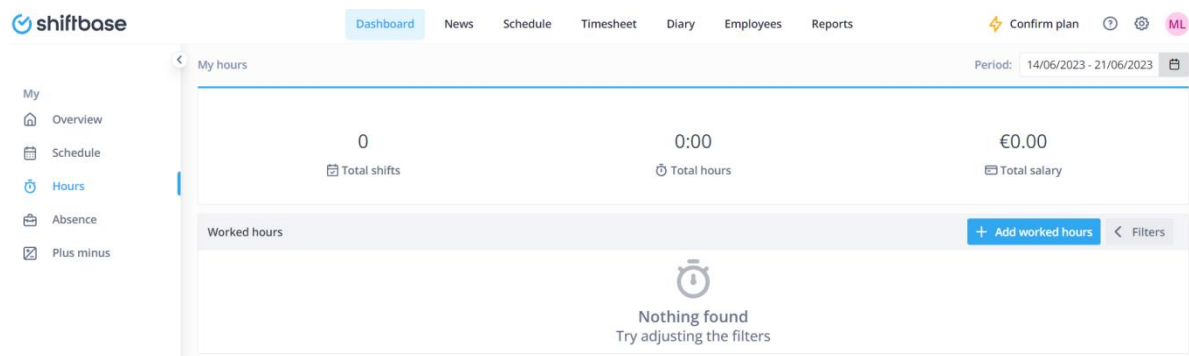


Рисунок 1.6 – Бухгалтерія згідно графіку

Цей додаток має підрахунок виплат зарплат для робітників, що є зручним для контролю оплати праці кожного з робітників. Також додаток підраховує робочий час в той чи інший день або місяць. Наявна відповідна вкладка яка контролює відсутність користувача, а також бухгалтерія яка контролює доходи та витрати за певний період часу.

Інтерфейс додатку одномовний, але функціонал досить зрозумілий та простий у використанні. Також він є адаптивним та зручним як на мобільних пристроях, так і на широких екранах.

Для порівняння вище описаних веб-додатків було створено порівняльну таблицю 1.1.

Таблиця 1.1

Порівняння веб-додатків

Характеристика	wheniwork.com	www.deputy.com	app.shiftbase.com
Робочий графік	Зручний і функціональний	Функціональний, але не зовсім зручний	Не зовсім зручний
Графік відпусток	Зручний і функціональний	Функціональний, але не зовсім зручний	Не зовсім зручний
Заяви на працевлаштування	Відсутні	Відсутні	Відсутні

Адаптивність інтерфейсу	Адаптивний	Адаптивний	Адаптивний
Бухгалтерія	Лише підрахунок годин	Лише підрахунок годин та зарплат	Повноцінна на всі прибутки та витрати
Декілька організацій	Можливо додати	Можливо додати	Можливо додати
Функціональність	Багатий функціонал	Багатий функціонал	Середній функціонал

Розглянувши таблицю 1.1, можна зробити висновок, що кожен додаток виконує лише частину поставлених задач. «wheniwork.com» в повній мірі виконує функції графіків, але не має повноцінної бухгалтерії. «www.deputy.com» є непоганий по функціоналу та зручності, але не має заяв на працевлаштування. «app.shiftbase.com» має добре зроблені підрахунки прибутків та витрат, але не зовсім зручний функціонал графіків. Але жоден з додатків не виконує усі поставлені задачі одночасно, що може бути не зручним для користувачів.

За результатами порівняльного аналізу було виявлено недоліки та переваги кожного з додатків для подальшого проектування веб-додатку «Автоматизація робочого графіку».

1.3 Визначення архітектури веб-додатку

Одним з завдань випускної роботи є визначення основної архітектури веб-додатку. Згідно до вимог до продукту, було визначено, що результатом цієї роботи має бути веб-додаток. Так як додаток має зберігати та систематизувати дані, а також надавати можливості редагування даних лише деяким групам користувачів, було вирішено використати клієнт-серверну архітектуру додатку.

В цьому випадку вся необхідна логіка буде прихована на сервері, а на клієнті дані будуть відображатись в зручному для користувача вигляді.

1.3.1 Визначення архітектури серверної частини веб-додатку

Для серверної частини додатку були розглянуті такі архітектурні підходи:

- Монолітна архітектура
- Мікросервісна архітектура
- Модульна архітектура

Проаналізувавши завдання та необхідний функціонал, було визначено, що мікросервісна архітектура не є доцільною для розробки веб-додатку «Автоматизації робочого графіку». Мікросервісна архітектура буде більш доцільною для масштабніших додатків, які містять в собі функціонал, який може працювати окремо один від одного. Дана архітектура занадто навантажувальна для додатку такого типу і лише сповільнить розробку додатку. Крім того, частини даного веб-додатку будуть досить залежні один від одного.

Модульна архітектура є не поганий варіант для розробки відповідного додатку та є простішою за мікросервісну архітектуру. Така архітектура дає більше контролю над серверною частиною, адже робить модулі майже не залежними одне від одного, що є досить добре у масштабованості та додаванні нового функціоналу в майбутньому. Не зважаючи на всі переваги даної архітектури, вона не повністю підходить для обраної технології написання серверної частини, а саме фреймворку Strapi [1].

Монолітна архітектура значно пришвидшить розробку даного веб-додатку. Але така архітектура важко масштабується, тобто ускладниться процес розробки нового функціоналу додатку. Хоч і Strapi базується на монолітній архітектурі було прийнято рішення робити контролери максимально не залежними як в модульній архітектурі, що дасть більш гнучку розробку та кращу масштабованість. Така архітектура не сповільнює написання додатку, спрощує його написання:

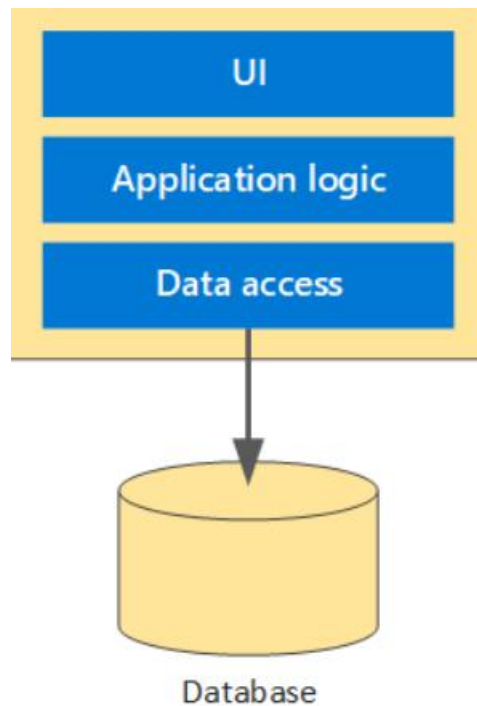


Рисунок 1.7 – Монолітна архітектура

За монолітної архітектури код серверної частини побудований як єдиний і неподільний компонент, містить свою логіку та інше. Всі функціональні модулі і компоненти додатку працюють разом в одному процесі. Також додаток буде менш складним і простішим в розробці ніж при мікросервісній архітектурі. Пришвидшиться процес ознайомлення з структурою додатку, для нових розробників.

Відповідні контролери не є повністю незалежними, але кожен з них має свою логіку яка не має впливу на інші компоненти. Вони працюють усі як єдиний механізм та залежать від бази даних, тобто не маючи доступу до бази даних контролер нічого не поверне клієнтській частині.

Орієнтовна структура одного компонента буде складатись з:

- контролера;
- контент-менеджера, де описана повна структура сутності (схема);
- посилань (ендпоінтів) для запитів до контролера та взаємодії з даними;
- сервіси;

Кожен компонент може мати в контролері декілька обробників кожен з яких буде відповідати за певний ендпоінт запиту до бази даних [11].

Отже, монолітна архітектура є найбільш оптимальним варіантом серед розглянутих архітектур. З використанням такої архітектури, додаток буде простим в розробці, складніше масштабуватиметься ніж при модульній архітектурі, але компоненти будуть проектуватися максимально не залежним.

1.3.2 Визначення структури клієнтської частини веб-додатку

Визначивши архітектуру серверної частини додатку як монолітна, задля спрощення розробки, а також подальшої розробки нового функціоналу, варто використати модульну архітектуру на клієнтській частині.

Тобто, якщо на сервері є компонент «vacations», який відповідає за функціональність графіків відпусток, то на клієнтській частині буде модуль «vacations», який міститиме в собі все необхідне для відображення та взаємодії з графіками відпусток.

Такий підхід дозволить швидше орієнтуватись в програмному коді додатку, а також прискорить розробку нового функціоналу для додатку. Також, за потреби, простіше можна буде розділити обов'язки між декількома розробниками або командами проекту:

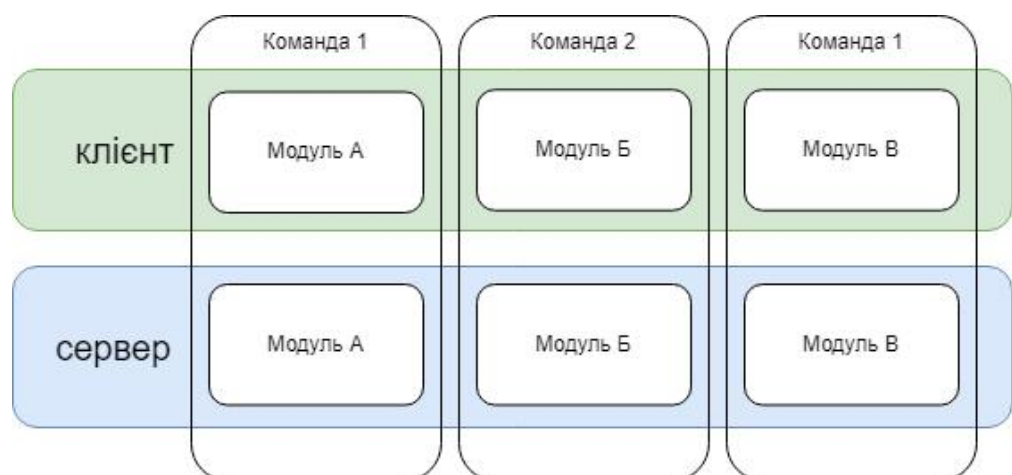


Рисунок 1.8 – Розподіл обов'язків

Також було досліджено види клієнтських додатків, а саме:

- SPA (односторінковий додаток)

- MPA multi page application (багатосторінковий додаток)

При використанні SPA, переходи між сторінками не потрібні та сторінка не перезавантажується. Для реалізації такого підходу використовується мова «JavaScript». Головною перевагою такого підходу є швидкість, адже не потрібно кожен раз завантажувати дані, які з'являються на кожній сторінці, оновляться лише необхідні компоненти.

Недоліками SPA є повільне перше завантаження, через те, що файли-скрипти таких сторінок є досить складними і великими, та займають деякий час для завантаження. Також можуть виникнути проблеми у користувачів, які відключили JavaScript в налаштуваннях [26].

Багатосторінковий додаток перезавантажує всю сторінку повністю при переході між сторінками, але також допускається завантаження даних динамічно. Такий підхід має недоліки, серед них:

- Пов'язаність клієнтської та серверної частини між собою
- Складність розробки
- При кожному оновленні сторінки завантаження усіх даних навіть якщо вони не змінювалися
- Складна розробка адаптивного інтерфейсу

Було вирішено розробити клієнтський додаток як односторінковий додаток, тобто SPA взявши до уваги усі переваги та недоліки обох підходів.

Отже, було визначено, що клієнтська частина буде реалізовано за допомогою підходу SPA додатку, який буде структурований подібно компонентам серверної частини. Такий підхід дозволить значно спростити розробку, а також пришвидшити завантаження сторінок.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Так як була обрана монолітна архітектура для написання серверної частини веб-додатку, було вирішено використати Strapi. Strapi – це фреймворк

NodeJS для створення гнучких серверних додатків. Вона призначена для створення монолітних додатків, також підтримується всі інші архітектури і їх можливо впровадити при бажанні. Також існує багато бібліотек, що розширюють функціонал платформи [1, 3].

Клієнтську частину вирішено написати на мові JavaScript, а саме на фреймворку NextJS, що компілюється в JavaScript. Перевагою даного фреймворку є простота у використанні, серверний рендеринг сторінок, що додає швидкості додатку та великий функціонал, що дає не залежність для даного фреймворку [2].

Також було обрано бібліотеку AXIOS для запитів на сервер та взаємодії з базою даних, спрощення процесу написання запитів до бази даних. Як СУБД було обрано PostgerSQL, так як дана СУБД підтримує «json» поля в таблицях, має багато зручних типів для використання та навіть надає можливість створювати власні типи і підтримується фреймворком Strapi [5, 8].

Було сформовано системні вимоги, за умови виконання яких, система працюватиме стабільно:

1. Вимоги до середовища виконання:

Система повинна задовольняти вимогам на комп'ютері, що знаходиться в наступній мінімальній комплектації:

- операційна система Windows 7 або вище, Ubuntu 16 або вище;
- 64-розрядний (x64) процесор із тактовою частотою 1 ГГц або вище;
- 2 ГБ ОЗУ;
- 40 ГБ (для 64-розрядної системи) вільного місця на жорсткому диску;

2. Вимоги до СУБД та доступу до даних:

- у ядрі системи повинна бути представлена СУБД PostgerSQL.

3. Вимоги до клієнтського пристрою

- операційна система з підтримкою браузерів (або їх наступних версій):
 - Chrome 50
 - Firefox 48

- Edge 17
- Opera 37
- 1 ГБ ОЗУ

Висновки до розділу 1

В даному розділі було проведено аналіз предметної області та проаналізовано вимоги до додатку. Поставлено задачу та визначені етапи виконання поставленого завдання.

Проведено порівняльний аналіз аналогів, виокремленні їх переваги та недоліки для подальшого використання при проектуванні додатку «Автоматизація робочого графіку».

Проаналізовано архітектурні підходи до розробки веб-додатків, визначено, що відповідний додаток буде розроблено як клієнт-серверний додаток. Серверна частина має бути розроблена як монолітний додаток.

Визначено структуру клієнтської частини, яка являтиме собою SPA додаток, який буде структурований подібно компонентам серверної частини.

Обраний підхід є оптимальним для вирішення поставлених задач, а також дозволить масштабувати додаток та розробляти новий функціонал в майбутньому.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ «АВТОМАТИЗАЦІЇ РОБОЧОГО ГРАФІКУ»

2.1. Визначення варіантів використання та структури системи

Визначено високо рівневі вимоги, якими повинна володіти система “Автоматизації робочого графіку”, щоб мати можливість безперервно виконувати покладені на неї функції. Зазначено умови, яких повинен дотримуватись користувач для роботи з даною системою. Приведені можливості системи, обмеження та показники якості.

Бізнес вимоги

1. Основні цілі: проект створюється з метою спрощення взаємодії між працівниками компанії та керівництвом.
2. Представлення проекту: проект буде реалізовано у вигляді веб-додатку, що містить зручний інтерфейс та структуровану інформацію по даній тематичній області.

Вимоги користувачів

1. Можливість створення та редагування власного профілю;
2. Можливість перегляду профілю співробітників;
3. Можливість робітниками вільно переглядати робочі графіки або графіки відпусток компаній де вони працюють;
4. Можливість подавати заяви на працевлаштування у компанії, які мають відкриті вакансії;
5. Можливість бачити новини компанії в якій користувач працює;
6. Можливість здійснювати пошук компаній;
7. Можливість бачити інформацію щодо своєї посади та заробітної плати;

Вимоги персоналу веб-додатку (власник):

1. Створення, видалення та редагування робочих графіків та графіків відпусток;

2. Можливість розглядання усіх заяв на роботу та відповідно їх погодження або відхилення;
3. Можливість створення бухгалтерії (контроль зарплат усіх посад та файлу з прибутками і витратами) і бачити чистий прибуток за весь місяць;

Характеристика об'єкта комп'ютеризації

Користувач додатку буде мати можливість створювати обліковий запис як звичайного робітника так і власника компанії. Звичайний робітник буде мати базовий функціонал, а власник компаній розширений, що надасть йому змогу ефективно управляти компанією або компаніями.

Функціональні вимоги:

1. Авторизація в системі: В системі повинна бути представлена можливість реєстрації користувача та присвоєння йому відповідної ролі (робітник або власник).
2. Можливість збереження інформації: Система повинна зберігати інформацію і надавати можливість керувати нею.

Нефункціональні вимоги:

1. Сприйняття
 - Час, потрібний для навчання інструментами роботи з інформаційною системою для звичайних користувачів – 1 робочий день (8 годин), а для досвідчених – чверть робочого дня (2 години)
 - Час відповіді системи для звичайних запитів не повинен перевищувати 5 секунд, а для більш складних запитів – 20 сек.
 - Інтерфейс представлення повинен бути інтуїтивно зручним для користувача та не вимагати від нього додаткової підготовки.
2. Надійність
 - Доступність – час, потрібний для обслуговування системи не повинен перевищувати 5% від загального часу роботи.
 - Середній час безперервної роботи – 20 робочих днів.

- Максимальна норма помилок та дефектів в роботі системи – 1 помилка на 1000 запитів користувача.

3. Продуктивність

- Система повинна підтримувати мінімум 100 одночасно працюючих користувачів, пов'язаних з спільною базою даних.

4. Можливість експлуатації

- Масштабування – система повинна мати можливість збільшувати потужності (продуктивність), зі збільшенням користувачів таким чином, щоб це аж ні як негативно не відобразилося на її роботі.

Аналіз вимог користувачів дав підстави для визначення варіантів використання додатку «Автоматизація робочого графіку». На рис. 2.1 наведено діаграму використання програмної системи.

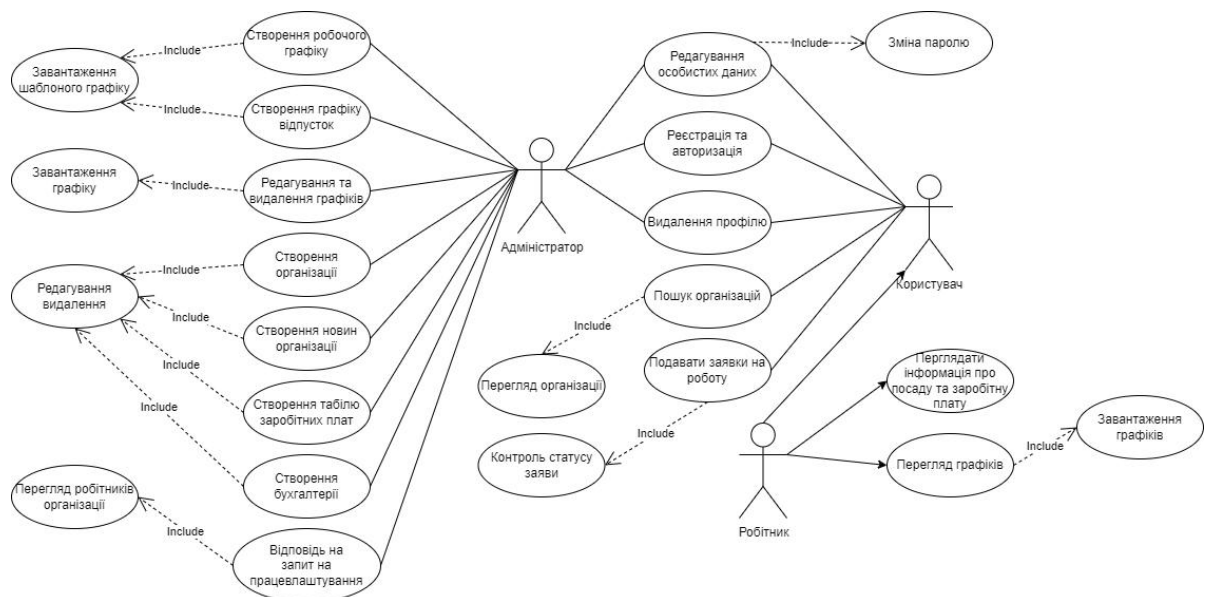


Рисунок 2.1 – Варіанти використання системи «Автоматизація робочого графіку»

Проаналізувавши функціональні вимоги, було сформовано діаграму компонентів (рис. 2.2), з яких буде складатиметься система, діаграму сутностей (рис. 2.3), що виділяє класи, які є представленням відповідних таблиць в БД.

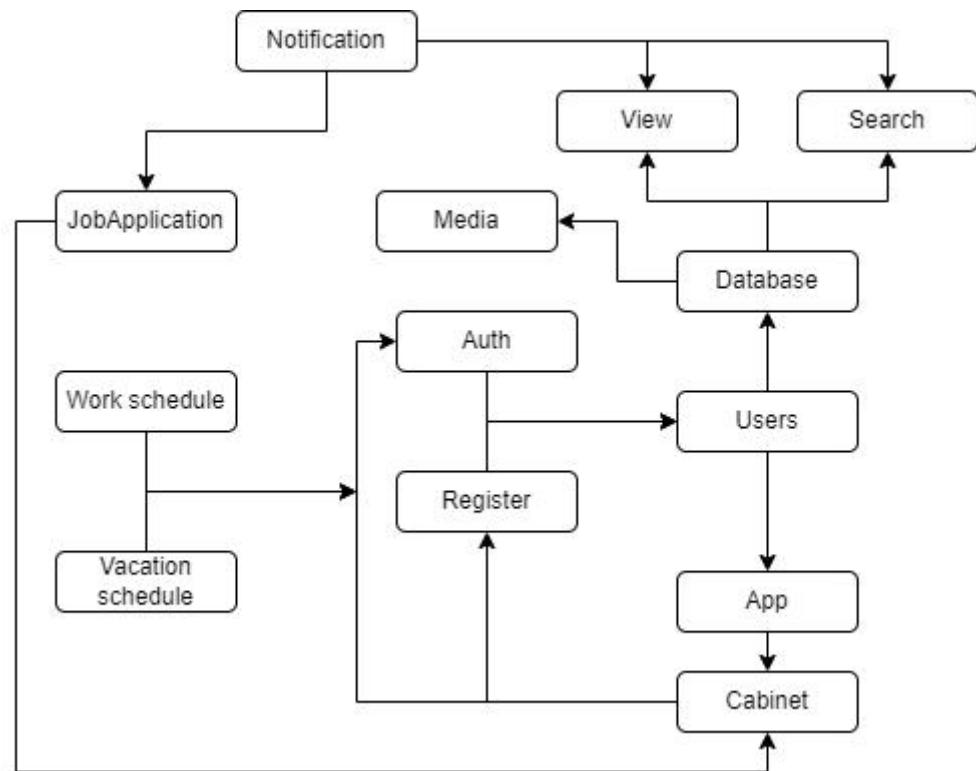


Рисунок 2.2 – Діаграма компонентів системи «Автоматизація робочого графіку»

На діаграмі компонентів зображені частини системи, кожен з компонентів виконує свою функцію. Компоненти можуть залежати один від одного, або від модулів з Strapi. Наприклад, компонент Cabinet залежить від компонентів Auth та Register, так як без авторизації в кабінет потрапити неможливо.

Було спроектовано 13 компонентів, а саме

1. App – вихідний компонент додатку, ініціалізує API додатку;
2. Users – компонент, що надає інформацію щодо користувачів;
3. Auth – компонент, що відповідає за авторизацію та інші задачі пов'язані з сесією користувача;
4. Register – компонент, що відповідає за реєстрацію користувача;
5. Cabinet – компонент, що відповідає за особистий кабінет користувача;
6. WorkSchedule – компонент відповідає за робочі графіки та залежить від авторизації користувача;

7. VacationSchedule – компонент, відповідає за графіки відпусток, та аналогічно залежить від авторизації користувача;
8. Database – відповідає за зв'язок та надсилання запитів до бази даних;
9. JobApplication – компонент відповідає за подання заяв на працевлаштування та їх обробку;
- 10.View – компонент для відображення даних із серверу;
- 11.Notification – компонент необхідний для відображення повідомлень користувачеві;
12. Search – відповідає за пошук організацій та користувачів;
- 13.Media – компонент для взаємодії з медіа наповненням та коректних запитів до БД пов'язаних з відповідним наповненням.

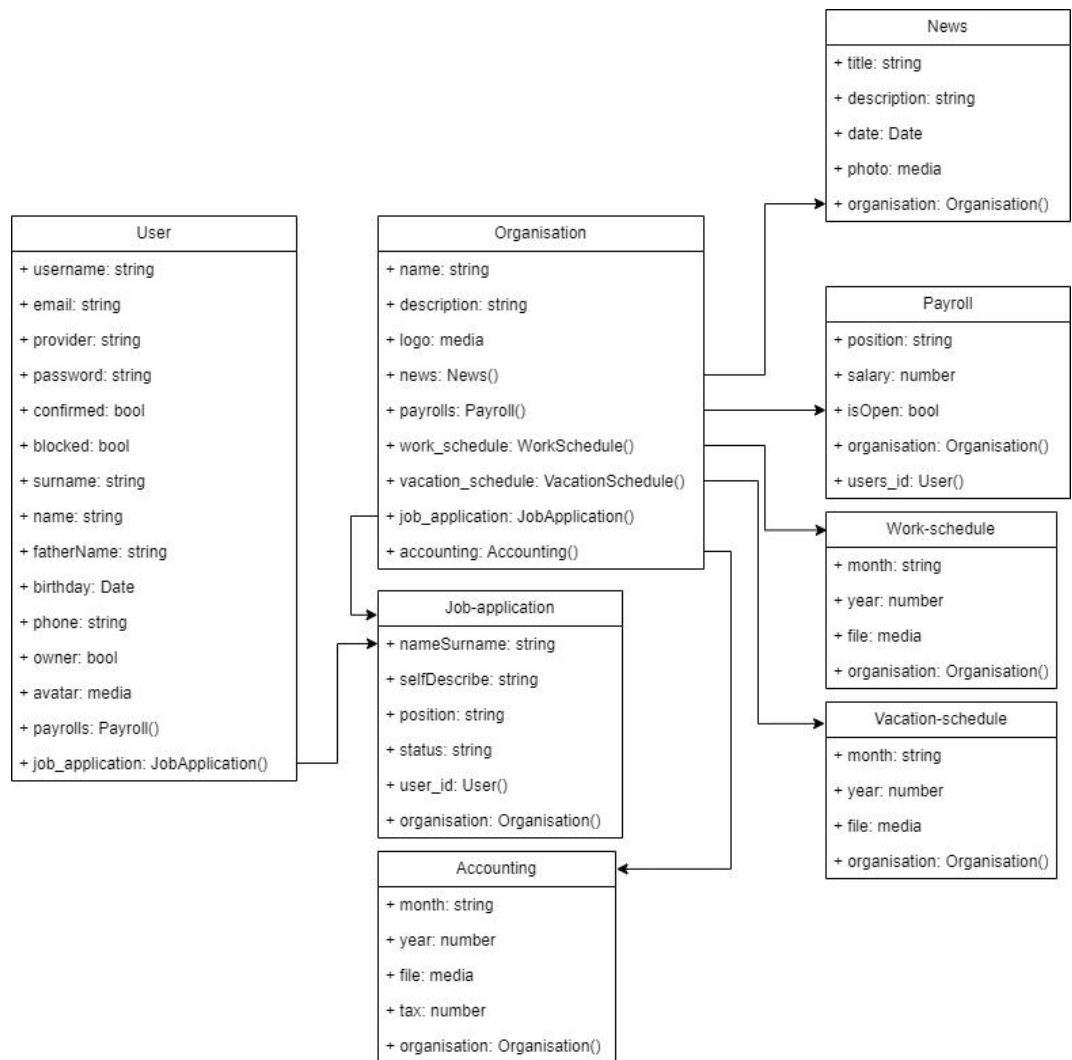


Рисунок 2.3 – Діаграма класів сутностей «Автоматизація робочого графіку»

Діаграма класів сутностей зображає лише ті класи, які мають відповідне представлення в базі даних. Сутності лише несуть інформацію та не мають методів. Операції над сутностями виконують контролери. В додатку представлені 8 сутностей:

1. User – зберігає дані користувача;
2. Organisation – зберігає дані організації;
3. News – новини певної організації;
4. Payroll – таблиць заробітних плат для кожної організації;
5. Work-schedule – зберігає робочі графіки організації;
6. Vacation-schedule – зберігає графіки відпусток організації;
7. Job-application – відповідає за збереження заяв на працевлаштування організації;
8. Accounting – зберігає бухгалтерію організації;

Залежності в компонентах вводяться з контейнеру, за реєстрацію залежностей відповідатиме Strapi. В кожному компоненті ми можемо зареєструвати залежності.

2.2. Розробка бази даних

Було вирішено, що як СУБД в додатку буде використано PostgreSQL. Також в додатку використано фреймворк Strapi, який генерує всі необхідні файли для взаємодії з базою даних [8].

Було створено таблиці в базі даних на основі сутностей, описаних в діаграмі сутностей (рис. 2.3). Результатом генерації стало 8 таблиць:

- user – таблиця для збереження даних користувачів;
- organisation – таблиця для збереження даних організації;
- payroll – допоміжна таблиця для зв'язку «багато до багатьох» між таблицями user та organisation;
- news – таблиця для новин організації;
- work-schedule – таблиця для робочих графіків організації;

- vacation-schedule – таблиця для графіків відпусток організації;
- job-application – таблиця для заяв на працевлаштування організації;
- accounting – таблиця бухгалтерії організації;

З вихідних таблиць була побудована діаграма бази даних:

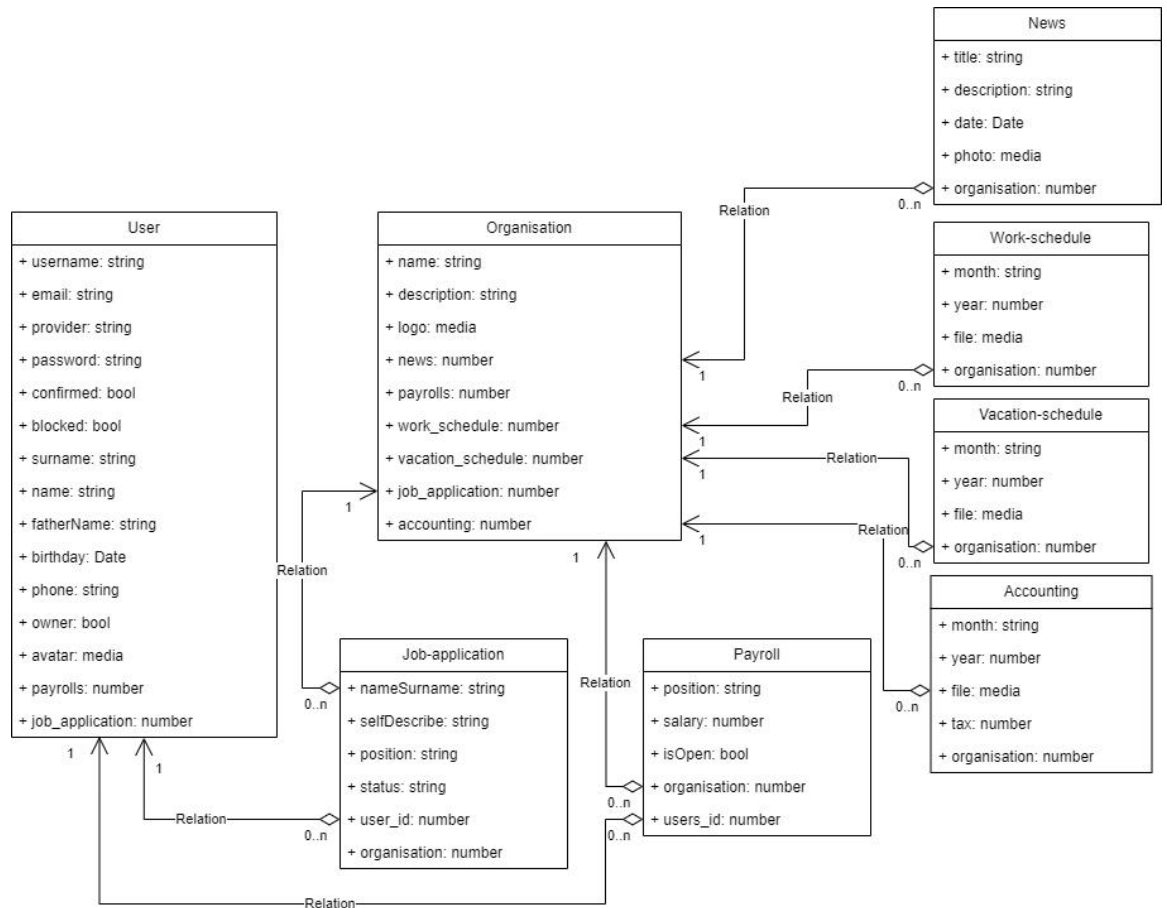


Рисунок 2.4 – Діаграма бази даних

На даній діаграмі продемонстровані усі таблиці та зв'язки між ними. Також таблиці не потребують каскадного видалення чи оновлення.

2.3 Проектування та реалізація алгоритмів роботи додатку

Так клієнтський додаток являтиме собою Single Page Application, було вирішено використати звичайний алгоритм SPA як основний (Рис 2.5). Потрапивши на сторінку додатку, сервер відправляє користувачу актуальну версію сторінки. Далі, в залежності від дій користувача, клієнт відправляє запити на сервер. Отримавши відповідь від серверу, клієнт змінює вигляд сторінки [26].



Рисунок 2.5 – Загальний алгоритм роботи додатку

Визначившись з основним алгоритмом додатку, потрібно визначити алгоритм для окремих випадків. Для моделювання таких випадків зазвичай використовується діяльності (activity diagram). Більшість сценаріїв використання є тривіальними і не потребують додаткового роз’яснення. Важливо спроектувати сценарії з складною логікою.

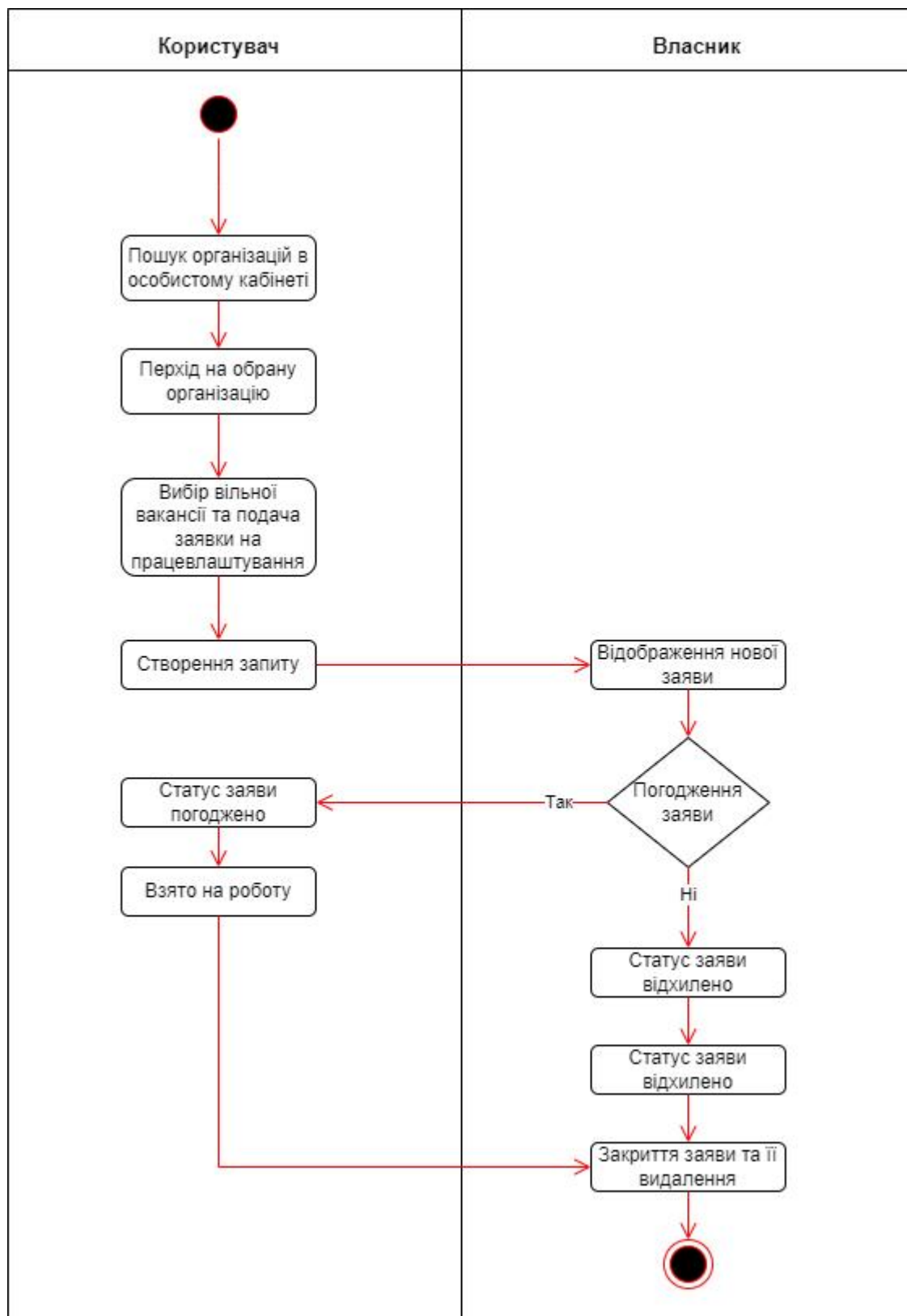


Рисунок 2.6 – Діаграма діяльності «Створення запиту»

2.4. Реалізація функціоналу додатку

2.4.1 Реалізація основної архітектури додатку

Додаток буде складатися з 2 частин: клієнтської та серверної, кожна з яких буде окремим додатком та знаходитиметься в окремому каталозі «next» та «strapi» відповідно.

Серверний додаток було створено за допомогою інтерфейсу командного рядка та фреймворку «Strapi». Створений проект міститиме одразу модуль та функцію для запуску, а також встановить усі необхідні бібліотеки для запуску додатку [1].

Вхідною точкою в додаток є функція «bootstrap» файлу «app.example.js». В даній функції створюється та налаштовується об'єкт додатку:

```
const bootstrap = (app) => {  
  console.log(app);  
};
```

```
export default {  
  config,  
  bootstrap,  
};
```

Далі йде модуль server який є головним модулем додатку. В даному модулі відбуваються налаштування для серверу та його завантаження:

```
module.exports = ({ env }) => ({  
  host: env('HOST', '0.0.0.0'),  
  port: env.int('PORT', 1337),  
  app: {  
    keys: env.array('APP_KEYS'),  
  },  
  webhooks: {  
    populateRelations: env.bool('WEBHOOKS_POPULATE_RELATIONS', false),  
  },  
});
```

Клієнтський додаток було створено за допомогою інтерфейсу командного рядка та фреймворку NextJS. Створений проект було очищено від не потрібних компонентів та залежностей, які створені за замовчуванням [2].

Вхідною точкою в додаток є файл `_app.js`. в якому створюється кореневий компонент `React`, реєструються всі шляхи до сторінок та постачальники контекстів:

```
function App({ Component, pageProps }) {  
  return (  
    <Provider store={store}>  
      <Layout>  
        <Component {...pageProps} />  
      </Layout>  
    </Provider>  
  );  
}
```

В даному фрагменті коду ми можемо побачити такі компоненти:

- `Provider` – компонент з бібліотеки `Redux`, що надає доступ до його стану;
- `Layout` – компонент, що надає обгортку для кожної із сторінок;
- `Component` – параметр який приходить в `_app.js` та є сторінкою на яку перейшов користувач;

Для зміни візуального відображення компонентів системи було використано бібліотеку «`styled-components`». Дана бібліотека дозволяє додавати компонентам `CSS`-властивості створюючи компоненти в стилях та використовуючи їх на сторінках. Бібліотека сканує файли проекту на наявність таких компонентів та додає їх в вихідний `CSS` [13].

Для підключення даної бібліотеки було встановлено бібліотеку «`styled-components`» за допомогою менеджера пакетів «`npm`». Також було додано налаштування для надання стилів дочірнім компонентам. Для створення стилів за допомогою цієї бібліотеки, потрібно лише створювати файл `styled.js` в каталозі з відповідним компонентом та використовувати відповідний синтаксис бібліотеки, далі буде приведено приклад стилів для контейнера додатку:

```
export const Container = styled.div`  
  display: flex;  
  justify-content: center;  
  align-items: start;  
  max-width: 1320px;  
  padding: 0 20px;
```

```

flex: auto;
@media (min-width: 0) and (max-width: 1200px) {
  margin: 0 auto;
}

@media (max-width: 1200px) {
  width: 1024px;
}

@media (max-width: 1024px) {
  width: 992px;
}

@media (max-width: 992px) {
  width: 768px;
}

@media (max-width: 768px) {
  width: 576px;
}

@media (max-width: 576px) {
  width: 100%;
}
`;

```

Також в клієнтській частині присутній глобальне сховище яке контролює стан додатку. Щоб забезпечити роботу такого сховища потрібно встановити бібліотеку «Redux» та всі необхідні для неї допоміжні бібліотеки, основною з яких було обрано «redux-thunk», адже ця бібліотека надає хороші можливості для асинхронних запитів до серверу, що покращує роботу додатку. Також можливо було задіяти таку бібліотеку як «redux-saga» яка є схожою до попередньої, але вона занадто велика за своїм розміром та є не доцільна при розробці додатку такого типу, адже вона буде його перевантажувати. Далі буде наведено приклад створення та налаштування сховища [6, 7]:

```

import { applyMiddleware, createStore } from 'redux';
import thunk from 'redux-thunk';
import rootReducer from './reducers';

const store = createStore(rootReducer, applyMiddleware(thunk));

export default store;

```

В даному файлі налаштування бачимо імпортування самого «thunk», що і надасть можливість для зручних запитів до серверу. Також метод для створення сховища «createStore» та middleware, який об'єднує в собі усі middleware «applyMiddleware».

Також для коректної роботи потрібні ред'юсери та дії, які будуть запускати певні послідовності в ред'юсерах. Для функціонування усіх ред'юсерів їх потрібно об'єднати в один батьківській ред'юсер, далі буде продемонстровано налаштування такого:

```
import { combineReducers } from 'redux';
const rootReducer = combineReducers({
  collection,
  user,
  users,
  employer,
  organisationsByName,
});
```

```
export default rootReducer;
```

Ці налаштування забезпечать контроль над станом додатку, що покращить його роботу та дозволить відтворювати тільки ті частини інтерфейсу користувача з якими він взаємодіє та в будь-якому місці додатку.

Таке розділення дозволить розділити файли додатку на багато маленьких частин і для користувача завантажувати лише потрібні частини при використанні:

Route (pages)	Size	First Load JS
λ /	6.09 kB	141 kB
λ /_app	0 B	135 kB
λ /[nameSchedule]/schedule/[id]	3.65 kB	248 kB
○ /404	2.56 kB	137 kB
λ /accounting/[id]	111 kB	356 kB
λ /cabinet	16.2 kB	265 kB
λ /news/[id]	2.28 kB	137 kB
λ /organisation/[id]	5.07 kB	140 kB
λ /user/[id]	3.31 kB	253 kB
λ /vacationSchedule	525 B	250 kB
λ /workSchedule	517 B	250 kB
+ First Load JS shared by all	135 kB	
chunks/framework-2c79e2a64abdb08b.js	45.2 kB	
chunks/main-4fe93428a0daae30.js	32.2 kB	
chunks/pages/_app-ab08c58731f21d58.js	55.4 kB	
chunks/webpack-f23de935ddf4b94c.js	1.85 kB	
css/d62bfb8008219d64.css	248 B	

Рисунок 2.7 – Файли додатку після розділення

2.4.2 Реалізація компонента для роботи із файлами

Так як було вирішено розробити додаток за технологією NextJS, яка в свою чергу базується на ReactJS, то відповідно потрібно створювати компоненти, які можливо повторно використовувати в місцях де цього потребує той чи інший функціонал. Одним із таких компонентів став компонент для взаємодії з файлами та їх завантаженням до бази даних [2, 4].

Розглянемо серверну утиліту яка допомагає коректно обробити файл не залежно від його розширення та додати його до бази даних, а також в відповідну таблицю.

```
const formData = new FormData();

formData.append('files', dataCreate[mediaField][0]);
const uploadedFile = await apiRequest.post(`/api/upload`, formData, {
  headers: {
    'Content-Type': 'multipart/form-data',
  },
});
```

```
const {
  data: { data },
} = await apiRequest.post(`${endPoint}`, {
  data: { ...dataCreate, [mediaField]: uploadedFile.data[0].id },
});
```

В даному фрагменті коду виконується зчитування файлу який було передано до утиліти та записання його до спеціального об'єкту «formData», що надасть зручну взаємодію з відповідним файлом. Далі утиліта додає файл в базу даних за допомогою запита який є за замовчуванням у Strapi. Після додавання відповідного файлу до бази даних в зміну отримується повний об'єкт відповіді серверу, де є ідентифікатор потрібного нам файлу, але тепер вже на сервері. Маючи відповідний ідентифікатор залишається тільки по відповідно переданому ендпоінту виконати запит на створення запису з файлом, а саме його ідентифікатором на сервері та зв'язати запис з відповідним файлом.

Також було створено компонент для візуального відображення на клієнтській частині:

```
export default function MediaField({ fileName, register, accept, isXL }) {
  return (
    <DropImage>
      {isXL ? (
        <BsFiletypeXls size={30} color="#40282C" />
      ) : (
        <RiImageAddFill size={30} color="#40282C" />
      )}
      <Notification>Перетяніть сюди файл або оберіть його з носія</Notification>
      <FileName>{fileName || 'Файл не обрано'}</FileName>
      <Button name="openFileManager">Обрати файл</Button>
      <input type="file" {...register} accept={accept} tabIndex={-1} />
    </DropImage>
  );
}
```

2.4.3 Реалізація та налаштування конфігураційного файлу підключення до бази даних

Цей модуль використовується для підключення Strapi до бази даних при запуску сервера та збірці всього додатку.

В конфігурації налаштування потрібно вказати певні параметри для успішного та захищеного підключення, що надасть доступ до усіх даних, які наявні у базі даних.

Дані продемонстровано конфігураційний файл:

```
const path = require('path');

module.exports = ({ env }) => ({
  connection: {
    client: 'postgres',
    connection: {
      host: 'localhost',
      port: 5432,
      database: "dyplom",
      user: "postgres",
      password: '060802',
    },
  },
  useNullAsDefault: true,
});
```

Як бачимо із даного файлу тут налаштовуються наступні параметри для підключення до бази даних:

- client - клієнту який створював базу даних;
- host - адреса локального хоста на якому буде запущено сервер;
- port - номер порту на якому буде запущено сервер;
- database - назва бази даних;
- user - логін користувача який створював базу даних;
- password - пароль користувача який створював базу даних;

2.4.4 Реалізація контролера та створення свого ендпоінту на сервері

Для зв'язку з сервером не за посиланнями, які налаштовані за замовчуванням потрібно створювати обробники в контролері та власні посилання на дані обробники (ендпоінти). Це надасть змогу створювати власні запити до бази даних, яким потрібно також налаштовувати доступ на сервері. Такий механізм дає змогу гнучкої розробки API для запитів до власної бази даних.

Далі буде продемонстровано один із контролерів, який відповідає за дії пов'язані з таблицею новин:

```
const { createCoreController } = require("@strapi/strapi").factories;

/**
 * news controller
 */

const schema = require("../content-types/news/schema.json");

module.exports = createCoreController("api::news.news", ({ strapi }) => ({
  async getNews(context) {
    try {
      const { id } = context.params;
      const news = await strapi.db.query("api::news.news").findMany({
        where: {
          organisation: { id },
        },
        populate: ["photo", "organisation"],
      });

      context.body = {
        status: 200,
        news,
        schema: schema.attributes,
      };
    } catch (e) {
      context.body = {
        status: 500,
        news: [],
        schema: schema.attributes,
      };
    }
  },
}),
```

```

async deleteByOrganisationId(context) {
  try {
    const { id } = context.params;

    const findNews = await strapi.db.query("api::news.news").findMany({
      where: {
        organisation: { id },
      },
      populate: ["organisation"],
    });

    await strapi.db.query("api::news.news").deleteMany({
      where: {
        id: { $in: findNews.map((news) => news.id) },
      },
    });

    context.body = {
      status: 200,
    };
  } catch (e) {
    context.body = {
      status: 500,
    };
  }
},
}));

```

Тут як ми бачимо створено два обробника кожен з яких обробляє свій запит до бази даних та повертає певну інформацію клієнтській частині. Для того щоб викликати ці обробники потрібно створити відповідні ендпоінти для кожного з них.

Далі буде продемонстровано створення власних ендпоінтів:

```

module.exports = {
  routes: [
    {
      method: "GET",
      path: "/get-news/:id",
      handler: "news.getNews",
      config: {
        auth: false,
      },
    },
    {
      method: "DELETE",
      path: "/delete-news/:id",

```

```

        handler: "news.deleteByOrganisationId",
        config: {
            auth: false,
        },
    },
],
};

```

В даному файлі ми можемо бачити масив об'єктів кожен з яких представляє окремий ендпоінт та містить деякі параметри для його налаштування:

- method – метод який буде використовуватися при виклику даного запиту;
- path – шлях для виклику відповідного запиту;
- handler – обробник в контролері, який відповідає за виконання даного запиту;
- auth - налаштування для авторизації.

Висновки до розділу 2

В процесі проектування веб-додатку «Автоматизація робочого графіку» було сформовано бізнес, функціональні та нефункціональні вимоги, а також вимоги користувачів. На основі сформованих вимог були визначені варіанти використання системи. Була спроектована модульна структура додатку та приблизна структура модулю.

Були спроектовані сутності додатку. На основі сутностей були згенеровані таблиці бази даних та спроектована діаграма бази даних.

Створений основний алгоритм роботи додатку та задокументовано сценарії використання. Описано деякі процеси розробки на серверній та клієнтській частині.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ «АВТОМАТИЗАЦІЯ РОБОЧОГО ГРАФІКУ»

3.1. Порядок встановлення та налаштування параметрів додатку

Додаток «Автоматизація робочого графіку» складається з декількох компонентів, тому її встановлення вимагає розгортання та налаштування кожного з них.

Для зображення компонентів системи та візуалізації розгортання системи була побудована діаграма розгортання:

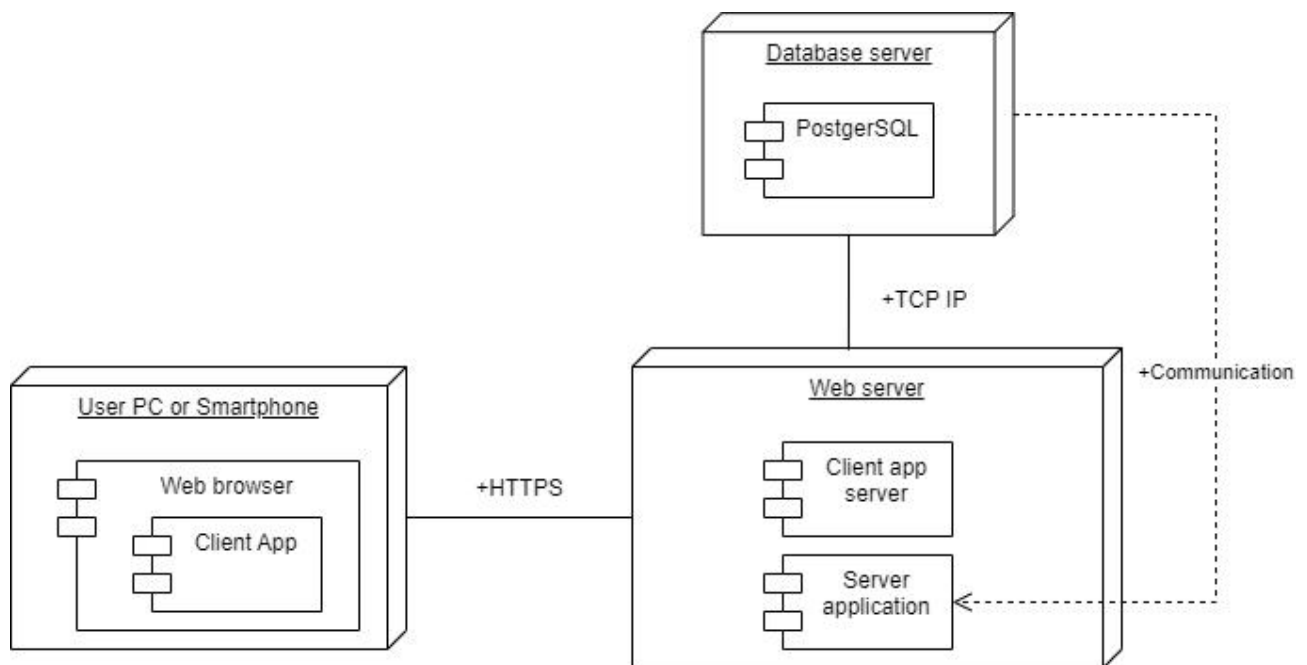


Рисунок 3.1 – Діаграма розгортання

Система складається з 3 вузлів: сервер, сервер бази даних, мобільний пристрій або ПК користувача. Мобільний пристрій або ПК користувача обмінюються даними з веб-сервером за допомогою протоколу HTTPS. Додаток Nginx (або будь який інший зі схожим функціоналом), за допомогою протоколу HTTP, оброблює запити та направляє на відповідні додатки всередині серверу: клієнтський та серверний.

HTTP/HTTPS (Hyper Text Transfer Protocol) протокол передачі даних, базується на системі запитів/відповідей. Протокол підходить стандарту TCP.

Сервер бази даних та веб-сервер обмінюються даними за допомогою протоколу TCP/IP. Цей протокол дозволяє передавати дані без втрат.

Для встановлення додатку «Автоматизація робочого графіку» необхідно розгорнути та налаштувати кожен компонент системи. На сервері бази даних повинен бути встановлений сервер PostgreSQL, створена база даних та налаштовані дані для підключення. Клієнтський додаток вимагає від кожного користувача лише встановлений веб-браузер [8].

На сервері, де буде встановлена система необхідно встановити додаток Node.js версії 16.0 або вище. Пакет для встановлення системи містить файл з інструкцією для встановлення, за якою необхідно заповнити файл налаштувань «.env». В даному файлі знаходяться такі налаштування [3]:

- «HOST» – хост на якому буде запущено сервер;
- «PORT» – порт на якому буде прослуховуватися сервер;
- «APP_KEYS» – ключ серверу для шифрування даних;
- «API_TOKEN_SALT» – використовується для створення API токенів;
- «ADMIN_JWT_SECRET» – секретний ключ використовується для підпису та перевірки;
- «TRANSFER_TOKEN_SALT» – використовується для генерації API токенів передачі даних;
- «DATABASE_CLIENT» – вказує клієнтську базу даних;
- «DATABASE_FILENAME» – шлях та назва бази даних;
- «JWT_SECRET» – секретний ключ для підпису та перевірки JWT;

Після заповнення файлу конфігурації можливо запускати сервер. Сервер повністю налаштується та надасть вам доступ до відповідної бази даних.

3.2. Структура інтерфейсу веб-додатку та порядок роботи

Було спроектовано та розроблено інтерфейс додатку з врахуванням того, що веб-додатком можуть користуватись як з ПК так і з мобільних пристроїв. Кожна сторінка є адаптивною, користувач може користуватись додатком як з мобільного пристрою та і з ПК.

При переході на веб-сторінку додатку, користувач потрапить на головну сторінку (Рис 3.2). Сторінка відкривається і користувач відразу бачить повідомлення про те що необхідно увійти щоб бачити інформацію. В шапці додатку користувач має змогу зареєструватися або увійти в обліковий запис.



Рисунок 3.2 – Головна сторінка

Якщо користувач не має власного профілю, то він може його створити. При створенні профілю користувач має змогу обрати який тип профілю він би хотів створити, а саме звичайного робітника або ж власника компанії.

Прізвище

Ляшук

Ім'я

Макси

По батькові

Володимирович

Ваш день народження

06 . 08 . 2002

Нікнейм

testUser

Електронна пошта

test@gmail.com

Номер телефону

+38 0960217907

Пароль

.....

Повторіть пароль

.....

☒

Встановіть галочку якщо хочете створити аккаунт власника бізнесу

Зареєструватися

Рисунок 3.3 – Реєстрація користувача

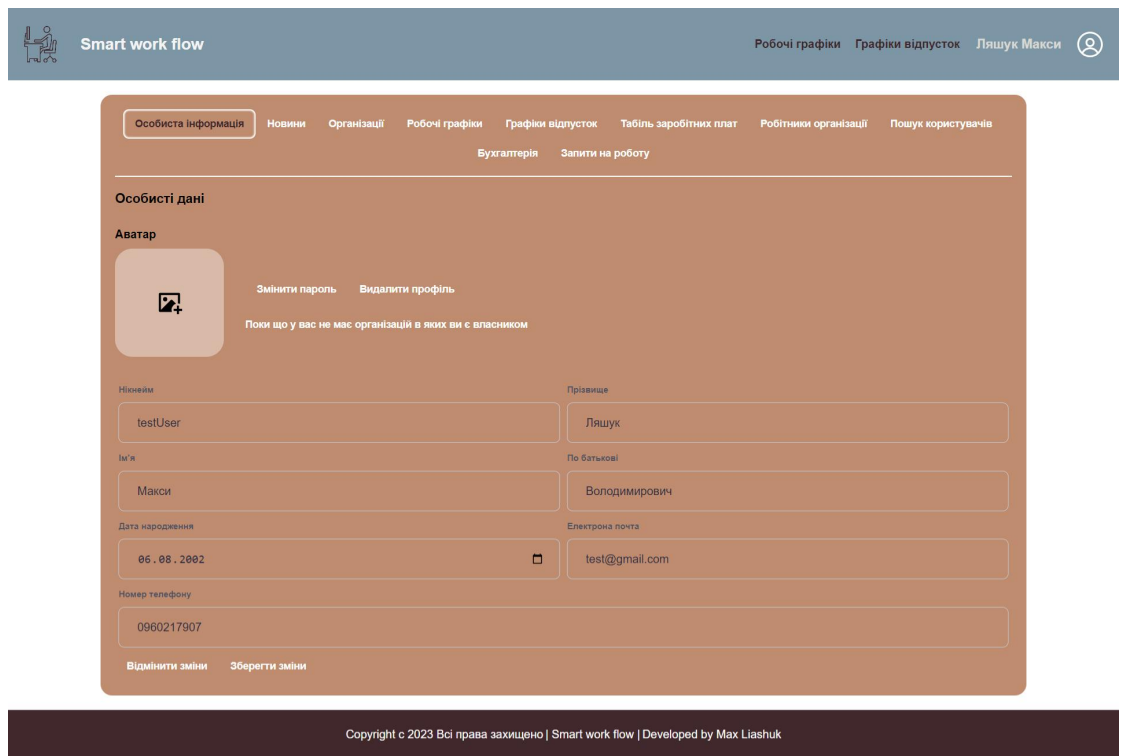


Рисунок 3.4 – Особистий кабінет

В особистому кабінеті будь-який користувач може змінювати власні дані, додавати або видаляти аватар, змінювати пароль або видаляти свій обліковий запис.

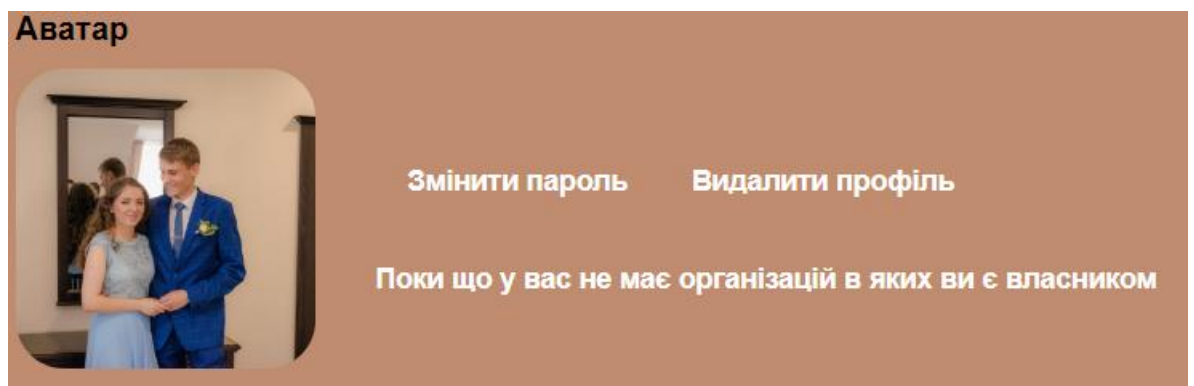


Рисунок 3.5 – Приклад зміни особистих даних, а саме аватар

Додаток складається з двох основних частин: панель адміністратора та основна (користувацька) частина.

3.2.1. Інтерфейс та порядок роботи з панеллю адміністратора

Панель адміністратора прихована від звичайних користувачів. Для того, щоб потрапити туди необхідно увійти в обліковий запис, який матиме роль

власника. Панель адміністратора є частиною особистого кабінету. На наступному рисунку відображено вкладки для адміністратора:

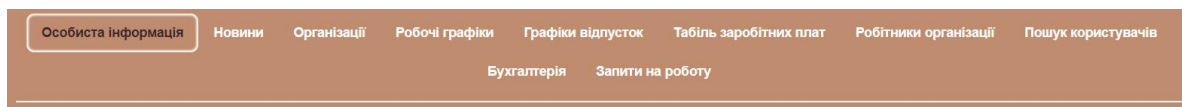


Рисунок 3.6 – Вкладки адміністратора

Власник може створювати одну або більше власних компаній якими він керує, також він може редагувати інформацію компанії та видаляти її. Якщо власник має більше ніж одну компанію, то він має змогу перемикатися між ними та редагувати дані відповідно обраної компанії. Далі буде продемонстровано додавання компанії:

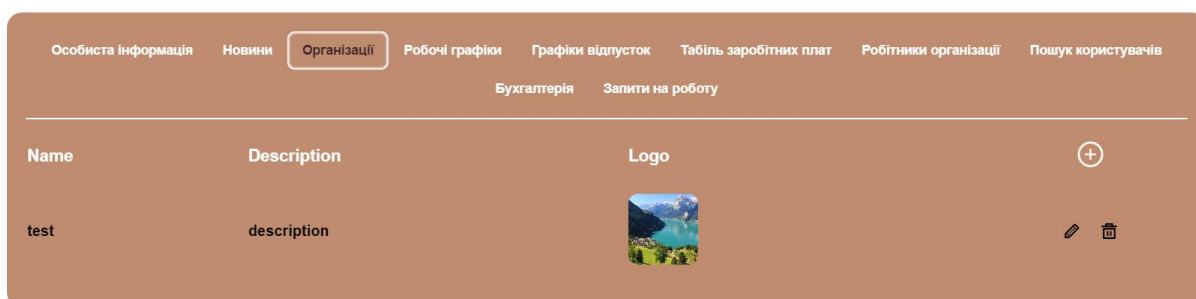


Рисунок 3.7 – Створена нова компанія

Після створення нової компанії власник може виконувати будь-які дії з нею. Також він може додавати до неї ряд допоміжної інформації, а саме: новини, робочі графіки, графіки відпусток, створювати посади та керувати їх заробітними платами та додавати файли з бухгалтерією. Далі буде продемонстровано додавання новини до компанії (все інше є аналогічним):

title

Новина

description

Це скоро відбудеться

date

23 . 06 . 2023

Перетяніть сюди файл або оберіть його з носія

BackOrganisation.jpg

Обрати файл

Створити

Рисунок 3.8 – Створення новини для компанії

Після створення новину можна буде побачити в панелі адміністратора для відповідно обраної організації або на головній сторінці, що буде продемонстровано на наступному рисунку:

Особиста інформація	Новини	Організації	Робочі графіки	Графіки відпусток	Табіль заробітних плат	Робітники організації	Пошук користувачів
		Бухгалтерія	Запити на роботу				
Title	Description	Date	Photo				
Новина	Скоро відкриваємося	2023-06-30					

Рисунок 3.9 – Новини компанії в панелі адміністратора

Також при видалені будь якого запису буде виведено попереджувальне повідомлення:

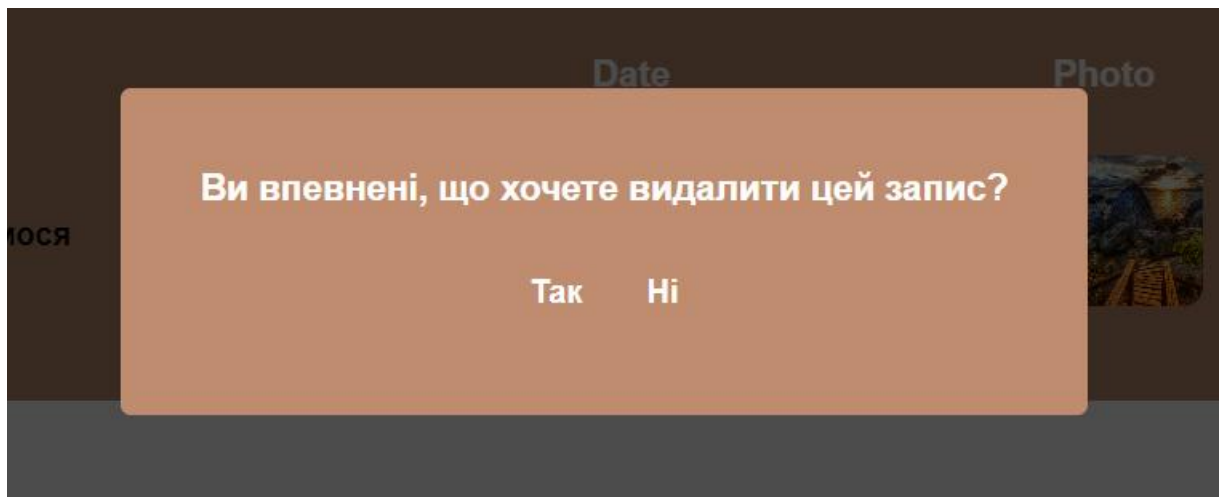


Рисунок 3.10 – Попереджувальне повідомлення при видаленні запису

Також власник може додати робочі графіки та графіки відпусток. Для їх додавання потрібно заповнити відповідну форму, де є обов'язковим файл типу .xls в якому буде відповідно графік. Далі цей файл конвертується та зчитується на клієнтській частині. Після його додавання його можна коригувати або видаляти, як і всі інші дані. Також його можна переглянути в панелі адміністратора або ж подивитися на відповідній сторінці, як це буде бачити звичайний робітник.

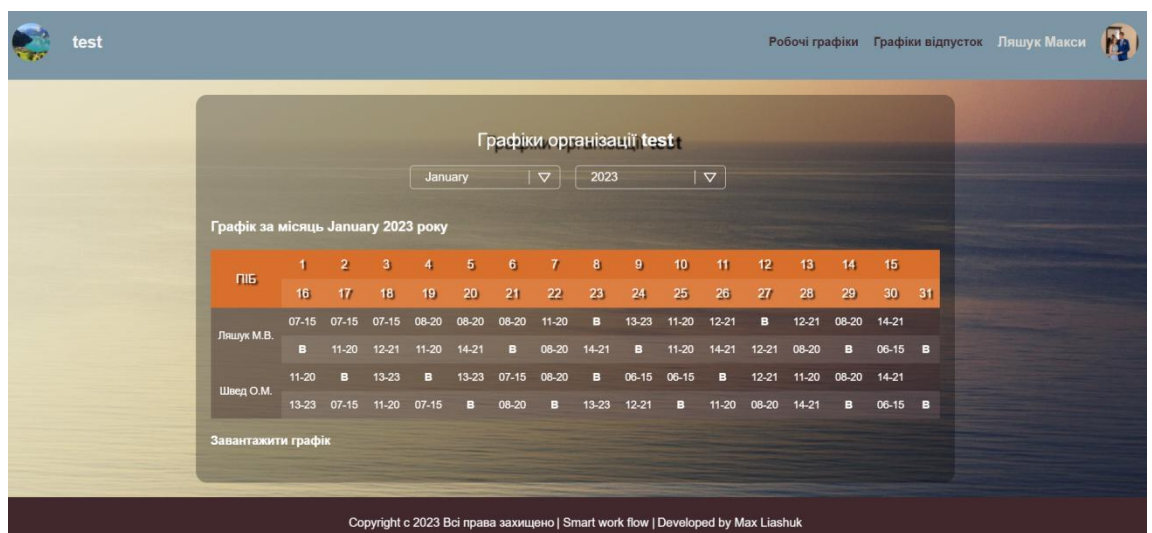


Рисунок 3.11 – Відображення робочого графіку

Як ми бачимо користувач має змогу фільтрувати графіки за місяцем та роком, що надає зручності для їх перегляду. Також відповідний графік можливо завантажити собі на пристрій. Аналогічно відображаються графіки відпусток.

Далі власник може додати посади, які будуть в його компанії та виставити відповідні заробітні плати, що дозволить контролювати облік бухгалтерії. Також посади можуть бути відкрити та закритими, що буде відображатися користувачам при пошуку та перегляду організацій.

Особиста інформація	Новини	Організації	Робочі графіки	Графіки відпусток	Табіль заробітних плат	Робітники організації	Пошук користувачів
		Бухгалтерія	Запити на роботу				
Position	Salary	IsOpen	+				
Власник	1000	×	✎ 🗑				
Менеджер	1000	✓	✎ 🗑				

Рисунок 3.12 – Приклад доданої посади

Також ми бачимо таку посаду, як «Власник», але ця посада створюється автоматично при створенні компанії і користувач який створив відповідну компанію і являється власником. Ця посада за замовчуванням є закритою, тому на неї не можливо подати заяву на працевлаштування.

Далі здійснимо пошук користувачів та візьмемо на роботу одного з знайдених, для цього потрібно відправити відповідний запит на сервер через діалогове вікно:

Оберіть посаду для майбутнього працівника

Менеджер

▼

Ви впевнені, що бажаєте додати цього користувача до організації test ?

Так
Ні

Рисунок 3.13 – Прийняти на роботу користувача

Тепер при пошуку користувачів даний користувач буде відображатися, але власник вже не зможе взяти його на роботу, адже він вже є членом відповідної компанії. Це можна побачити у розділі робітники організації:

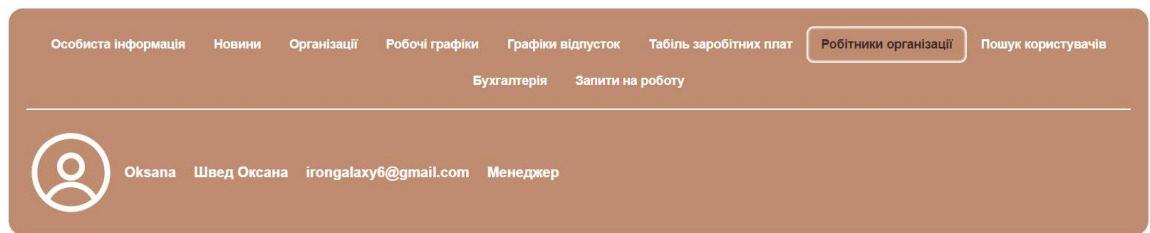


Рисунок 3.14 – Робітники організації

Також можливо перейти на сторінку з інформацією робітника при натисканні на відповідного робітника:



Рисунок 3.15 – Сторінка робітника

Аналогічно до графіків можливо додати файл з бухгалтерією за певний місяць. Після додавання можливо перейти на сторінку де повноцінно буде обрахована бухгалтерія та показано графік прибутку та витрат взятих з файлу:

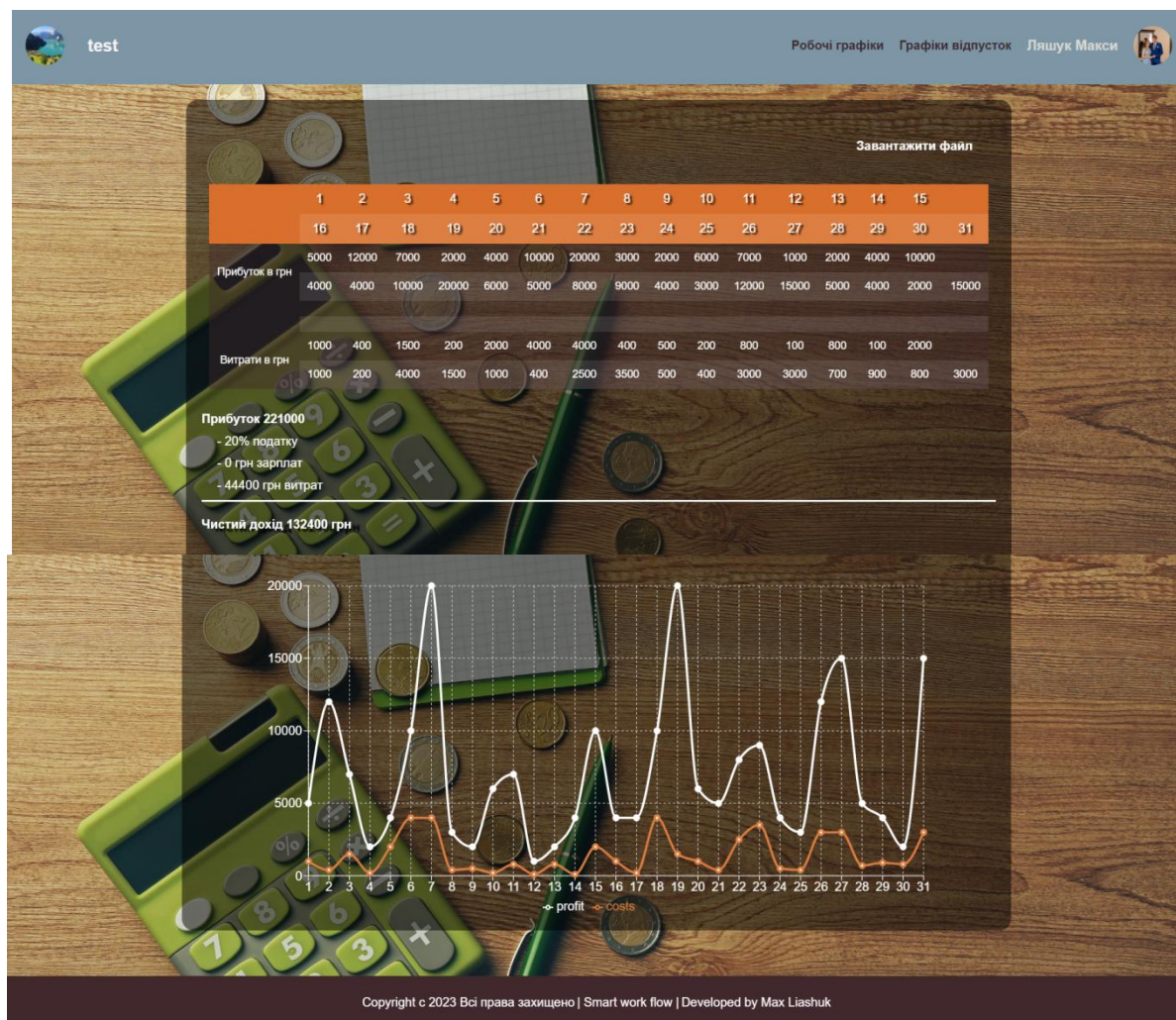


Рисунок 3.16 – Сторінка бухгалтерії

Також в панелі адміністратора наявний пункт запитів на роботу. В цьому пункті відображаються запити, які були подані користувачами на працевлаштування у вашу компанію, якщо були відкриті вакансії. Після надходження таких запитів адміністратор може погодити заяву або відхилити, після цього у користувача який надіслав даний запит оновить його статус та він буде знати відповідь на його заяву. Далі користувач може видалити заяву, якщо вона йому не потрібна. Якщо власник погодив заяву, то користувача буде прийнято на роботу і він це побачить в своєму особистому кабінеті:

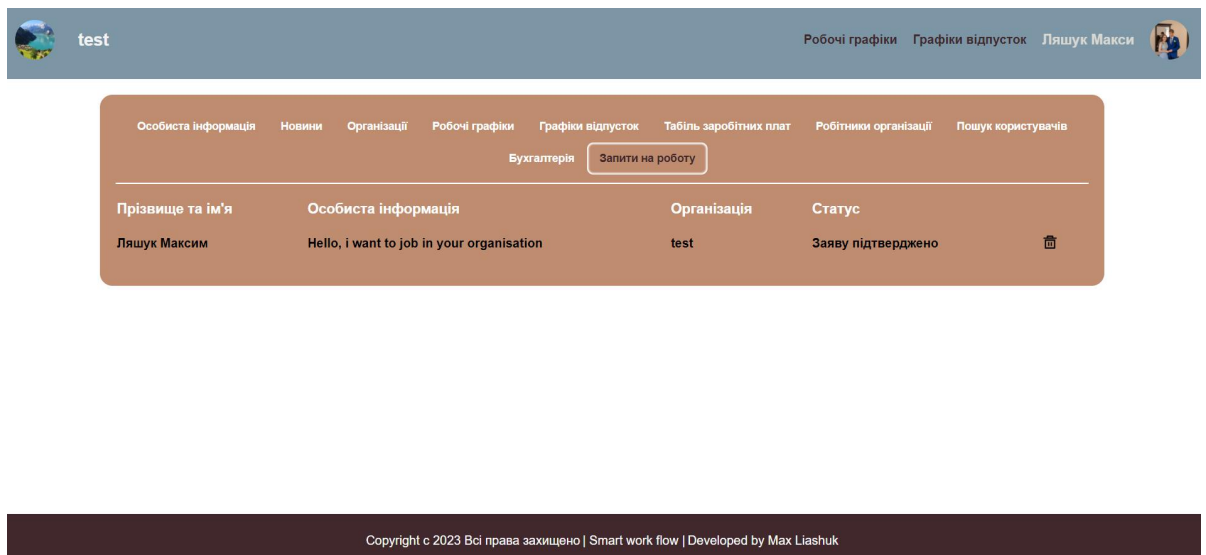


Рисунок 3.17 – Приклад заяви на працевлаштування

При виконанні будь-яких запитів до бази даних користувача повідомляється про успішність або виникнення помилки відповідним повідомленням:

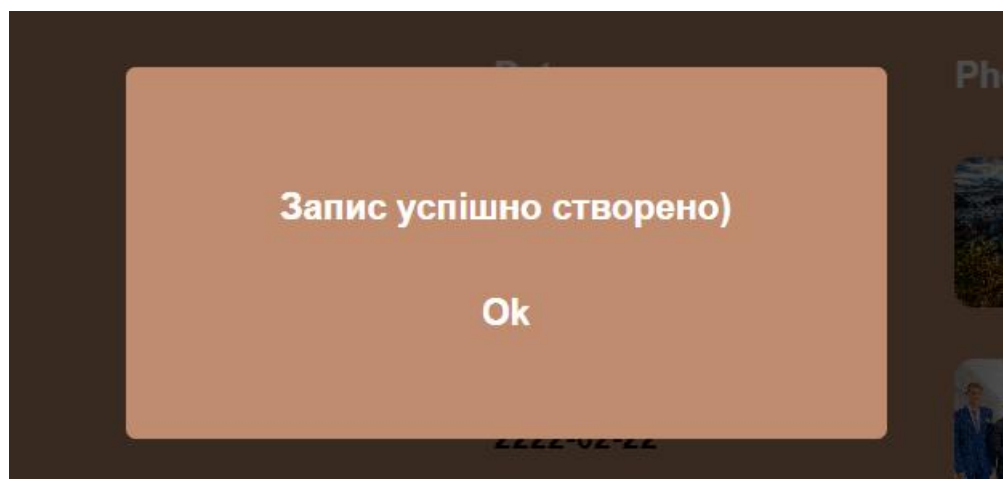


Рисунок 3.18 – Успішне повідомлення на створення запису

3.2.2. Інтерфейс та порядок роботи з основною частиною додатка

Для переходу між сторінкам в додатку використовується навігація, яка розташована в шапці веб-додатку. На даний момент в навігації є посилання на 2 сторінок, а також у випадному списку посилання на особистий кабінет та 2 динамічних посилання для звичайних користувачів та ще 4 динамічних посилання для ролі власника. Кожна сторінка завантажується динамічно та заповнюється на стороні серверу, що максимально пришвидшує їх роботу.

					ІПЗ.КР.Б – 121 – 23 – ПЗ	Арк.
						52

Рисунок 3.19 – Шапка сайта

Робітник на головній сторінці бачить новини обраної компанії в особистому кабінеті, але тільки після того як він влаштується в якусь компанію робітником. Так як користувач може працювати в декількох компаніях, то він може змінювати відображення усієї інформації до відповідно обраної компанії у випадному списку в особистому кабінеті:



Рисунок 3.20 – Головна сторінка з новинами

Користувач може також перейти на сторінку новини, щоб більш детально ознайомитися з нею. Це можливо зробити натиснувши на відповідну новину:

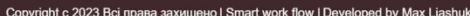


Рисунок 3.21 – Сторінка новини

При авторизації користувача також повідомить якщо він увів не коректні дані для входу:

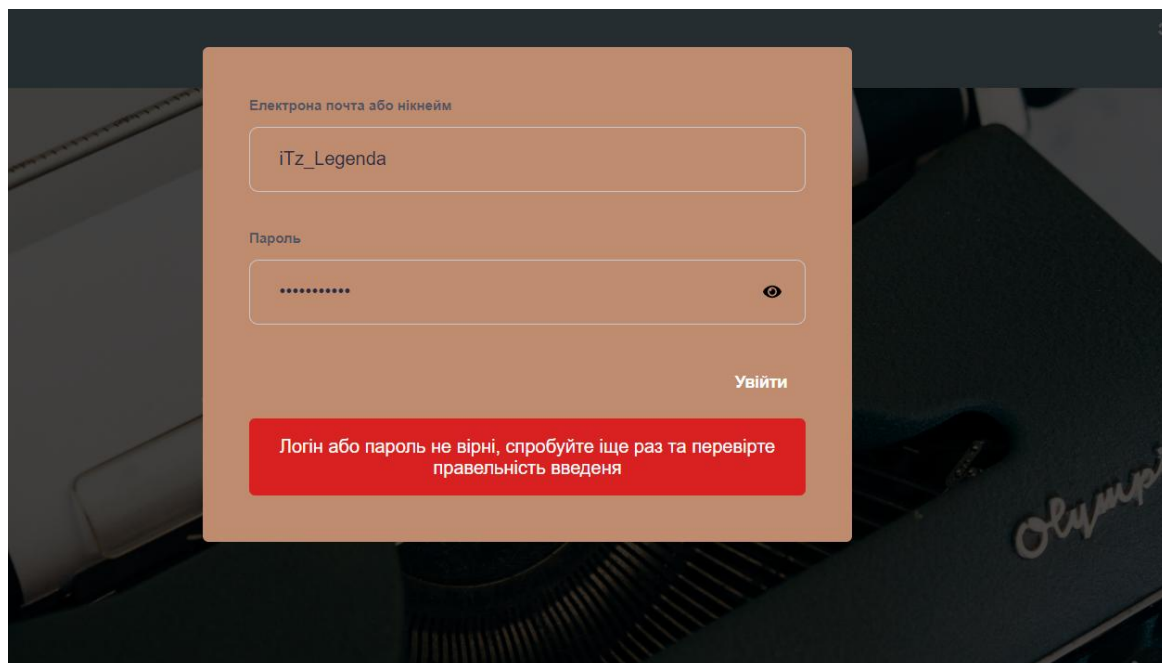


Рисунок 3.22 – Повідомлення про не коректні дані для входу

Далі наведено особистий кабінет користувача:

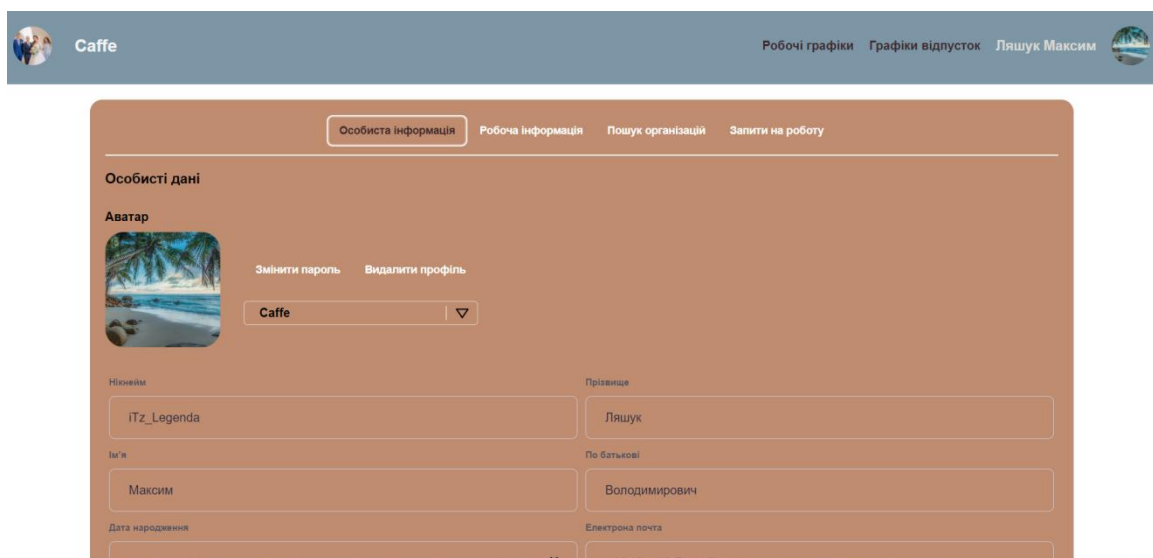


Рисунок 3.23 – Особистий кабінет користувача

Тут користувач може так само змінювати свої особисті дані, додати аватар, змінити його або видалити, також може змінити пароль та видалити свій профіль:

Скріншот форми для зміни паролю. Форма має три поля для введення паролю: "Поточний пароль", "Новий пароль" та "Підтвердіть пароль". Кожне поле має іконку ока для переключення між режимом введення крапками та режимом введення тексту. Нижче полів знаходиться кнопка "Змінити пароль".

Рисунок 3.24 – Форма для зміни паролю

Далі користувач може переглянути інформацію щодо своєї посади в компанії та заробітну плату яку він отримує:



Рисунок 3.25 – Інформація посади та заробітної плати

Також користувач має змогу пошуку організацій та переходу на сторінку організації щоб отримати детальну інформацію та дізнатися чи організація має відкриті вакансії для працевлаштування. Якщо ж користувач працює в даній організації, то йому буде виведено відповідне повідомлення:

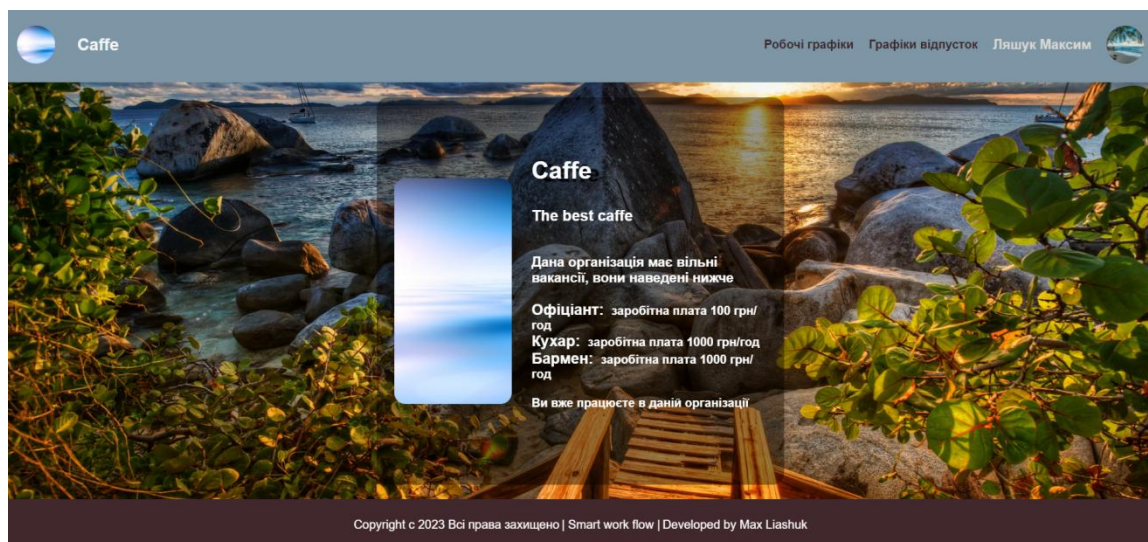


Рисунок 3.26 – Сторінка організації

Якщо ж користувач не працює у відповідній організації, то він може подати заяву на працевлаштування на будь-яку вільну посаду.

Рисунок 3.27 – Форма для надсилання запиту на працевлаштування

Після надіслання заяви вона з'явиться в особистому кабінеті з статусом в обробці. Далі власник компанії як тільки відповість на дану заяву, статус буде оновлено на відхилено або погоджено. Тут також можна бачити усі заяви, які користувач будь-коли подавав та не видалив, тому вони будуть відображатися до поки користувач не вирішить їх видалити:

Прізвище та ім'я	Особиста інформація	Організація	Статус	
Ляшук Максим	Hello, i want to job in your organisation	Caffe	Заяву підтверджено	🗑️
Ляшук Максим	Хотів би працювати у вас барменом. Маю стаж 5 років)	Caffe	Заява в обробці організацією	✎️ 🗑️

Copyright © 2023 Всі права захищено | Smart work flow | Developed by Max Liaschuk

Рисунок 3.28 – Заяви на працевлаштування в особистому кабінеті

Також користувачеві завжди будуть відображатися інформаційні повідомлення, якщо він робить щось не так або не є робітником жодної з організацій.

3.3. Тестування роботи веб-додатку

Тестування допомагає забезпечити високу якість та надійність додатку, шляхом виявлення та усунення помилок, недоліків та потенційних проблем. Тестування також дозволяє перевірити, чи відповідає додаток вимогам та очікуванням користувачів.

Таким чином в ході розробки додатку здійснювалось тестування нового функціоналу під час його впровадження, також було проведене повне тестування функціоналу додатку після завершення розробки. Після виявлення та виправлення помилок роботи додатку проводилось повторне тестування.

В залежності від функціоналу, що тестувався, було використано різноманітні техніки тестування.

Тестування коректної роботи полів форм було протестовано за допомогою техніки «аналіз граничних значень». Наприклад, якщо поле вказано як для номеру телефону, тобто повинно мати 10 цілих чисел та правильного формату, то вводилося не вірний формат або менше ніж 10 цифр. В такому випадку поле підсвічувалось красним та було відповідне повідомлення.

Також були використані техніки «випадкове тестування» та «передбачення помилок». Таким чином був знайдений специфічний дефект клієнтської частини додатку:

передумови: користувач обирає з випадного списку компаній одну активну.

кроки: користувач переходить у особистий кабінет та з випадного списку обирає одну з доступних компаній.

очікуваний результат: компанія стає активною та вся інформація стає відповідна обраній компанії.

фактичний результат: завжди обиралася перша компанія із списку.

					ІПЗ.КР.Б – 121 – 23 – ІЗ	Арк.
						58

Дефект було виправлено і повторно протестовано.

Здійснювалось тестування адаптивності інтерфейсу додатку. Тестування проводилось на мобільному пристрої та на ПК. Перевірялись різні розширення екранів.

При розробці функціоналу були створені тест-кейси для перевірки правильності роботи основного функціоналу додатку.

1. Авторизація:

передумови: користувач вводить логін та пароль.

кроки: користувач натискає кнопку «Увійти» та очікує відповіді.

очікуваний результат: користувач авторизується в системі та заноситься до сесії із своїм унікальним токеном.

фактичний результат: користувач авторизується в системі та заноситься до сесії із своїм унікальним токеном.

2. Зміна паролю користувача:

передумови: користувач переходить в особистий кабінет.

кроки: користувач натискає на кнопку «Змінити пароль» та заповнює необхідні поля форми.

очікуваний результат: пароль успішно змінено та повідомлено користувача про успішність.

фактичний результат: пароль успішно змінено та повідомлено користувача про успішність.

3. Перехід на сторінку організації:

передумови: користувач переходить в особистий кабінет.

кроки: користувач в особистому кабінеті здійснює пошук організації та натискає на одну із знайдених.

очікуваний результат: користувача перенаправляє на сторінку відповідно обраної організації та відображається уся необхідна інформація.

фактичний результат: користувача перенаправляє на сторінку відповідно обраної організації та відображається уся необхідна інформація.

4. Надсилення заяви на працевлаштування:

передумови: користувач в особистому кабінеті знаходить організацію та переходить на її сторінку.

кроки: користувач натискає кнопку «Подати заяву», далі заповнює усі необхідні поля форми та натискає кнопку «Відправити».

очікуваний результат: заяву надіслано, виведено успішне повідомлення та відображену відповідну заяву у особистому кабінеті.

фактичний результат: заяву надіслано, виведено успішне повідомлення та відображену відповідну заяву у особистому кабінеті.

5. Вихід з облікового запису:

передумови: користувач авторизований.

кроки: користувач наводить курсор на випадний список у шапці додатку та натискає кнопку «Вийти».

очікуваний результат: користувач не авторизований, в шапці не відображається інформація користувача та він не має доступу до особистого кабінету.

фактичний результат: користувач не авторизований, в шапці не відображається інформація користувача та він не має доступу до особистого кабінету.

Висновки до розділу 3

В даному розділі був сформований порядок встановлення та налаштування веб-додатку. Наведений порядок роботи з додатком, а саме з панеллю адміністратора та з основною частиною. Розглянуті елементи інтерфейсу додатку та функціонал.

Проведене тестування додатку з метою виявлення дефектів та підвищення впевненості в рівні якості веб-додатку. Виявлені дефекти було виправлено та повторно протестовано.

ВИСНОВКИ

Під час виконання випускної роботи бакалавра було спроектовано та реалізовано систему «Авторизація робочого графіку» у вигляді веб-додатку. Під час розробки та проектування було виконано наступні завдання:

1. Була сформована і проаналізована задача. Проведений аналіз предметної області та порівняльний аналіз наявних аналогічних додатків, виявлені їх переваги та недоліки. Не було знайдено додатків, що виконують усі поставлені задачі. Проаналізовані наявні архітектурні підходи. Обрано монолітну архітектуру для серверу та модульну для клієнту, так як вони є найбільш оптимальними для реалізації поставленої задачі. Визначено середовище виконання NodeJS, фреймворки Strapi та NextJS як основні засоби для реалізації веб-додатку, а також мову програмування JavaScript.

2. Сформовані та проаналізовані функціональні та нефункціональні вимоги до системи, визначені варіанти використання. Визначені компоненти, з яких буде складатися система та зв'язки між ними. Наведені сутності системи, що є представленням даних, які зберігатимуться в системі. Була згенерована база даних на основі визначених сутностей.

3. Сформовано основний алгоритм роботи додатку та алгоритми для роботи окремих частин функціоналу.

4. Спроектований та реалізований веб-додаток «Авторизація робочого графіку». Наведена реалізація окремого компонента додатку для роботи з файлами та їх завантаженням на сервер. Описаний процес створення сторінки клієнтської частини веб-додатку, описано використання компонента.

5. Реалізована система, що задовольняє вимоги до додатку, містить функціонал для автоматизації робочого процесу, зручну роботу з робочими графіками та графіками відпусток, бухгалтерії та взаємодії власника компанії з його робітниками.

6. Інтерфейс додатку є адаптивний і доступний на багатьох розширеннях екранів.

7. Сформований порядок встановлення та налаштування веб-додатку.
Проведене тестування додатку, виявлено та виправлено дефекти.

Веб-додаток можна налаштувати та використовувати для різних компаній.

Реалізований веб-додаток може бути використаний для автоматизації робочого процесу будь-якої компанії, для управління графіками та бухгалтерією, для пошуку користувачами роботи та ефективної співпраці власника з його робітниками.

ЛІТЕРАТУРА

1. Strapi`s documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.strapi.io/>
2. React framework for the Web / NextJS [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/>
3. NodeJS [Електронний ресурс] – Режим доступу до ресурсу:
<https://nodejs.org/en/docs>
4. ReactJS [Електронний ресурс] – Режим доступу до ресурсу:
<https://legacy.reactjs.org/docs/getting-started.html>
5. AXIOS / API [Електронний ресурс] – Режим доступу до ресурсу:
<https://axios-http.com/docs/intro>
6. Redux / Store [Електронний ресурс] – Режим доступу до ресурсу:
<https://redux.js.org/>
7. Redux Thunk [Електронний ресурс] – Режим доступу до ресурсу:
<https://handsonreact.com/docs/redux-thunk>
8. PostgreSQL / DB [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.postgresql.org/docs/>
9. SSR / Next [Електронний ресурс] – Режим доступу до ресурсу:
<https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering>
10. Strapi / Controller [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.strapi.io/dev-docs/backend-customization/controllers>
11. Strapi / Routes [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.strapi.io/dev-docs/backend-customization/routes>
12. Strapi / Configurations [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.strapi.io/dev-docs/configurations>
13. Styled-components [Електронний ресурс] – Режим доступу до ресурсу:
<https://styled-components.com/docs>
14. Strapi / Models [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.strapi.io/dev-docs/backend-customization/models>

15. Strapi / Structure [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.strapi.io/dev-docs/project-structure>
16. Strapi / REST API [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.strapi.io/dev-docs/api/rest>
17. Strapi / Content type-builder [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.strapi.io/user-docs/content-type-builder>
18. NextJS / Routing [Електронний ресурс] – Режим доступу до ресурсу:
<https://nextjs.org/docs/pages/building-your-application/routing/custom-document>
19. QAGroup / Види тестування та відмінності між ними [Електронний ресурс] – Режим доступу до ресурсу:
<https://qagroup.com.ua/publications/vydy-testuvannya-ta-vidminnosti-mizh-nymy/>
20. Reactjs / Virtual DOM and internals [Електронний ресурс] – Режим доступу до ресурсу:
<https://uk.legacy.reactjs.org/docs/faq-internals.html>
21. Reactjs / React.lazy [Електронний ресурс] – Режим доступу до ресурсу:
<https://uk.legacy.reactjs.org/docs/code-splitting.html#reactlazy>
22. Reactjs / Правила хуків [Електронний ресурс] – Режим доступу до ресурсу:
<https://uk.legacy.reactjs.org/docs/hooks-rules.html>
23. NextJS / Lazy Loading [Електронний ресурс] – Режим доступу до ресурсу:
<https://nextjs.org/docs/pages/building-your-application/optimizing/lazy-loading>
24. NextJS/ Image [Електронний ресурс] – Режим доступу до ресурсу:
<https://nextjs.org/docs/pages/api-reference/components/image>
25. NextJS / Head [Електронний ресурс] – Режим доступу до ресурсу:
<https://nextjs.org/docs/pages/api-reference/components/head>
26. SPA [Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/en-us/azure/active-directory/develop/index-spa>

27. Client Server Application [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.bmc.com/docs/ars1805/client-server-application-model-804716436.html>
28. Breakpoint / Media CSS [Електронний ресурс] – Режим доступу до ресурсу: <https://getbootstrap.com/docs/5.0/layout/breakpoints/>
29. Eslint [Електронний ресурс] – Режим доступу до ресурсу: <https://eslint.org/docs/latest/>
30. QA / Техніки тестування [Електронний ресурс] – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/course-materials/glossary-test-design/>

ДОДАТКИ

					ПЗ.КР.Б – 121 – 23 – ПЗ	Арк.
						66

Лістинг convertXLS.js (утиліта для конвертації .xls файлів):

```
import apiRequest from '@api/api';
import * as XLSX from 'xlsx';

const convertXLS = async (url, callback) => {
  try {
    const { data } = await apiRequest.get(
      `${process.env.NEXT_PUBLIC_STRAPI_API_URL}${url}`,
      {
        responseType: 'arraybuffer',
      },
    );
    const fileData = new Uint8Array(data);
    const workbook = XLSX.read(fileData, { type: 'array' });
    const firstSheetName = workbook.SheetNames[0];

    callback(
      XLSX.utils.sheet_to_json(workbook.Sheets[firstSheetName], {
        header: 1,
      }),
    );
  } catch (e) {
    callback([]);
  }
};
```

```
export default convertXLS;
```

Лістинг НОСAuth.js (компонент вищого порядку для слідкування чи авторизований користувач):

```
import { useRouter } from 'next/router';
import { useEffect } from 'react';

const isAuth = (WrappedComponent) => {
  // eslint-disable-next-line func-names
  return function (props) {
    const router = useRouter();

    useEffect(() => {
      if (!localStorage.getItem('token')) {
        router.push('/404');
      }
    }, []);

    return <WrappedComponent {...props} />;
  };
};

export default isAuth;
```

Лістинг create.js (утиліта для створення запису в базі даних):

```
import apiRequest from '@api/api';

const create = async (dataCreate, schema, endPoint, activeOrgId, userId) => {
  try {
    const mediaField = Object.keys(schema).find(
      (key) => schema[key].type === 'media',
    );

    if (endPoint.includes('news')) {
      dataCreate.organisation = activeOrgId;
    }

    if (mediaField) {
      const formData = new FormData();

      formData.append('files', dataCreate[mediaField][0]);
      const uploadedFile = await apiRequest.post(`/api/upload`, formData, {
        headers: {
          'Content-Type': 'multipart/form-data',
        },
      });

      const {
        data: { data },
      } = await apiRequest.post(`${endPoint}`, {
        data: { ...dataCreate, [mediaField]: uploadedFile.data[0].id },
      });

      if (endPoint.includes('organisations')) {
        await apiRequest.post('/api/payrolls', {
          data: {
            position: 'Власник',
            salary: 1000,
            organisation: data.id,
            isOpen: false,
            users_permissions_users: userId,
          },
        });
      }
    } else if (endPoint.includes('payrolls')) {
      await apiRequest.post(`${endPoint}`, {
        data: { ...dataCreate, organisation: activeOrgId },
      });
    }
  }
}
```

```

    } else {
      await apiRequest.post(`${endPoint}`, {
        data: dataCreate,
      });
    }

    return true;
  } catch (e) {
    return false;
  }
};

```

`export default create;`

Лістинг delete.js (утиліта для видалення запису із бази даних):

```

import apiRequest from '@api/api';

const deleteEntity = async (schema, data, endPoint, idDelete) => {
  try {
    const mediaField = Object.keys(schema).find(
      (key) => schema[key].type === 'media',
    );

    if (mediaField) {
      await apiRequest.delete(
        `/api/upload/files/${
          data.find((item) => item.id === idDelete)[mediaField][0].id
        }`,
      );
    }

    if (endPoint.includes('organisation')) {
      const deleteRelationEntity = [
        '/api/delete-accounting',
        '/api/delete-workSchedules',
        '/api/delete-applications',
        '/api/delete-news',
        '/api/delete-payrolls',
        '/api/delete-vacations',
      ];
      const PromisesDelete = deleteRelationEntity.map((entity) =>
        apiRequest.delete(`${entity}/${idDelete}`),
      );

      await Promise.all(PromisesDelete);
    }

    await apiRequest.delete(`${endPoint}/${idDelete}`);
  }
};

```

```

    return true;
  } catch (e) {
    return false;
  }
};

export default deleteEntity;

```

Лістинг update.js (утиліта для оновлення запису в базі даних):

```

import apiRequest from '@api/api';

const update = async (dataUpdate, schema, endPoint, data, idChange) => {
  try {
    const mediaField = Object.keys(schema).find(
      (key) => schema[key].type === 'media',
    );

    if (mediaField) {
      const formData = new FormData();

      await apiRequest.delete(
        `/api/upload/files/${
          data.find((item) => item.id === idChange)[mediaField][0].id
        }`,
      );

      formData.append('files', dataUpdate[mediaField][0]);
      const uploadedFile = await apiRequest.post(`/api/upload`, formData, {
        headers: {
          'Content-Type': 'multipart/form-data',
        },
      });

      await apiRequest.put(`${endPoint}/${idChange}`, {
        data: { ...dataUpdate, [mediaField]: uploadedFile.data[0].id },
      });
    } else {
      await apiRequest.put(`${endPoint}/${idChange}`, { data: dataUpdate });
    }

    return true;
  } catch (e) {
    return false;
  }
};

export default update;

```