ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Цифрова система управління обліком робочого часу співробітників малого бізнесу з використанням мови програмування TypeScript»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Сиволап Едуард Віталійович
(підпис)

Виконав: здобувач вищої освіти групи ТЦР-44
Сиволап Едуард Віталійович

Керівник: Гніденко Ірина Андріївна
ст.викладач

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Технології цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Вікторія ЖЕБКА

« ____ » _____ 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сиволапу Едуарду Віталійовичу

1. Тема кваліфікаційної роботи: «Цифрова система управління обліком робочого часу співробітників малого бізнесу з використанням мови програмування TypeScript», керівник кваліфікаційної роботи старший викладач кафедри технологій цифрового розвитку Гніденко Ірина Андріївна затверджені наказом Державного університету інформаційно-комунікаційних технологій від «___» _____ 2026 р. № ___.
2. Строк подання кваліфікаційної роботи «__» _____ 2026 р.
3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про особливості організації обліку робочого часу співробітників малого бізнесу , принципи побудови рольової моделі та клієнт-серверної

архітектури , документація щодо використання технологій TypeScript, Next.js, React, NestJS, Prisma ORM, PostgreSQL, Redis , а також практичні рекомендації щодо контейнеризації та розгортання за допомогою Docker і Docker Compose.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
    1. Аналіз предметної області та існуючих веб-застосунків для обліку робочих годин співробітників малого бізнесу.
    2. Обґрунтування вибору засобів реалізації клієнтської та серверної частин веб-застосунку.
    3. Проєктування архітектури системи, бази даних, серверних модулів та інтерфейсу користувача.
    4. Програмна реалізація та тестування функціональності веб-застосунку.
5. Перелік графічних матеріалів: *презентація*
    1. Аналіз аналогів.
    2. Вимоги до програмного забезпечення.
    3. Програмні засоби реалізації.
    4. Діаграма ролей користувачів і сценаріїв використання.
    5. Архітектура веб-застосунку.
    6. Структура бази даних.
    7. Екранні форми.
6. Дата видачі завдання «20» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2026	
2	Аналіз та дослідження існуючих аналогів	05.03.-13.03.2026	
3	Огляд та аналіз засобів реалізації веб-застосунку для обліку робочих годин співробітників малого бізнесу	14.03.-16.03.2026	
4	Проєктування архітектури веб-застосунку для обліку робочих годин співробітників малого бізнесу	17.03.-24.03.2026	
5	Програмна реалізація веб-застосунку для обліку робочих годин співробітників малого бізнесу	25.03.-26.04.2026	
6	Тестування функціональності веб-застосунку для обліку робочих годин співробітників малого бізнесу	27.04.-29.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	30.04.-11.05.2026	
8	Розробка демонстраційних матеріалів	12.05.-16.05.2026	
9	Попередній захист роботи	17.05.-30.05.2026	

Здобувач вищої освіти _____Едуард СИВОЛАП
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ГНІДЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня 94 стор., 1 табл., 17 рис., 20 джерел.

Мета роботи - спрощення процесу обліку робочого часу співробітників малого бізнесу за рахунок розробки веб-застосунку.

Об’єкт дослідження - процес обліку робочих годин співробітників малого бізнесу, що охоплює формування графіків роботи, призначення змін, фіксацію відпрацьованого часу та організацію взаємодії між адміністратором і працівниками.

Предмет дослідження - програмне забезпечення для автоматизації обліку робочих годин співробітників малого бізнесу у формі веб-застосунку.

Короткий зміст роботи: у роботі проаналізовано особливості організації обліку робочого часу співробітників малого бізнесу, розглянуто специфіку формування змін і графіків роботи, визначено цільову аудиторію веб-застосунку та досліджено існуючі аналоги, зокрема Clockify, Toggl Track і Jibble. Сформульовано функціональні та нефункціональні вимоги до системи. Обґрунтовано вибір засобів реалізації клієнтської та серверної частин веб-застосунку. Спроєктовано архітектуру системи, структуру бази даних, серверні модулі, клієнтську частину, маршрутизацію, рольову модель користувачів, механізми перевірки доступу та обмеження ролей. Програмну реалізацію виконано з використанням TypeScript, Next.js, React, NestJS, Prisma ORM, PostgreSQL, Redis, Docker і Docker Compose. У системі реалізовано автентифікацію, рольовий доступ, керування користувачами, підрозділами, графіками роботи, робочими змінами, службовими повідомленнями та додатковими інтеграціями. Проведено тестування функціональності веб-застосунку та аналіз результатів тестування.

Сферою використання веб-застосунку є автоматизація внутрішнього обліку робочого часу на підприємствах малого бізнесу, зокрема для адміністрування

персоналу, формування графіків роботи, контролю змін і впорядкування інформації про зайнятість працівників у межах єдиного цифрового середовища.

КЛЮЧОВІ СЛОВА: ОБЛІК РОБОЧОГО ЧАСУ, МАЛИЙ БІЗНЕС, ВЕБ-ЗАСТОСУНОК, ГРАФІК РОБОТИ, РОБОЧІ ЗМІНИ, TYPESCRIPT, NEXT.JS, REACT, NESTJS, POSTGRESQL, PRISMA, REDIS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕННЬ

API - інтерфейс прикладного програмування

CRUD - базові операції створення, читання, оновлення та видалення даних

CORS - механізм контрольованого міждоменного доступу до ресурсів

CSS - каскадні таблиці стилів

HTML - мова розмітки гіпертексту

HTTP - протокол передавання гіпертексту

HTTPS - захищена версія протоколу HTTP

IDE - інтегроване середовище розробки

JSON - текстовий формат обміну структурованими даними

ORM - технологія об’єктно-реляційного відображення

REST - архітектурний підхід до побудови веб-сервісів

SQL - мова структурованих запитів до баз даних

UI - користувацький інтерфейс

URL - уніфікований локатор ресурсу

UX - користувацький досвід взаємодії із системою

WebSocket - протокол двостороннього обміну даними в реальному часі між клієнтом і сервером

ВСТУП

Актуальність: облік робочих годин співробітників малого бізнесу є важливою складовою організації внутрішніх процесів підприємства, оскільки впливає на планування змін, координацію роботи персоналу, контроль фактично відпрацьованого часу та впорядкування адміністративної діяльності. У практиці малого бізнесу такі процеси часто реалізуються за допомогою таблиць, паперових записів, повідомлень у месенджерах або кількох розрізнених цифрових засобів, що ускладнює централізоване зберігання даних, підвищує ймовірність помилок і збільшує витрати часу на формування графіків та перевірку відомостей. Сучасні веб-технології дозволяють створювати веб-застосунки, які забезпечують зручне керування співробітниками, підрозділами, робочими змінами та графіками в межах єдиного інтерфейсу. Веб-застосунок для обліку робочих годин є доцільним рішенням для організації такого процесу, надаючи користувачам можливість працювати з даними через централізовану систему доступу, переглядати актуальну інформацію, вносити зміни до графіків і контролювати робочий час у зручному форматі.

Об'єктом дослідження є процес обліку робочих годин співробітників малого бізнесу, що охоплює формування графіків роботи, призначення змін, фіксацію відпрацьованого часу та організацію взаємодії між адміністратором і працівниками.

Предметом дослідження є програмне забезпечення для автоматизації обліку робочих годин співробітників малого бізнесу у формі веб-застосунку.

Метою роботи є спрощення процесу обліку робочого часу співробітників малого бізнесу за рахунок використання веб-застосунку.

*Методи дослідження***:** першим кроком було проаналізовано особливості організації обліку робочих годин у малому бізнесі та існуючі програмні рішення для керування робочим часом, змінами та графіками з метою формування функціональних і нефункціональних вимог до веб-застосунку. Проведено огляд технологічного стеку, обрано TypeScript для клієнтської та серверної частин,

Next.js і React для побудови користувацького інтерфейсу, NestJS для реалізації серверної логіки, Prisma ORM для взаємодії з базою даних, PostgreSQL як систему зберігання структурованих даних, Redis для підтримки допоміжних серверних механізмів, а також Docker і Docker Compose для розгортання проєкту. Дослідження реалізації проводилося під час розробки клієнтської частини, серверних модулів, бази даних, механізмів автентифікації та функціональних компонентів системи обліку робочого часу.

Практична новизна роботи полягає в розробленні веб-застосунку для обліку робочих годин співробітників малого бізнесу, у якому поєднано керування співробітниками, підрозділами, робочими змінами, графіками роботи, статусами змін і рольовим доступом у межах єдиного програмного рішення. Запропонована система орієнтована на потреби малого бізнесу, де облік робочого часу часто здійснюється фрагментарно, із використанням електронних таблиць, повідомлень або розрізнених цифрових засобів. Новизна прикладного результату полягає в об’єднанні основних операцій адміністрування робочого часу в одному веб-середовищі з централізованим збереженням даних, клієнт-серверною взаємодією та розмежуванням функцій адміністратора і працівника.

Практична значущість результатів полягає в можливості використання реалізованого веб-застосунку для адміністрування персоналу малого бізнесу, формування графіків роботи, контролю робочих змін і впорядкування інформації про зайнятість працівників у межах єдиного цифрового середовища.

1 АНАЛІЗ ВЕБ-ЗАСТОСУНКУ ДЛЯ ОБЛІКУ РОБОЧИХ ГОДИН СПІВРОБІТНИКІВ МАЛОГО БІЗНЕСУ

1.1 Предметна область обліку робочого часу співробітників малого бізнесу

Веб-застосунок для обліку робочих годин співробітників малого бізнесу - це спеціалізований онлайн-ресурс, який надає можливість централізовано зберігати, переглядати та опрацьовувати дані про працівників, підрозділи, графіки роботи, робочі зміни та фактично відпрацьований час. Такі веб-застосунки допомагають користувачам, зокрема адміністраторам, керівникам невеликих команд і співробітникам, організовувати робочий процес через єдиний інтерфейс, у якому поєднано засоби планування, контролю та перегляду інформації. Основна мета веб-застосунку полягає у забезпеченні впорядкованого та доступного способу обліку робочого часу, що дозволяє користувачам ефективно координувати змінну зайнятість персоналу незалежно від кількості працівників або підрозділів.

Основні компоненти веб-застосунку для обліку робочих годин включають:

- Інформаційну панель: сторінки для перегляду основних відомостей про стан системи, кількість працівників, зміни та інші зведені показники;

- Керування співробітниками: засоби додавання, редагування та перегляду даних про працівників, їхню роль у системі та належність до підрозділів;

- Керування підрозділами: інструменти створення та адміністрування структурних одиниць підприємства, для яких ведеться окремий облік робочого часу;

- Графіки роботи: механізми формування, перегляду та коригування розкладу роботи працівників на визначений період;

- Робочі зміни: засоби призначення, фіксації та контролю змін із вказанням часу початку, завершення, тривалості та статусу;

- Аутентифікацію та профіль: можливості входу до системи, розмежування доступу залежно від ролі та роботи з персональними даними користувача;

- Сповіщення та службові стани: елементи, що відображають зміни в системі, оновлення даних та результати опрацювання робочих змін.

Основні цілі веб-застосунку для обліку робочих годин включають:

- Забезпечення централізованого доступу до актуальних даних про працівників, зміни та графіки роботи;

- Спрощення процесу планування та контролю робочого часу;

- Підтримку взаємодії між адміністратором і працівниками в межах єдиної системи;

- Зменшення кількості ручних операцій під час ведення обліку;

- Підвищення зручності роботи з розкладом і змінами через інтуїтивний інтерфейс;

- Забезпечення структурованого зберігання даних про зайнятість персоналу;

- Підтримку адаптивної роботи системи на різних пристроях.

Оцінимо переваги та недоліки веб-застосунку як засобу для обліку робочих годин співробітників малого бізнесу.

Переваги:

- Доступність: веб-застосунок дозволяє працювати з даними з різних пристроїв за наявності мережевого доступу, що спрощує взаємодію з системою;

- Упорядкованість даних: інформація про працівників, підрозділи, графіки та зміни зберігається в межах єдиного інформаційного середовища;

- Гнучкість керування: адміністратор може швидко змінювати графік, редагувати зміни та контролювати статуси записів;

- Розмежування доступу: використання ролей дозволяє обмежувати функції відповідно до типу користувача;

- Економія часу: автоматизація типових операцій зменшує навантаження на адміністративний персонал і прискорює щоденну роботу.

Недоліки:

- Технічна залежність: робота системи потребує стабільного доступу до мережі та коректного функціонування серверної частини;

- Необхідність первинного налаштування: перед повноцінним використанням потрібно внести дані про працівників, підрозділи та базові параметри роботи;

- Чутливість до помилок введення: некоректно внесені записи можуть впливати на відображення графіків і змін;

- Потреба в адмініструванні: система вимагає контролю доступу, оновлення даних та підтримки актуального стану облікової інформації.

1.2 Специфіка організації обліку робочих годин, змін і графіків роботи

Організація обліку робочих годин, змін і графіків роботи є складним процесом, що відрізняється від загального кадрового обліку своєю прив’язаністю до часових інтервалів, оперативних змін у розкладі та необхідності постійного оновлення даних. У малому бізнесі робочий процес часто залежить від змінної кількості персоналу, режиму роботи підрозділів, внутрішніх замін працівників і нерівномірного навантаження протягом тижня. Через це система обліку повинна забезпечувати точне відображення дати зміни, часу її початку і завершення, загальної тривалості роботи, статусу опрацювання та належності запису до конкретного співробітника і підрозділу. Дані про графіки роботи та фактично відпрацьований час мають структурований характер, однак потребують постійної актуалізації, оскільки навіть незначне коригування однієї зміни впливає на загальний стан розкладу.

Організаційні труднощі пов’язані з тим, що в малому бізнесі розподіл робочого часу нерідко виконується без окремої кадрової або диспетчерської ланки. Адміністратор або керівник одночасно формує графік, вносить зміни до нього, переглядає персонал, контролює статуси записів і узгоджує фактичну зайнятість працівників. За відсутності єдиного цифрового середовища це призводить до дублювання інформації, втрати актуальних відомостей і розриву

між запланованими та фактичними змінами. Користувачі системи потребують інтерфейсу, у якому дані про співробітників, підрозділи, розклад і робочі зміни відображаються в узгодженому вигляді, а доступ до них визначається роллю користувача.

Функціональні особливості веб-застосунку для обліку робочих годин зосереджені на забезпеченні впорядкованої роботи з даними через:

- відображення інформації про співробітників, підрозділи, графіки та робочі зміни в межах єдиної системи;

- призначення змін із фіксацією дати, часу початку, часу завершення та тривалості роботи;

- формування та коригування графіків роботи для окремих працівників або підрозділів;

- використання статусів змін для відображення стану їх опрацювання;

- розмежування доступу між адміністратором і працівником відповідно до ролі в системі;

- перегляд персональних і загальних відомостей про зайнятість через веб-інтерфейс.

Система працює з чітко структурованими організаційними даними, що охоплюють відомості про користувачів, підрозділи, календарні періоди, робочі зміни, статуси записів і службові повідомлення. Така модель забезпечує узгоджене збереження інформації та дає можливість швидко отримувати потрібні відомості без звернення до кількох незалежних джерел. Використання централізованої бази даних дозволяє підтримувати цілісність записів і зменшує ризик розбіжностей між графіком, змінами та обліковими даними працівників.

Технічні можливості застосунку включають:

- збереження даних про співробітників, підрозділи, графіки та зміни в єдиній серверній системі;

- обробку клієнтських запитів до серверної частини через програмні модулі, що відповідають за користувачів, підрозділи, графік роботи та робочі зміни;

- використання механізмів автентифікації та сесійної взаємодії для контролю доступу до функцій системи;

- відображення клієнтського інтерфейсу з окремими сторінками для інформаційної панелі, співробітників, підрозділів, графіків, профілю та налаштувань;

- підтримку статусної логіки змін і сповіщень, пов’язаних із подіями системи;

- розгортання застосунку як окремих сервісів клієнтської частини, серверної частини, бази даних і допоміжних компонентів.

Поточна модель організації обліку робочих годин, змін і графіків роботи у веб-застосунку забезпечує впорядковане керування персоналом і часовими записами в межах малого бізнесу. Завдяки поєднанню клієнтської та серверної частин, централізованого збереження даних, рольового доступу та структурованої роботи зі змінами система надає зручний інструмент для щоденного адміністрування робочого часу. Обмеження, пов’язані з потребою постійного оновлення записів і точного введення даних, компенсуються цілісністю інформаційної моделі, зручністю взаємодії та можливістю швидкого коригування графіків роботи.

1.3 Цільова аудиторія веб-застосунку

Веб-застосунок для обліку робочих годин співробітників малого бізнесу орієнтований на кілька категорій користувачів, чия діяльність пов’язана з плануванням робочого часу, організацією змін і контролем фактично виконаної зайнятості. До цільової аудиторії належать як особи, що здійснюють адміністрування персоналу та розкладу, так і працівники, які використовують систему для перегляду власних змін, графіків і пов’язаних із ними даних. Застосунок формує єдине цифрове середовище для роботи з підрозділами, користувачами, статусами змін, профілем і службовими повідомленнями, що забезпечує узгоджену взаємодію між різними ролями в межах малого бізнесу.

Центральне місце в цільовій аудиторії посідають власники малого бізнесу, адміністратори та керівники, які відповідають за організацію роботи персоналу. Переважно це користувачі віком від 25 до 50 років, діяльність яких поєднує управлінські функції з безпосереднім контролем операційних процесів. Для них система є інструментом щоденного адміністрування, оскільки саме вони працюють зі структурою підрозділів, списком співробітників, призначенням і коригуванням змін, переглядом графіків та контролем узгодженості записів. Потреба цієї категорії полягає у швидкому доступі до повного обсягу інформації, чіткому розмежуванні прав доступу та можливості вносити зміни без зайвих проміжних операцій. Централізоване зберігання даних у межах єдиної системи зменшує організаційне навантаження, яке виникає за використання кількох таблиць або сторонніх сервісів.

До складу цільової аудиторії належать також менеджери змін, старші адміністратори або інші відповідальні особи в межах підрозділів, якщо структура малого бізнесу передбачає частковий розподіл управлінських функцій. Переважно це користувачі віком від 23 до 45 років, які не завжди мають повний адміністративний доступ, але залучені до практичної координації графіка роботи. Вони працюють із розкладом у межах окремого підрозділу, контролюють заповнення змін, уточнюють зайнятість працівників і відстежують стан створених записів. Для цієї категорії визначальне значення має наочність інтерфейсу, швидкість переходу між сторінками співробітників, підрозділів і графіків, а також зрозуміле подання календарної логіки та статусів змін. За відсутності такої структурованості зростає ризик помилок під час щоденної координації розкладу.

До цільової аудиторії входять і працівники підприємства, які використовують веб-застосунок для перегляду власного графіка, змін і персональних даних. У більшості випадків це користувачі віком від 18 до 45 років, взаємодія яких із системою є простішою порівняно з адміністративним рівнем, проте потребує постійного доступу до актуальної інформації про власну зайнятість. Для них визначальною є можливість швидко відкрити профіль,

переглянути призначені зміни, перевірити час початку і завершення роботи, орієнтуватися в статусі записів та своєчасно реагувати на зміни графіка. Якщо інтерфейс перевантажений функціями, які не пов’язані з їхньою щоденною взаємодією, це знижує зручність користування і ускладнює доступ до необхідних даних. Саме тому рольова модель системи має значення не лише для безпеки, а й для функціональної придатності інтерфейсу до потреб конкретної категорії користувачів.

Окремо слід враховувати підприємства, у яких облік робочого часу здійснюється для кількох локацій або структурних одиниць. За таких умов веб-застосунок виконує функцію не лише індивідуального перегляду змін, а й засобу координації між кількома підрозділами. Для таких користувачів визначальними є механізми прив’язки працівників до конкретних структурних одиниць, збереження окремих графіків та узгодженого ведення часових записів у межах єдиної системи. Ця частина аудиторії особливо чутлива до точності адміністрування та структурованості даних, оскільки будь-яка неузгодженість між підрозділом, зміною та профілем працівника безпосередньо впливає на організацію праці.

Функціональні можливості веб-застосунку відповідають потребам усіх зазначених категорій користувачів. Власники й адміністратори отримують інструмент для керування користувачами, підрозділами, графіками та змінами в межах єдиної інформаційної системи. Менеджери змін або відповідальні особи працюють із поточним розкладом, контролем записів і координацією діяльності персоналу в конкретному організаційному сегменті. Працівники використовують систему для перегляду власної зайнятості, відомостей про зміну та персонального профілю. Поєднання рольового доступу, централізованої серверної обробки, клієнтського веб-інтерфейсу та структурованої бази даних формує спільне робоче середовище, придатне для різних категорій користувачів у межах малого бізнесу.

1.4 Дослідження існуючих аналогів

Для визначення вимог до веб-застосунку доцільно проаналізувати програмні рішення, які вже використовуються для обліку робочого часу, змін і часових записів працівників. Порівняння існуючих аналогів дозволяє встановити, які функціональні можливості є стандартними для цього класу систем, які підходи застосовуються для організації інтерфейсу, які обмеження мають поширені сервіси та які особливості можуть бути враховані під час розробки власного рішення. Для аналізу обрано Clockify, Toggl Track і Jibble, оскільки ці сервіси належать до відомих веб-платформ для обліку часу, табелювання, звітності та організації командної роботи.

1.4.1 Clockify

Clockify - це веб-сервіс для обліку робочого часу, який дає змогу фіксувати години роботи за допомогою таймера, табеля або режиму kiosk, а також підтримує звітність, attendance, planning і payroll. Платформа позиціонується як time tracker для команд і проєктів та доступна через вебінтерфейс, настільні застосунки, мобільні застосунки й браузерні розширення. За своїм функціональним призначенням Clockify орієнтований на команди, яким потрібно вести облік часу, отримувати звіти та працювати з часовими записами в різних робочих середовищах.

Основні функції Clockify:

- фіксація часу за допомогою таймера або ручного внесення записів у timesheet;
- використання kiosk для clock-in і clock-out через спільний пристрій;
- формування звітів щодо витраченого часу, attendance, overtime і breaks;
- планування ресурсів, capacity та utilization;
- робота з time off, approval і request-механізмами;
- експорт та використання даних для payroll-операцій.

Ключові переваги Clockify:

- підтримка кількох способів обліку часу, що охоплюють timer, timesheet і kiosk;

- наявність засобів attendance, reporting, planning і payroll в межах однієї системи;

- доступність на вебплатформі, мобільних пристроях і настільних операційних системах;

- придатність для командної роботи завдяки механізмам approval і request;

- можливість використання спільного пристрою для onsite-обліку робочого часу.

Обмеження сервісу:

- система значною мірою орієнтується на універсальний time tracking для проєктів і команд, а не на вузьку модель внутрішнього адміністрування малого бізнесу;

- частина функцій, пов’язаних із погодженням і розширеним керуванням записами, доступна лише в платних планах;

- багатофункціональність сервісу ускладнює інтерфейс для користувачів, яким потрібен лише облік змін, графіків і ролей;

- модель роботи побудована навколо часових записів і робочих просторів, тому вона не повністю відповідає структурі системи, де центральними сутностями є підрозділи, статуси змін і внутрішня рольова взаємодія.

Технічні особливості:

- доступ через браузер, настільні застосунки для Windows, macOS і Linux, мобільні застосунки та браузерні розширення;

- підтримка shared-device kiosk для onsite-режиму;

- серверна обробка часових записів і формування звітів у централізованому середовищі;

- використання моделі workspace із командами, проєктами, часовими записами та заявками.

Clockify є функціонально насиченим рішенням для time tracking, attendance і payroll-oriented обліку, проте його архітектура більше орієнтована на

універсальне відстеження часу в межах робочих просторів, ніж на спеціалізовану модель керування підрозділами, змінами та графіками малого бізнесу. Саме тому під час розробки власного веб-застосунку доцільно врахувати сильні сторони Clockify, пов'язані з фіксацією часу та звітністю, але спростити структуру інтерфейсу й зосередити її на внутрішніх процесах підприємства. Загальний вигляд інтерфейсу сервісу Clockify наведено на рис. 1.1.

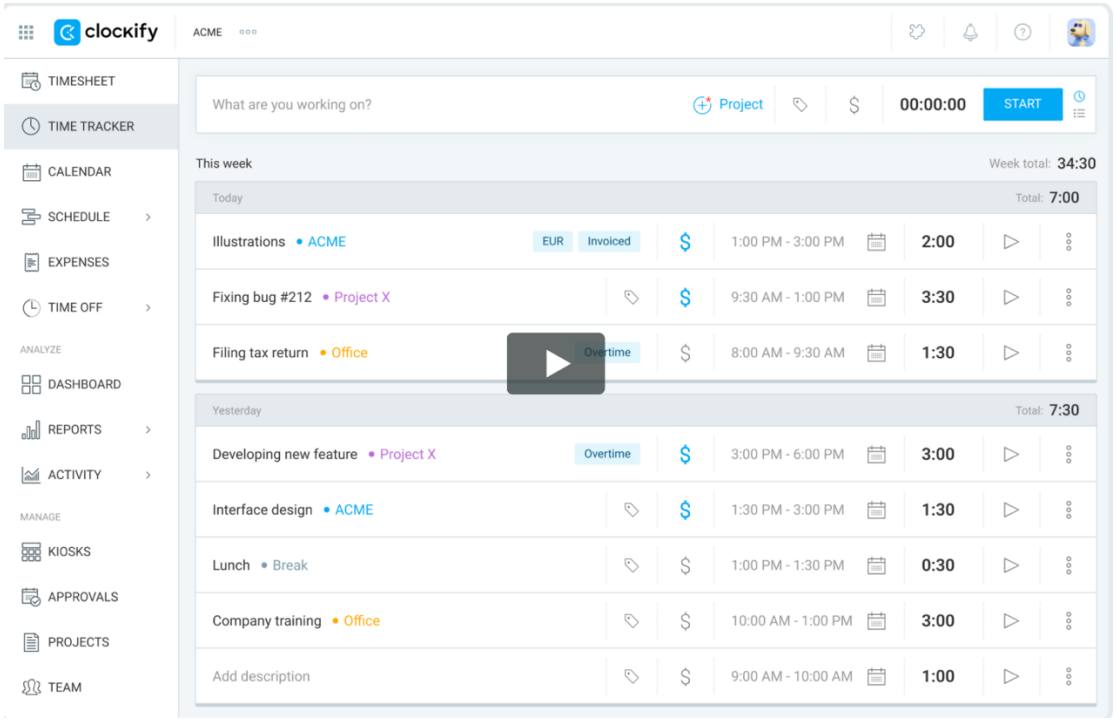

Рис. 1.1. Інтерфейс Clockify

1.4.2 Toggl Track

Toggl Track - це веб-сервіс для time tracking, який поєднує облік часу, timesheet-записи, звітність, automated tracking, calendar view, integrations і time off management. Платформа орієнтована на команди, яким потрібен контроль витраченого часу, звіти щодо продуктивності та взаємодія з іншими бізнес-інструментами. У структурі сервісу значне місце займають проєкти, клієнтські задачі, billable/non-billable hours, а також аналітичні засоби для оцінювання прибутковості й навантаження.

Основні функції Toggl Track:

- облік часу через online timer і автоматизоване відстеження активності;

- ведення timesheets і перетворення часових записів на звітні дані;

- побудова ready-made reports і customized time reporting;

- calendar view із синхронізацією з Google Calendar та Outlook;

- time off management із leave requests, approvals і team schedule visibility;

- integrations із зовнішніми сервісами та робота з billable hours.

Ключові переваги Toggl Track:

- поєднання класичного time tracking із розширеною звітністю та аналітикою;

- інтеграція календаря, що спрощує узгодження часових записів із подіями;

- наявність time off management для відображення доступності команди;

- розвинена система integrations, придатна для включення сервісу в ширше цифрове середовище;

- придатність для користувачів, які працюють із задачами, проєктами та звітами щодо ефективності.

Обмеження Toggl Track:

- сервіс орієнтований переважно на time tracking у контексті проєктної та клієнтської роботи, а не на внутрішнє адміністрування змін і підрозділів малого бізнесу;

- частина функцій, пов’язаних із approvals, reminders і time off workflows, прив’язана до платних тарифів;

- у фокусі системи перебувають часові записи, продуктивність і billable logic, а не сценарії організації змінного графіка персоналу;

- структура сервісу менш придатна для випадків, де необхідно централізовано керувати працівниками, відділами, статусами змін і внутрішніми службовими записами в межах одного підприємства.

Технічні особливості:

- веб-платформа, настільні застосунки та синхронізація з календарними сервісами;

- automated tracking і timeline-based presentation часових записів;

- засоби time reporting, integrations і calendar-linked workload visibility;

- модульна побудова функцій, де значна частина логіки пов’язана з проєктами, задачами та обліком часу для звітності.

Toggl Track є зручним інструментом для фіксації часу, формування звітів і координації робочої активності, проте його модель більше відповідає проєктному або клієнтському time tracking, ніж внутрішньому табелюванню змінної зайнятості працівників малого бізнесу. Під час проєктування власного веб-застосунку доцільно врахувати позитивні сторони Toggl Track, пов’язані з reporting, calendar view і прозорістю часових записів, але адаптувати функціональність до предметної області, у якій базовими сутностями є підрозділи, графіки, статуси змін і ролі користувачів. Приклад інтерфейсу сервісу Toggl Track наведено на рис. 1.2.

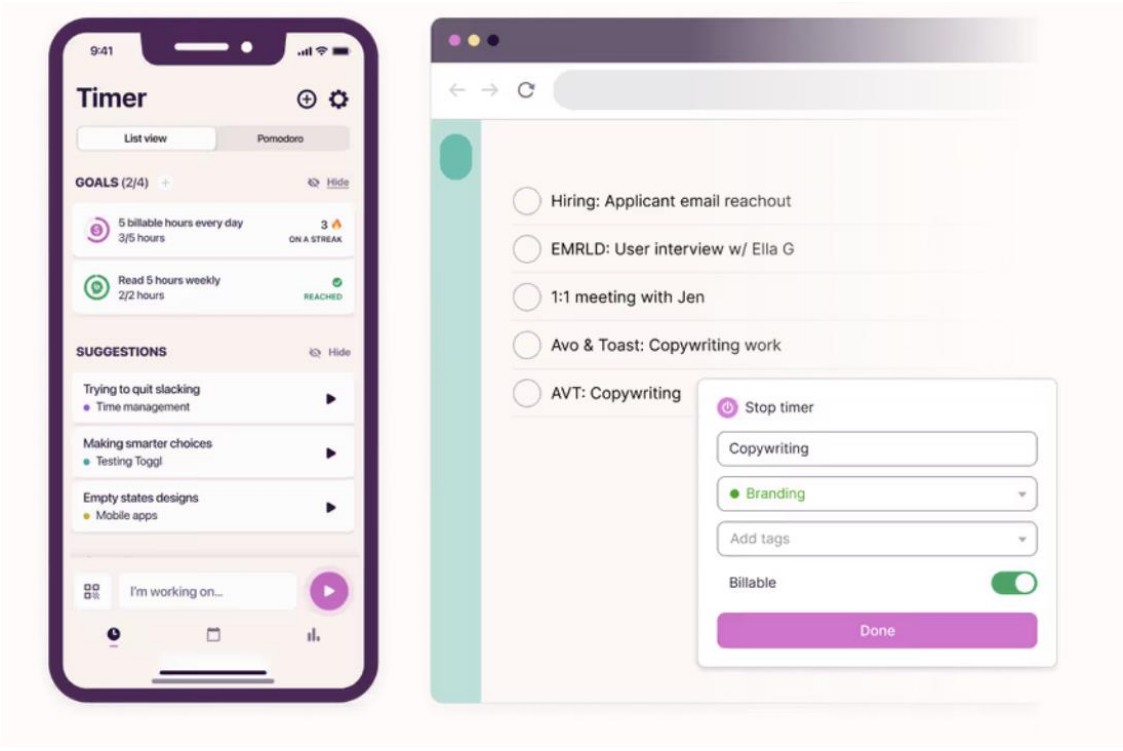

Рис. 1.2. Інтерфейс Toggl Track

1.4.3 Jibble

Jibble - це веб-сервіс для time tracking, time attendance, scheduling і автоматизованого формування timesheets, який позиціонується як безкоштовне рішення для необмеженої кількості користувачів. Платформа поєднує clock-in/clock-out, attendance control, automated timesheets, work schedules, payroll-ready records і додаткові механізми контролю, пов’язані з географічними межами, kiosk-режимом та варіантами перевірки присутності. Jibble орієнтований на команди, яким потрібен оперативний облік часу та відвідуваності без складного початкового впровадження.

Основні функції Jibble:

- clock-in і clock-out із автоматичним формуванням timesheets;
- time attendance records із відображенням regular hours, overtime і breaks;
- employee scheduling і робота з work schedule settings;
- payroll-ready timesheets, придатні для експорту;
- автоматизований або location-aware clock in/out з використанням geofencing;
- підтримка командного керування й централізованого перегляду даних про присутність.

Ключові переваги Jibble:

- орієнтація саме на attendance, shifts, timesheets і payroll-ready records;
- наявність scheduling-модуля для призначення змін і уникнення конфліктів у розкладі;
- автоматичне перетворення clock-in/clock-out даних на табелі;
- придатність для малого бізнесу завдяки простішій логіці впровадження;
- безкоштовна модель для необмеженої кількості користувачів, заявлена сервісом.

Обмеження Jibble:

- сервіс орієнтується насамперед на attendance і time clock сценарії, тому його внутрішня логіка не повністю збігається з моделлю системи, у якій

поєднуються підрозділи, профілі, статуси змін, графіки й окремі адміністративні модулі;

- частина функціональності побудована навколо загального обліку присутності та payroll-preparation, а не навколо індивідуалізованої внутрішньої структури конкретного малого підприємства;

- при використанні зовнішньої платформи підприємство залежить від її готової моделі сутностей, інтерфейсу та правил конфігурації;

- у межах власної розробки можна точніше адаптувати функції до структури підрозділів, ролей і сценаріїв роботи конкретної організації.

Технічні особливості:

- автоматичне формування timesheets із clock-in/clock-out подій;

- підтримка employee scheduling і work schedules для розрахунку overtime, deductions і payroll hours;

- використання geofencing для обмеження clock in/out authorized locations;

- наявність настільних і вебзасобів для роботи з часовими записами.

Jibble є одним із найближчих аналогів до системи обліку робочих годин, оскільки поєднує attendance, shifts, schedules і timesheets у межах одного сервісу. Проте навіть за наявності придатних функцій зовнішній сервіс не дає повної свободи у формуванні власної предметної моделі, де структура даних, ролі, підрозділи, сценарії доступу та логіка відображення інтерфейсу підлаштовуються під конкретну організацію. Саме це створює підстави для розробки власного веб-застосунку, орієнтованого на внутрішні процеси малого бізнесу. Приклад візуальної організації інтерфейсу сервісу Jibble наведено на рис. 1.3.

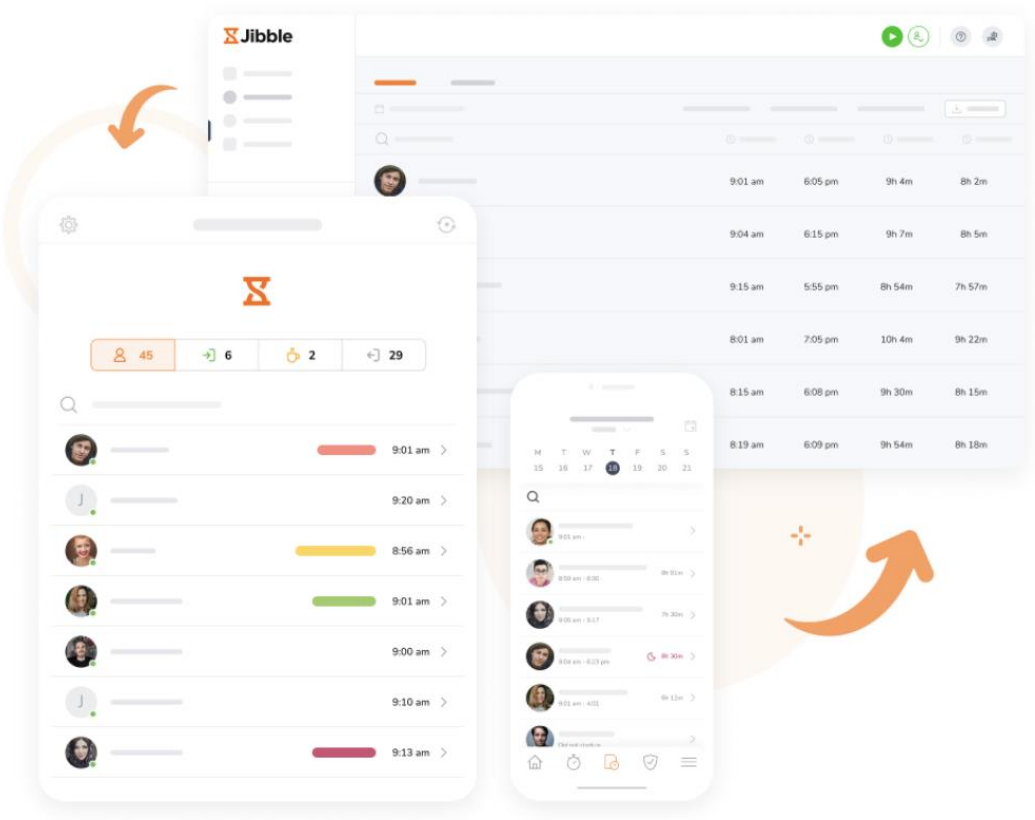

Рис. 1.3. Інтерфейс Jibble

1.5 Порівняльний аналіз аналогів і обґрунтування актуальності розробки

У таблиці 1.1 зведено результати аналізу існуючих веб-застосунків для обліку робочого часу співробітників малого бізнесу. Порівняння виконано за ознаками, які мають найбільше значення для нашої предметної області: тип застосунку, цільова аудиторія, логіка обліку часу, робота зі змінами та графіками, рівень придатності для малого бізнесу, а також структурні обмеження готових сервісів. Характеристики сервісів узгоджено з їхніми офіційними сторінками функцій і тарифів.

Таблиця 1.1

Порівняльний аналіз існуючих веб-застосунків для обліку робочого часу співробітників малого бізнесу

Показник	Clockify	Toggl Track	Jibble
Тип застосунку	Універсальна система обліку часу та attendance	Система time tracking і аналітики робочого часу	Система attendance, time tracking і scheduling
Цільова аудиторія	Команди, менеджери, компанії, що ведуть облік часу й звітність	Команди, фахівці, проєктні групи, користувачі з аналітичним фокусом	Команди, роботодавці, працівники, малі підприємства
Основна логіка обліку	Timer, timesheet, kiosk, attendance, reporting	Timer, calendar view, timesheets, reporting, integrations	Clock-in/clock-out, attendance, automated timesheets
Робота зі змінами та графіками	Частково реалізована через planning, attendance і kiosk	Орієнтація переважно на часові записи, а не на змінний розклад	Є scheduling, attendance і табелювання змін
Робота з ролями користувачів	Є адміністративне керування в межах workspace	Є командна модель доступу	Є централізоване керування працівниками й attendance

Продовження таблиці 1.1

Показник	Clockify	Toggl Track	Jibble
Звітність	Розвинена звітність, attendance, overtime, payroll-oriented дані	Розвинена аналітика, time reports, profitability	Звіти щодо attendance, timesheets, payroll hours
Інтеграція з іншими сервісами	Підтримується	Підтримується на розвиненому рівні	Є, але акцент зроблено на attendance-сценарії
Придатність для малого бізнесу	Висока, але з функціональною надмірністю	Помірна, більше підходить для проєктного time tracking	Висока, найближча до задач attendance і змін
Основні обмеження	Надлишкова універсальність, частина функцій платна	Проєктно-аналітичний фокус, а не внутрішнє адміністрування змін	Залежність від готової зовнішньої моделі сервісу
Відповідність темі роботи	Часткова	Часткова	Найвища серед розглянутих, але не повна

Проведене порівняння показує, що Clockify, Toggl Track і Jibble дійсно покривають значну частину задач, пов’язаних із фіксацією робочого часу, attendance, timesheets і звітністю. Водночас жоден із цих сервісів не відтворює повністю ту логіку, яка є пріоритетною для спеціалізованого веб-застосунку

обліку робочих годин співробітників малого бізнесу. Clockify поєднує timekeeping, attendance, planning і payroll-oriented функції, але будується як універсальна платформа для широкого кола бізнес-сценаріїв. Toggl Track орієнтується переважно на time tracking, reports, calendar view і інтеграції, через що тяжіє до проєктної моделі використання. Jibble є найближчим аналогом, оскільки підтримує attendance, clock-in/clock-out, scheduling і payroll-ready timesheets, проте все одно залишається зовнішнім сервісом із наперед заданою моделлю роботи.

Актуальність розробки власного веб-застосунку зумовлена тим, що малому бізнесу потрібне не просто універсальне рішення для time tracking, а система, побудована навколо конкретних внутрішніх сутностей: співробітників, підрозділів, графіків роботи, робочих змін, статусів змін, профілів користувачів і службових повідомлень. Готові аналоги або перевантажені додатковими функціями, або орієнтовані на проєктний облік часу, або ж накладають зовнішню модель конфігурації, яку не можна повністю підлаштувати під конкретну організацію. Саме тому розробка власного веб-застосунку є обґрунтованою, оскільки вона дає змогу реалізувати функціональність, структурно узгоджену з реальною організацією обліку робочих годин у малому бізнесі.

1.6 Визначення функціональних і нефункціональних вимог до веб-застосунку

Проаналізувавши можливості аналогічних платформ, особливості організації обліку робочих годин у малому бізнесі та структуру розроблюваного застосунку, було сформульовано такі функціональні вимоги до веб-застосунку обліку робочих годин співробітників малого бізнесу:

- Реєстрація користувачів, вхід до системи та доступ до захищених сторінок після автентифікації;

- Розмежування прав доступу між користувачами з ролями адміністратора і працівника;

- Ведення даних про співробітників із можливістю перегляду, додавання, редагування та адміністрування;

- Ведення даних про підрозділи підприємства та прив’язка працівників до відповідних структурних одиниць;

- Формування, перегляд і коригування графіків роботи співробітників;

- Створення, редагування та перегляд робочих змін із фіксацією часових параметрів;

- Відображення інформаційної панелі з основними робочими даними системи;

- Перегляд і редагування персонального профілю користувача;

- Підтримка налаштувань системи для роботи з внутрішніми параметрами застосунку;

- Обробка службових

повідомлень і подій, пов’язаних із роботою користувачів, змін і графіків.

Нефункціональні вимоги:

- Висока продуктивність - система повинна забезпечувати швидке завантаження сторінок, оперативну обробку запитів і зручну роботу з даними про працівників, підрозділи, зміни та графіки;

- Надійність - застосунок повинен зберігати цілісність даних і коректно працювати під час виконання типових операцій адміністрування та перегляду інформації;

- Безпека - доступ до захищених сторінок повинен виконуватися лише за наявності сесії, а права користувача мають визначатися відповідно до його ролі;

- Керованість доступом - система повинна підтримувати механізми контролю доступу залежно від ролі користувача та, за потреби, конфігурації типу пристрою;

- Масштабованість - архітектура застосунку повинна допускати розширення функціональності шляхом додавання нових модулів і сервісів без повного перепроєктування системи;

- Підтримуваність - кодова база повинна мати модульну структуру, придатну для супроводу, локальних змін і подальшого розвитку;

- Зручність використання - інтерфейс повинен забезпечувати зрозумілу навігацію між основними сторінками та скорочувати кількість зайвих дій під час щоденної роботи;

- Адаптивність - застосунок повинен коректно працювати в браузерному середовищі та підтримувати керований режим доступу для різних типів пристроїв. Ці вимоги узгоджуються з реалізованими механізмами перевірки сесії в middleware, перенаправленням на сторінки входу або інформаційної панелі, конфігурованим контролем доступу для ПК, планшетів і мобільних пристроїв, а також із клієнтським збереженням даних користувача й ознаки адміністратора у localStorage.

Сформульовані функціональні та нефункціональні вимоги визначають межі подальшого проєктування веб-застосунку, оскільки вони відображають не лише потреби цільової аудиторії, а й фактичну логіку реалізації системи, закладену в її клієнтській і серверній структурах. На їх основі може бути обґрунтовано вибір засобів реалізації, архітектурних рішень і способів організації взаємодії між окремими модулями застосунку.

1.7 Постановка завдання, визначення об’єкта, предмета, мети та завдань роботи

Результати аналізу предметної області та існуючих аналогів дають підстави сформулювати прикладне завдання розробки веб-застосунку для обліку робочих годин співробітників малого бізнесу як завдання створення єдиного цифрового середовища, у межах якого поєднуються адміністрування користувачів, ведення підрозділів, формування графіків роботи, призначення робочих змін, фіксація

часових параметрів зайнятості, контроль статусів змін і розмежування доступу відповідно до ролі користувача. Потреба в такому рішенні зумовлена тим, що на практиці облік робочого часу в малих підприємствах часто реалізується фрагментарно: частина відомостей зберігається в електронних таблицях, частина передається через повідомлення, а частина не має стабільного цифрового відображення. Такий підхід ускладнює підтримання актуальності даних, не забезпечує структурованого збереження записів про зміни й графіки, знижує швидкість адміністративних дій та підвищує ризик неузгодженості між фактичним і запланованим режимом роботи персоналу.

Постановка завдання передбачає розробку веб-застосунку, здатного усунути зазначені обмеження шляхом централізації облікових процесів і приведення їх до єдиної логіки взаємодії між користувачами системи. Програмне рішення має забезпечувати автентифікацію, роботу з персональними профілями, адміністрування списку співробітників, керування підрозділами, формування та коригування графіків роботи, створення й редагування робочих змін, підтримку статусної моделі опрацювання змін і подій, а також відображення даних через інтерфейс, придатний для щоденного використання в умовах малого бізнесу. Оскільки предметна область пов’язана з часовими інтервалами, календарною структурою та змінністю зайнятості, система має відображати не лише окремі записи, а й взаємозв’язки між працівником, підрозділом, графіком, конкретною зміною та її службовим станом.

Завдання розробки охоплює побудову цілісної моделі цифрового обліку робочого часу, у якій дані про працівників, підрозділи, графіки й зміни зберігаються централізовано та опрацьовуються через узгоджену клієнт-серверну взаємодію. Серверна частина має відповідати за збереження, перевірку й обробку даних, тоді як клієнтська частина повинна забезпечувати зручний доступ до основних функцій системи через веб-інтерфейс. Для адміністратора передбачається можливість керування користувачами, структурними одиницями, графіками роботи та змінами, а для працівника — перегляд персональних даних, призначених змін і пов’язаних із ними відомостей. Такий розподіл функцій

забезпечує відповідність системи реальним сценаріям використання в малому бізнесі.

Розроблюваний веб-застосунок має підтримувати рольову модель доступу, за якої користувачі отримують лише ті функціональні можливості, що відповідають їхньому статусу в системі. Адміністратор повинен мати доступ до розширених засобів керування персоналом, підрозділами, графіками та робочими змінами, тоді як працівник має взаємодіяти переважно з власним профілем, персональним розкладом і призначеними змінами. Такий підхід знижує ризик несанкціонованого редагування даних, спрощує роботу з інтерфейсом і дає змогу чітко розмежувати адміністративні та користувацькі дії.

Функціональна модель системи має охоплювати створення й редагування записів про співробітників, формування структури підрозділів, призначення працівників до відповідних організаційних одиниць, створення графіків роботи, внесення робочих змін, відображення статусів їх опрацювання та підтримку службових повідомлень. Окреме значення має узгодженість між усіма сутностями системи: зміна повинна бути пов’язана з конкретним працівником, підрозділом, часовим проміжком і статусом, а графік роботи — відображати актуальний стан зайнятості персоналу. За такої організації дані не існують як ізольовані записи, а формують взаємопов’язану інформаційну структуру.

Технічна реалізація веб-застосунку має забезпечувати стабільну роботу клієнтської та серверної частин, коректну взаємодію з базою даних, обробку запитів користувачів, підтримку автентифікації та захист доступу до службових сторінок. Система повинна бути придатною до подальшого розширення, зокрема шляхом додавання нових модулів, уточнення рольової моделі, розширення механізмів сповіщень або впровадження додаткових засобів аналітики. Модульна побудова дає змогу підтримувати окремі частини застосунку без повного перепроєктування його архітектури.

Отже, постановка завдання на розробку веб-застосунку зводиться до створення прикладної системи, яка впорядковує процес обліку робочих годин співробітників малого бізнесу, централізує дані про персонал, підрозділи,

графіки й робочі зміни, забезпечує розмежування доступу та надає користувачам інструменти для щоденного адміністрування робочого часу в єдиному цифровому середовищі.

2 ЗАСОБИ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ

2.1 Середовище та організація розробки

Середовище та організація розробки визначають технічну основу створення веб-застосунку, оскільки саме через них забезпечується узгоджена робота з клієнтською частиною, серверною логікою, конфігурацією проєкту, системою керування версіями та засобами розгортання. Для веб-застосунку «Система обліку робочих годин співробітників малого бізнесу» така організація має особливе значення, оскільки проєкт побудований як full-stack рішення з окремими каталогами work-core-client і workCore-server, файлом конфігурації .env.example та docker-compose.yml, а основною мовою реалізації є TypeScript. Це свідчить про модульну побудову системи та потребу в середовищі, яке підтримує одночасну роботу з кількома технологічними рівнями.

2.1.1 Visual Studio Code як середовище розробки

Для створення веб-застосунку доцільно використовувати Visual Studio Code як основне середовище розробки, оскільки цей редактор забезпечує ефективну роботу з багатомодульними веб-проєктами, у яких поєднуються клієнтська і серверна частини. У структурі розглядуваного проєкту клієнтський застосунок реалізовано на основі Next.js 15, React 19 і TypeScript, а серверну частину побудовано на NestJS, Prisma та TypeScript. За таких умов Visual Studio Code є зручним інструментом для одночасної роботи з компонентами інтерфейсу, серверними модулями, конфігурацією запуску, контейнерними файлами та службовими скриптами.

Практична придатність цього середовища для розробки веб-застосунку зумовлена тим, що воно підтримує навігацію по розгалуженій структурі каталогу, роботу з TypeScript, реактивними компонентами, серверними маршрутами, конфігураційними файлами та допоміжними засобами розробки. Для клієнтської частини це важливо через наявність окремих маршрутів dashboard, departments,

employees, profile, schedule і settings, а для серверної - через модулі auth, department, user, notifications, work.schedule, work.shift, work.shift.tag та інші. У межах одного редактора це дає змогу швидко переходити між взаємопов’язаними частинами системи й підтримувати цілісність логіки застосунку.

З погляду організації розробки Visual Studio Code виконує не лише функцію редактора коду, а й роль центрального робочого середовища, у якому зосереджено редагування клієнтських і серверних модулів, перевірку структури проєкту, запуск локальних сценаріїв та взаємодію з іншими інструментами розробки. Такий підхід є придатним для веб-застосунку обліку робочих годин, оскільки дає змогу підтримувати узгоджену роботу з інтерфейсом, бізнес-логікою, даними та конфігурацією розгортання.

2.1.2 GitHub як засіб керування версіями проєкту

Керування версіями проєкту є необхідною складовою сучасної програмної розробки, оскільки воно забезпечує фіксацію змін у кодовій базі, підтримання історії розвитку системи та впорядковану роботу з кількома технологічними компонентами. Для цього проєкту таким засобом виступає GitHub, де розміщено репозиторій GerychCode/PsCore. Репозиторій містить 21 коміт, окремі каталоги клієнтської та серверної частин, файл конфігурації середовища та опис контейнерного запуску. Це дає підстави розглядати GitHub не лише як місце зберігання коду, а як інструмент організації розробки всього веб-застосунку.

Використання GitHub у межах такого проєкту забезпечує структуроване оновлення клієнтської й серверної частин, контроль змін у конфігураційних файлах та збереження цілісності кодової бази під час поступового розвитку системи. Для full-stack застосунку це має принципове значення, оскільки зміни в інтерфейсі, серверній логіці, моделі даних і контейнерному запуску мають бути узгодженими. Репозиторна організація дає змогу відслідковувати еволюцію проєкту, повертатися до попередніх станів і підтримувати впорядкований цикл розробки.

З практичної точки зору GitHub також виконує функцію середовища представлення структури проєкту. Через нього чітко простежується розподіл системи на work-core-client і workCore-server, що відповідає поділу на клієнтську та серверну частини. Така організація сприяє прозорості архітектури й полегшує подальший супровід системи.

2.1.3 Docker і Docker Compose для контейнеризації та розгортання

Контейнеризація є важливим елементом організації розробки та запуску веб-застосунку, оскільки вона дозволяє відтворити стабільне середовище виконання для всіх частин системи незалежно від локальної конфігурації комп’ютера. У розглядуваному проєкті для цього використано Docker і Docker Compose. Файл docker-compose.ymlзадає окремі сервіси для серверної частини на NestJS, клієнтської частини на Next.js, бази даних PostgreSQL і кешу Redis. Для серверного контейнера передбачено виконання міграцій Prisma та запуск production-версії застосунку, тоді як клієнтський контейнер підключає каталог work-core-client і працює як окремий фронтенд-сервіс.

Такий підхід дає змогу розглядати контейнеризацію як не просто технічний спосіб запуску, а як елемент архітектурної організації системи. Кожна частина застосунку функціонує у власному контейнері, однак усі вони поєднані в межах єдиної конфігурації. Це забезпечує чітке розмежування обов’язків між клієнтом, сервером, сховищем даних і кешем, а також спрощує локальне тестування та подальше розгортання.

Для веб-застосунку обліку робочих годин такий спосіб організації є виправданим, оскільки система використовує кілька взаємопов’язаних компонентів: клієнтську частину з інтерфейсом користувача, серверні модулі автентифікації, користувачів, підрозділів, змін і графіків, базу даних для збереження структурованої інформації та Redis як допоміжний сервіс. Використання Docker Compose дозволяє запускати всю систему в єдиному узгодженому середовищі, що підвищує відтворюваність розробки, спрощує

конфігурування та створює надійну основу для подальшого супроводу й розгортання.

2.2 Технології клієнтської частини

Клієнтська частина веб-застосунку є тим рівнем системи, через який користувач взаємодіє з основною функціональністю: переглядає інформаційну панель, працює зі сторінками співробітників, підрозділів, графіків роботи, профілю та налаштувань.

У цьому проєкті клієнтський застосунок реалізовано на основі Next.js 15, React 19 і TypeScript 5. У складі залежностей також використано Zustand, TanStack React Query, Axios, Recharts, react-hook-form, react-day-picker, react-modal, socket.io-client, sonner, sass і motion. Такий стек свідчить, що клієнтська частина побудована як сучасний веб-застосунок із маршрутизацією, компонентною структурою, керуванням локальним станом, обробкою API-запитів, формами, візуалізацією даних і допоміжними засобами взаємодії з користувачем.

2.2.1 Next.js як основа побудови клієнтського веб-застосунку

Для реалізації клієнтської частини веб-застосунку використано Next.js, який є каркасом для створення сучасних React-застосунків і надає розробнику впорядковану основу для організації маршрутизації, структури сторінок, роботи з middleware та інтеграції клієнтської логіки в межах єдиного проєкту. У контексті системи обліку робочих годин такий вибір є виправданим, оскільки застосунок повинен підтримувати кілька окремих функціональних сторінок, пов’язаних між собою спільною логікою доступу та навігації.

Практичне значення Next.js у цьому проєкті полягає в тому, що він забезпечує чітку організацію клієнтської структури через маршрути й службові механізми платформи. Застосунок містить захищені сторінки для інформаційної панелі, співробітників, профілю, графіка роботи та підрозділів, а також окремий блок автентифікації. Додатково в middleware реалізовано перевірку сесії,

перенаправлення між сторінками входу і внутрішніми сторінками системи, а також логіку контролю доступу залежно від типу пристрою. Це означає, що Next.js у межах цього веб-застосунку виконує не лише роль засобу побудови інтерфейсу, а й функцію організації доступу до клієнтської частини та координації поведінки користувача в межах системи.

Для веб-застосунку обліку робочих годин це має особливу цінність, оскільки користувачі працюють із різними сценаріями: адміністратор повинен мати доступ до сторінок керування персоналом, підрозділами й графіком, тоді як працівник взаємодіє насамперед із профілем і власними робочими даними. Використання Next.js дозволяє структуровано реалізувати таку модель доступу, не перетворюючи клієнтський код на набір неузгоджених сторінок. У результаті клієнтська частина зберігає цілісність і придатність до подальшого розширення.

2.2.2 React і TypeScript у реалізації користувацького інтерфейсу

Користувацький інтерфейс веб-застосунку реалізовано на основі React і TypeScript. Поєднання цих технологій є доцільним для створення системи, у якій потрібно підтримувати велику кількість взаємопов’язаних компонентів, форм, сторінок і сервісів доступу до даних. React забезпечує компонентний підхід до побудови інтерфейсу, а TypeScript додає типову строгість, що підвищує передбачуваність поведінки компонентів і зменшує кількість помилок у роботі з даними. У залежностях клієнтської частини зафіксовано використання React 19, react-dom 19 і TypeScript 5, що підтверджує їхню роль як базових технологій інтерфейсного рівня.

Для системи обліку робочих годин компонентний підхід має принципове значення. Інтерфейс такого застосунку не є однорідним: він охоплює інформаційну панель, таблиці, форми введення та редагування, модальні вікна, календарні елементи, повідомлення про події й окремі сторінки для різних ролей користувача. React дозволяє реалізувати ці частини у вигляді самостійних компонентів, що спрощує супровід і повторне використання коду. TypeScript, своєю чергою, дає змогу формалізувати структури даних користувача, параметри

запитів, інтерфейси об’єктів і сигнатури функцій, завдяки чому підвищується керованість клієнтської кодової бази.

У цьому проєкті значення TypeScript посилюється ще й тим, що він використовується як на клієнтському, так і на серверному рівні. Така уніфікація спрощує узгодження між інтерфейсами даних, моделями користувача, логікою ролей і взаємодією з API. Для фронтенду це означає, що робота з профілем, роллю адміністратора, параметрами сторінок і сервісними викликами відбувається в більш передбачуваному типізованому середовищі. Це важливо для застосунку, у якому помилки в передачі чи інтерпретації даних можуть призводити до некоректного відображення змін, графіків або прав доступу.

2.2.3 Zustand, TanStack Query, Axios та допоміжні бібліотеки

Функціональність базових технологій клієнтської частини розширено за допомогою допоміжних бібліотек, які забезпечують реалізацію конкретних вимог веб-застосунку. У межах цього проєкту центральне значення мають Zustand, TanStack React Query та Axios, оскільки саме вони формують механізми локального стану, серверної синхронізації й доступу до API.

Zustand використовується для керування клієнтським станом користувача. У файлі user.store.ts зафіксовано збереження поточного користувача, ознаки адміністратора, а також методів оновлення профілю і виходу з системи. Додатково стан зберігається в localStorage під ключем workcore-user-storage, а ознака адміністратора визначається через роль користувача. Для системи обліку робочих годин це має безпосередню практичну цінність, оскільки рольова логіка інтерфейсу повинна працювати стабільно й бути доступною одразу після відновлення клієнтського стану.

TanStack React Query використовується для організації роботи із серверними даними. У такого типу веб-застосунку значна частина інтерфейсу залежить від актуальних відомостей, що надходять із серверної частини: списків працівників, відомостей про підрозділи, графіків, змін і службових повідомлень. React Query надає впорядкований механізм виконання запитів, синхронізації

отриманих результатів і повторного оновлення даних за потреби. Це особливо важливо для внутрішньої інформаційної системи, у якій інтерфейс має відображати поточний стан організаційних даних, а не застарілі локальні значення. Наявність @tanstack/react-query і devtools у залежностях клієнтської частини підтверджує використання цієї бібліотеки як окремого шару взаємодії з сервером.

Axios виконує роль клієнтського засобу надсилання HTTP-запитів. Його використання в межах цього проєкту є природним, оскільки клієнтська частина повинна звертатися до серверних ендпоінтів для автентифікації, отримання даних про користувачів, підрозділи, зміни та графіки. Порівняно з прямим використанням низькорівневих механізмів, Axios спрощує організацію сервісного шару й робить роботу з API більш однорідною. Сам факт наявності окремих сервісів для auth, department, shift, shift.tag, user і work.schedule у структурі клієнтської частини свідчить про системний підхід до інкапсуляції запитів, у якому Axios виконує роль технічної основи.

Поряд із цими бібліотеками у фронтенді використано й інші допоміжні засоби. react-hook-form забезпечує зручну роботу з формами, що є суттєвим для сторінок входу, редагування профілю або керування сутностями системи. react-day-picker придатний для сценаріїв, пов’язаних із календарними інтерфейсами та вибором дат. react-modalпідтримує використання модальних вікон. recharts дає можливість візуалізувати дані, що може бути застосовано в інформаційній панелі або аналітичних блоках. socket.io-client створює основу для клієнтської участі в подієвій взаємодії. sonner придатний для відображення сповіщень, sass - для організації стилів, а motion - для анімованої взаємодії елементів інтерфейсу. У сукупності ці бібліотеки не формують ядро системи, але значно розширюють можливості реалізації зручного й функціонально насиченого клієнтського застосунку.

Отже, технології клієнтської частини веб-застосунку утворюють узгоджену інтерфейсну основу системи. Next.js визначає архітектурну організацію фронтенду і маршрутизацію, React і TypeScript забезпечують компонентну

побудову та типізовану реалізацію інтерфейсу, а Zustand, TanStack Query, Axios і допоміжні бібліотеки формують рівень стану, запитів, форм, модальних вікон, календарної логіки та допоміжної взаємодії. Саме така комбінація технологій є придатною для побудови веб-застосунку обліку робочих годин співробітників малого бізнесу, у якому потрібні одночасно структурованість, зручність взаємодії та підтримка роботи з динамічними даними.

2.2 Технології серверної частини

Серверна частина веб-застосунку забезпечує обробку бізнес-логіки, взаємодію з базою даних, контроль доступу, керування сесіями, валідацію вхідних даних і координацію роботи окремих функціональних модулів системи. Для веб-застосунку обліку робочих годин співробітників малого бізнесу такий рівень є визначальним, оскільки саме на ньому реалізуються операції, пов’язані з користувачами, підрозділами, графіками роботи, змінами, повідомленнями та службовими станами. У серверному каталозі проєкту використано NestJS 10, Prisma Client, PostgreSQL, Redis, express-session, cookie-parser, bcrypt, class-validator, class-transformer, socket.io, nestjs-telegraf і telegraf, що вказує на модульну архітектуру з окремими механізмами зберігання даних, автентифікації, сесійної взаємодії та допоміжних інтеграцій.

2.3.1 NestJS як основа серверної архітектури

Для реалізації серверної частини веб-застосунку використано NestJS, який є фреймворком для побудови структурованих Node.js-систем із чітким розподілом відповідальності між модулями, сервісами й контролерами. Такий вибір є виправданим для розробки системи обліку робочих годин, оскільки серверна логіка цього проєкту не обмежується одним набором CRUD-операцій, а охоплює роботу з кількома взаємопов’язаними предметними сутностями. У головному модулі застосунку підключено PrismaModule, AuthModule, UserModule, DepartmentModule, WorkShiftModule, WorkScheduleModule,

WorkShiftTagModule і TelegramModule, а також глобальний ConfigModule. Така композиція підтверджує, що серверна архітектура побудована як набір спеціалізованих модулів, кожен із яких відповідає за окремий фрагмент бізнес-логіки.

Практична придатність NestJS для цього проєкту пов’язана з тим, що він дає змогу підтримувати впорядковану архітектуру серверної частини в умовах зростання кількості функціональних компонентів. Для системи, у якій потрібно одночасно адмініструвати користувачів, підрозділи, зміни, графіки та повідомлення, модульна побудова є не лише зручною, а й необхідною. У main.ts додатково реалізовано глобальну конфігурацію CORS, ValidationPipe для перевірки вхідних даних, підключення cookie parsing, сесійного механізму та сервісу статичних файлів. Це означає, що NestJS у межах цього проєкту виконує не лише роль каркаса серверних модулів, а й функцію єдиного середовища виконання для безпеки, валідації, конфігурації та прикладної логіки.

2.3.2 Prisma ORM як засіб взаємодії з базою даних

Для взаємодії серверної частини з базою даних використано Prisma ORM. У схемі schema.prisma прямо визначено datasource db з провайдером postgresql, а також набір моделей, що відображають основні сутності предметної області: User, Department, WorkShift, Tag, WorkSchedule, WorkScheduleLock і Not ification. Додатково описано переліки Role, ShiftStatus і NotificationType, які формалізують рольову модель користувачів, стани змін і типи службових повідомлень. Така структура свідчить, що Prisma використовується не як допоміжний інструмент, а як центральний засіб формалізації серверної моделі даних.

Застосування Prisma є доцільним для цієї системи через чітку структурованість даних і необхідність підтримувати зв’язки між сутностями. Наприклад, модель User пов’язана зі змінами, графіками й повідомленнями, модель Department - зі змінами, графіками й блокуванням розкладу,

а WorkShift містить посилання на працівника, підрозділ, часові параметри та статус опрацювання. Така модель добре відповідає предметній області обліку робочих годин, де інформація повинна бути не просто збережена, а логічно пов’язана. У практичному вимірі Prisma спрощує підтримання цілісності серверної частини, оскільки дозволяє працювати з типізованою моделлю бази даних, узгодженою з бізнес-логікою застосунку. Окремим підтвердженням її інтеграції в життєвий цикл системи є команда запуску сервера через npx prisma migrate deploy && npm run start:prod, яка використовується в контейнеризованому розгортанні.

2.3.3 Redis, сесійний механізм і засоби автентифікації

Суттєве місце в серверній частині займають Redis, сесійний механізм і засоби автентифікації, оскільки вони визначають модель безпечного доступу до функцій веб-застосунку. У складі серверних залежностей присутні express-session, connect-redis, ioredis, cookie-parser і bcrypt, що вказує на використання сесійної автентифікації з Redis як сховищем сесій та cookie-based взаємодією між клієнтом і сервером. У main.ts реалізовано створення Redis-клієнта через ioredis, підключення RedisStore, налаштування секрету сесії, імені cookie, httpOnly, secure, sameSite, часу життя cookie та префікса для збереження сесій. Це означає, що серверна частина не покладається на спрощену локальну схему збереження стану, а використовує окремий механізм централізованого зберігання сесій.

Застосування такого підходу є виправданим для веб-застосунку обліку робочих годин, оскільки система працює з різними ролями користувачів і захищеними сторінками. На рівні серверної архітектури це підтримується наявністю AuthModule, який імпортує UserModule, а на рівні моделі даних - полем role у моделі User, де передбачено значення Employe і Admin. Отже, автентифікація в цьому проєкті не відокремлена від предметної логіки, а безпосередньо пов’язана з ролями, які визначають характер доступу до функцій системи. Використання bcrypt у залежностях додатково вказує на застосування

хешування паролів як складової захисту облікових даних. У сукупності Redis, сесійний механізм, cookie-based взаємодія та модуль автентифікації формують серверний рівень безпеки, необхідний для коректної роботи системи з адміністративними та персональними сценаріями доступу.

У підсумку технології серверної частини утворюють цілісну основу для реалізації веб-застосунку. NestJS забезпечує модульну архітектуру й керовану організацію бізнес-логіки, Prisma ORM формалізує взаємодію з реляційною базою даних і предметними сутностями системи, а Redis разом із сесійним механізмом і засобами автентифікації підтримує захищений доступ до функцій застосунку. Саме така комбінація технологій є придатною для серверної реалізації системи обліку робочих годин співробітників малого бізнесу.

2.3 База даних та інструменти зберігання інформації

База даних у веб-застосунку обліку робочих годин виконує не лише функцію технічного збереження записів, а формує структурну основу всієї системи, оскільки саме через неї забезпечується зв’язок між користувачами, підрозділами, графіками роботи, окремими змінами, службовими повідомленнями та допоміжними обмеженнями, пов’язаними з календарною логікою. Для цього проєкту така роль бази даних є визначальною, оскільки застосунок працює з формалізованими сутностями, для яких важливі цілісність зв’язків, однозначність полів і можливість коректно відтворювати стан організаційного процесу в кожний момент часу. У серверній частині це підтверджується використанням postgresql як джерела даних у Prisma-схемі, а також окремим контейнером postgres:16у конфігурації Docker Compose.

У межах цього проєкту інструменти зберігання інформації побудовано як поєднання реляційної бази даних PostgreSQL, ORM-рівня Prisma та допоміжного сервісу Redis. PostgreSQL відповідає за збереження основних предметних даних, Prisma формалізує структуру моделей і взаємодію серверної логіки з таблицями, а Redis використовується як окремий інфраструктурний компонент, що

підтримує сесійний механізм і пов’язану з ним службову взаємодію. Така побудова є логічною для системи, у якій потрібно одночасно зберігати сталі організаційні дані та підтримувати динамічний стан доступу користувачів до захищених функцій.

2.4.1 PostgreSQL як система керування базами даних

Для збереження основної інформації у веб-застосунку використано PostgreSQL. Це безпосередньо зафіксовано у Prisma-схемі через provider = "postgresql", а також у конфігурації Docker Compose, де для бази даних виділено окремий сервіс postgres на основі образу postgres:16. Такий вибір є обґрунтованим для системи обліку робочих годин, оскільки предметна область містить чітко структуровані сутності та стабільні зв’язки між ними: працівник належить до певної ролі, зміна пов’язана з конкретним користувачем і підрозділом, графік також має прив’язку до користувача й підрозділу, а повідомлення адресуються конкретному користувачеві. Реляційна модель тут є природною, оскільки дозволяє підтримувати логічну узгодженість між записами й запобігати втраті структурних зв’язків між елементами системи.

PostgreSQL у цьому проєкті доцільно розглядати як основу для роботи з організаційними даними, для яких важлива не лише наявність окремих полів, а і коректність зв’язків між таблицями. Наприклад, у моделі WorkShiftзберігаються посилання на працівника і підрозділ, дата зміни, час початку, час завершення, загальна кількість годин і статус. У моделі WorkSchedule аналогічно поєднуються посилання на користувача, підрозділ, дату, часові межі та ознаку вихідного дня. Для таких структур реляційна база даних є зручною, оскільки забезпечує формалізований опис сутностей, уможливлює контроль унікальності, підтримує чіткі зовнішні зв’язки й добре пристосована до предметної області, де кожен запис має бути інтерпретований у зв’язку з іншими елементами системи.

Окремою перевагою використання PostgreSQL у межах цього проєкту є його включення до контейнеризованого середовища запуску. У Docker Compose для бази передбачено окремий контейнер, змінні середовища, зовнішній порт та

змонтований том postgres_data, що забезпечує відокремлене збереження даних. Для серверного контейнера додатково визначено змінну DATABASE_URL, яка формується на основі параметрів підключення до PostgreSQL. Це означає, що база даних інтегрована не як локальний тимчасовий компонент, а як повноцінна частина узгодженого середовища розгортання, придатного для відтворення на різних машинах і в різних етапах життєвого циклу системи.

2.4.2 Особливості зберігання даних про користувачів, зміни, графіки й підрозділи

Структура зберігання даних у цьому веб-застосунку відображає основні сутності предметної області обліку робочих годин. Центральне місце займає модель User, у якій містяться персональні дані працівника або адміністратора: ідентифікатор, Telegram ID, аватар, ім’я, прізвище, електронна пошта, телефон, адреса, хеш пароля, роль, дата народження, а також часові позначки створення й оновлення запису. Через зв’язки ця модель пов’язана зі змінами, графіками роботи та повідомленнями. Така побудова є логічною, оскільки саме користувач виступає вихідною точкою для інтерпретації значної частини даних системи. Наявність поля role з варіантами Employe і Admin свідчить, що модель користувача одночасно виконує функцію ідентифікації особи та розмежування доступу до функцій системи.

Модель Department забезпечує збереження даних про підрозділи підприємства. Вона містить назву, Telegram ID, параметри часу відкриття та закриття в будні й вихідні дні, адресу, географічні координати та ознаку активності. Через зв’язки підрозділ пов’язується зі змінами, графіками та блокуваннями розкладу. Це означає, що підрозділ у системі є не допоміжним довідником, а самостійною організаційною сутністю, у межах якої відбувається планування й облік робочого часу. Така модель зберігання добре відповідає реаліям малого бізнесу, де окремі точки, відділи або локації мають власний режим роботи та власний набір працівників.

Модель WorkShift відображає фактичну або призначену робочу зміну. У ній зберігаються ідентифікатори користувача й підрозділу, дата, час початку, час завершення, загальна кількість годин, статус зміни, а також часові позначки створення й оновлення. Додатково зміна має зв’язок із тегами через відношення ShiftTags. Наявність окремого переліку ShiftStatus зі значеннями PENDING, APPROVED і REJECTED показує, що система працює не лише з фіксацією часових меж, а й зі службовим станом запису. Це є суттєвою особливістю моделі зберігання, оскільки зміна трактується не просто як часовий інтервал, а як об’єкт адміністративного опрацювання.

Модель WorkSchedule відображає графік роботи як окремий рівень зберігання даних, відмінний від фактичних змін. У ній зберігаються посилання на користувача й підрозділ, дата, часові межі та ознака вихідного дня. Такий поділ між WorkSchedule і WorkShift є методологічно виправданим, оскільки дає змогу відокремити плановий розклад від конкретної зміни, яка має власний статус і додаткові службові атрибути. Для системи обліку робочих годин це важливо, адже в реальній роботі графік може існувати як запланована структура зайнятості, тоді як зміна може проходити окремий цикл створення, уточнення, погодження або відхилення.

Окремим елементом структури зберігання є WorkScheduleLock, у якому фіксуються блокування графіка на рівні підрозділу і тижня. Модель містить посилання на підрозділ, дату початку тижня та ознаку блокування, причому для пари departmentId, weekStart визначено унікальність. Це вказує на наявність у системі додаткового механізму контролю над календарною структурою розкладу. Для малого бізнесу така можливість є корисною, оскільки дозволяє уникати хаотичних змін у вже сформованих графіках і підтримувати стабільність планування на визначені періоди.

Система також передбачає збереження службових повідомлень у моделі Notification, де для кожного запису визначено користувача, заголовок, текст повідомлення, тип, ознаку прочитання та час створення.

Перелік NotificationType міститьзначення INFO, WARNING, SUCCESS і E RROR, що дозволяє класифікувати повідомлення за їхнім призначенням. Наявність окремої моделі повідомлень означає, що система не обмежується зберіганням статичних організаційних даних, а підтримує подієвий рівень взаємодії, у якому користувачі отримують службову інформацію про зміни стану системи.

Отже, зберігання даних у веб-застосунку побудовано як структурована реляційна модель, у якій кожна ключова сутність предметної області має окреме відображення в базі даних. PostgreSQL забезпечує стабільне середовище збереження таких даних, а Prisma формалізує їхню організацію через моделі, зв'язки та переліки. У результаті система отримує впорядковану основу для роботи з користувачами, підрозділами, графіками, змінами та службовими повідомленнями, що є необхідною умовою для коректної реалізації веб-застосунку обліку робочих годин співробітників малого бізнесу.

2.4 Додаткові технології та інтеграції

Додаткові технології та інтеграції в межах веб-застосунку виконують не допоміжну, а функціонально значущу роль, оскільки розширюють стандартну модель взаємодії між клієнтською і серверною частинами. Для цього проєкту такими засобами є Socket.IO та Telegram-інтеграція. Їх використання засвідчено як складом залежностей серверної і клієнтської частин, так і наявністю окремих модулів для подій, сповіщень і Telegram-взаємодії на серверному рівні.

У серверній частині підключено @nestjs/platform-socket.io, @nestjs/websockets, socket.io, nestjs-telegraf і telegraf, а в клієнтській - socket.io-client.

2.5.1 Socket.IO для подій і сповіщень

Використання Socket.IO у цьому веб-застосунку доцільно розглядати як основу для подієвої взаємодії між сервером і клієнтом. На рівні серверних

залежностей присутні @nestjs/platform-socket.io, @nestjs/websockets і socket.io, а на рівні клієнтської частини - socket.io-client, що вказує на закладену в архітектуру можливість двостороннього обміну подіями в реальному часі між фронтендом і бекендом. Така модель є виправданою для системи обліку робочих годин, оскільки вона працює з динамічними станами: змінами, сповіщеннями, оновленнями статусів і службовими подіями, які бажано відображати без ручного перезавантаження сторінки.

Функціональна роль Socket.IO у проєкті простежується через модуль сповіщень. У NotificationsService після створення нового запису в таблиці notification викликається eventsGateway.emitToUser(userId, 'new_notification', notification), тобто сервер передає адресну подію конкретному користувачеві. У NotificationsModule EventsGateway винесено в окремий провайдер і використано разом із сервісом сповіщень, а в структурі серверного каталогу наявний окремий каталог events з файлами events.gateway.ts та events.module.ts.

Це означає, що подієвий шар не є абстрактною можливістю, а включений у логіку опрацювання повідомлень.

Для веб-застосунку обліку робочих годин така інтеграція має прикладне значення. Якщо адміністратор створює або змінює запис, а працівник повинен побачити нове службове повідомлення або оновлення статусу без затримки, подієва модель взаємодії є технічно виправданою. Socket.IO у такій системі виконує функцію оперативного каналу доставлення подій, тоді як основне збереження інформації все одно здійснюється через серверну логіку й базу даних. Саме тому його варто розглядати як додаткову технологію, що підсилює зручність використання інтерфейсу, а не замінює основні механізми обробки даних.

2.5.2 Telegram-інтеграція як додатковий канал взаємодії

Telegram-інтеграція у цьому проєкті реалізована як окремий канал взаємодії із системою, винесений за межі браузерного інтерфейсу. На рівні головного модуля серверної частини підключено TelegrafModule.forRoot із

використанням токена TELEGRAM_BOT_TOKEN, а також окремий TelegramModule. У складі цього модуля визначено TelegramService і TelegramUpdate, що дає підстави розглядати інтеграцію не як формальне розширення, а як самостійний функціональний компонент серверної архітектури.

Практична роль Telegram-сервісу полягає у відправленні повідомлень користувачам, якщо їхній telegramId вже збережено в системі. У TelegramService реалізовано метод sendMessage, який використовує бот Telegraf для надсилання тексту з параметром parse_mode: 'HTML'. Цей сервіс безпосередньо залучений у NotificationsService: після створення запису сповіщення система не лише зберігає його в базі даних і передає через подієвий механізм, а й дублює повідомлення в Telegram, якщо в користувача наявний прив’язаний Telegram-ідентифікатор.

Отже, Telegram-інтеграція використовується як додатковий позавебовий канал доставлення службової інформації.

Зміст Telegram-взаємодії не обмежується пасивним отриманням сповіщень. У TelegramUpdate реалізовано обробку команди /start, приймання шестизначного коду авторизації, пошук цього коду в Redis за ключем telegram-code:<code>, оновлення поля telegramId у записі користувача та видалення використаного коду з Redis. Після успішного зв’язування акаунта користувач отримує клавіатуру з командою початку зміни. Далі бот перевіряє наявність користувача, шукає активну зміну, звертається до розкладу на поточну дату і, якщо зміна запланована, створює новий запис workShift зі статусом PENDING і поточним часом початку. Це свідчить, що Telegram у межах проєкту виконує не лише інформаційну, а й операційну функцію, пов’язану з робочими змінами.

Для системи обліку робочих годин така інтеграція є змістовно виправданою. Вона розширює межі взаємодії з користувачем за рахунок винесення частини сценаріїв у мобільне повідомлювальне середовище, яке не потребує постійного відкриття веб-застосунку. Це особливо доречно для працівників, які мають швидко отримувати службову інформацію або виконувати окремі дії, пов’язані зі зміною, без переходу до браузерного інтерфейсу. У

результаті Telegram-інтеграція підсилює прикладну цінність системи і створює додатковий рівень доступності функцій для кінцевих користувачів.

3 ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНКУ

3.1 Проєктування архітектури системи

Архітектура веб-застосунку «Система обліку робочих годин співробітників малого бізнесу» побудована за розподіленою клієнт-серверною моделлю, у межах якої окремо виділено клієнтську частину, серверну частину, реляційну базу даних, кешувально-сесійний шар і додаткові канали взаємодії. Такий підхід відповідає фактичній структурі репозиторію, де клієнтський застосунок винесено в каталог work-core-client, серверну частину - в workCore-server, а інфраструктурний запуск організовано через docker-compose.yml з окремими сервісами client, server, postgres і redis. Саме така побудова дає змогу розмежувати інтерфейсний рівень, прикладну бізнес-логіку, збереження даних і службові механізми доступу.

Клієнтський рівень архітектури реалізовано на основі Next.js 15, React 19 і TypeScript. Через нього користувач взаємодіє з інформаційною панеллю, сторінками співробітників, підрозділів, профілю, графіка роботи та налаштувань. Структура маршрутизації в каталозі src/app/(isAuth) містить сторінки dashboard, departments, employees, profile, schedule і settings, що підтверджує поділ інтерфейсу на функціональні розділи, пов’язані з основними сутностями системи. Клієнтська частина також містить middleware, який перевіряє сесію користувача, перенаправляє між відкритими та захищеними маршрутами і враховує обмеження доступу залежно від типу пристрою.

Серверний рівень побудовано на NestJS і організовано як модульну систему. У головному модулі застосунку підключено AuthModule, UserModule, DepartmentModule, WorkShiftModule, WorkScheduleModule, WorkShiftTagModule, PrismaModule і TelegramModule, а також глобальний ConfigModule. Це означає, що прикладна логіка не зосереджена в одному монолітному блоці, а розподілена

між окремими підсистемами, кожна з яких відповідає за власну частину предметної області.

Рівень збереження даних реалізовано через PostgreSQL і Prisma ORM. У схемі бази даних визначено моделі User, Department, WorkShift, Tag, WorkSchedule, WorkScheduleLock і Notification, а також переліки Role і ShiftStatus. Така структура демонструє, що архітектура системи орієнтована не на абстрактний облік часу, а на чітко формалізовану предметну модель, у якій користувачі пов'язані зі змінами, графіками і повідомленнями, підрозділи - із графіками, змінами та блокуваннями розкладу, а окремі зміни мають часові параметри і статус опрацювання. Це забезпечує цілісність організаційних даних і дає змогу підтримувати розмежування між плановим розкладом і фактичними записами про роботу.

Окремий шар архітектури становлять механізми доступу та службової взаємодії. Сесійна модель підтримується через Redis, який підключено як окремий сервіс у Docker Compose, а на серверному рівні - через залежності ioredis, connect-redis і express-session. Поряд із цим у проєкті використано подієвий механізм на основі Socket.IO та Telegram-інтеграцію. Серверний сервіс сповіщень після створення повідомлення зберігає його в базі, передає користувачеві через подію new_notification і, за наявності telegramId, дублює його в Telegram. Отже, архітектура включає не лише стандартну взаємодію клієнта з REST- або HTTP-рівнем, а й окремі канали подій і зовнішнього інформування.

З погляду логіки функціонування система має такий цикл роботи. Користувач проходить автентифікацію та взаємодіє з клієнтським інтерфейсом. Через сервісний шар клієнтської частини запити передаються до серверних модулів, де виконується перевірка доступу, прикладна обробка і взаємодія з базою даних через Prisma. Після цього результати повертаються в клієнтський застосунок для відображення на відповідних сторінках. Якщо в процесі роботи виникає подія, що потребує інформування користувача, сервер може сформувати запис у системі сповіщень, передати його через WebSocket-канал або

продублювати через Telegram-бота. Така схема поєднує централізоване збереження даних, модульну серверну обробку і багатоканальну взаємодію з користувачем.

Архітектуру системи наведено на рис. 3.1.

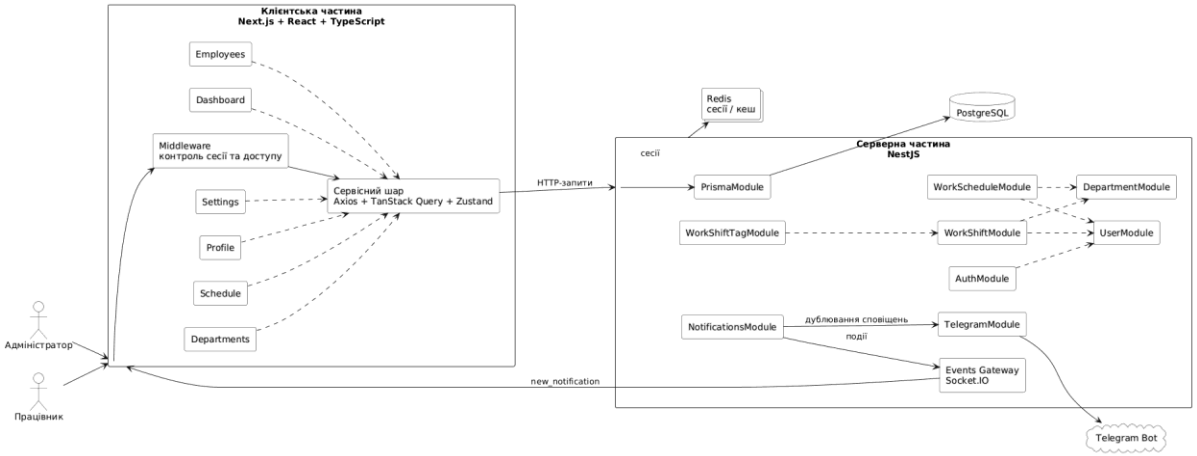

Рис. 3.1 Архітектура веб-застосунку «Система обліку робочих годин співробітників малого бізнесу»

Запропонована схема відображає фактичну логіку побудови проєкту: розмежування клієнтської і серверної частин, модульну організацію серверного рівня, використання PostgreSQL як основного сховища даних, Redis як сесійного шару, а також подієву й Telegram-взаємодію як додаткові канали комунікації. Така архітектура є придатною для системи обліку робочих годин, оскільки забезпечує структурованість коду, узгоджене збереження даних і можливість подальшого розширення функціональності без руйнування базової моделі системи.

3.2 Моделювання вимог до веб-застосунку

Моделювання вимог до веб-застосунку доцільно виконати у формі діаграми варіантів використання, оскільки такий підхід дає змогу формалізувати основні сценарії взаємодії користувачів із системою та встановити межі функціональності, що підлягає подальшому проєктуванню. Для системи обліку

робочих годин це має практичне значення, адже в ній чітко розмежовуються ролі адміністратора і працівника, а сама функціональність охоплює автентифікацію, роботу з працівниками, підрозділами, графіками, змінами, профілем, сповіщеннями та додатковим Telegram-каналом взаємодії. Така логіка узгоджується з побудовою репозиторію, де виділено окремі клієнтську і серверну частини, а в серверному шарі наявні модулі auth, user, department, work.schedule, work.shift, notifications і telegram.

Основними акторами системи є адміністратор і працівник. Адміністратор виконує повний набір операцій, пов’язаних із підтриманням внутрішньої організації праці: входить до системи, переглядає інформаційну панель, керує співробітниками, підрозділами, графіками роботи, змінами, а також працює з власним профілем і сповіщеннями. Працівник використовує систему у вужчому сценарії: проходить автентифікацію, переглядає власний графік і призначені зміни, працює з профілем та отримує повідомлення. Окремим зовнішнім актором можна виділити Telegram-користувача, оскільки серверна частина містить Telegram-модуль і підтримує інтеграцію бота як додатковий канал взаємодії із системою.

Модель вимог повинна відображати не лише перелік функцій, а й підпорядкованість сценаріїв. Усі основні дії в межах захищеної частини системи залежать від проходження автентифікації. Після входу адміністратор переходить до сценаріїв керування структурою підприємства і часовими записами, тоді як працівник взаємодіє переважно з персоналізованими даними. Такий розподіл узгоджується і з клієнтською частиною проєкту, де застосовано Next.js, React, TypeScript, а також окремий шар сервісів для взаємодії з сервером; у складі залежностей клієнта присутні Axios, Zustand, TanStack Query і socket.io-client, що підтверджує реалізацію інтерфейсу як окремого рівня доступу до функцій системи. Структуровану модель функціональних і нефункціональних вимог до веб-застосунку наведено на рис. 3.2.

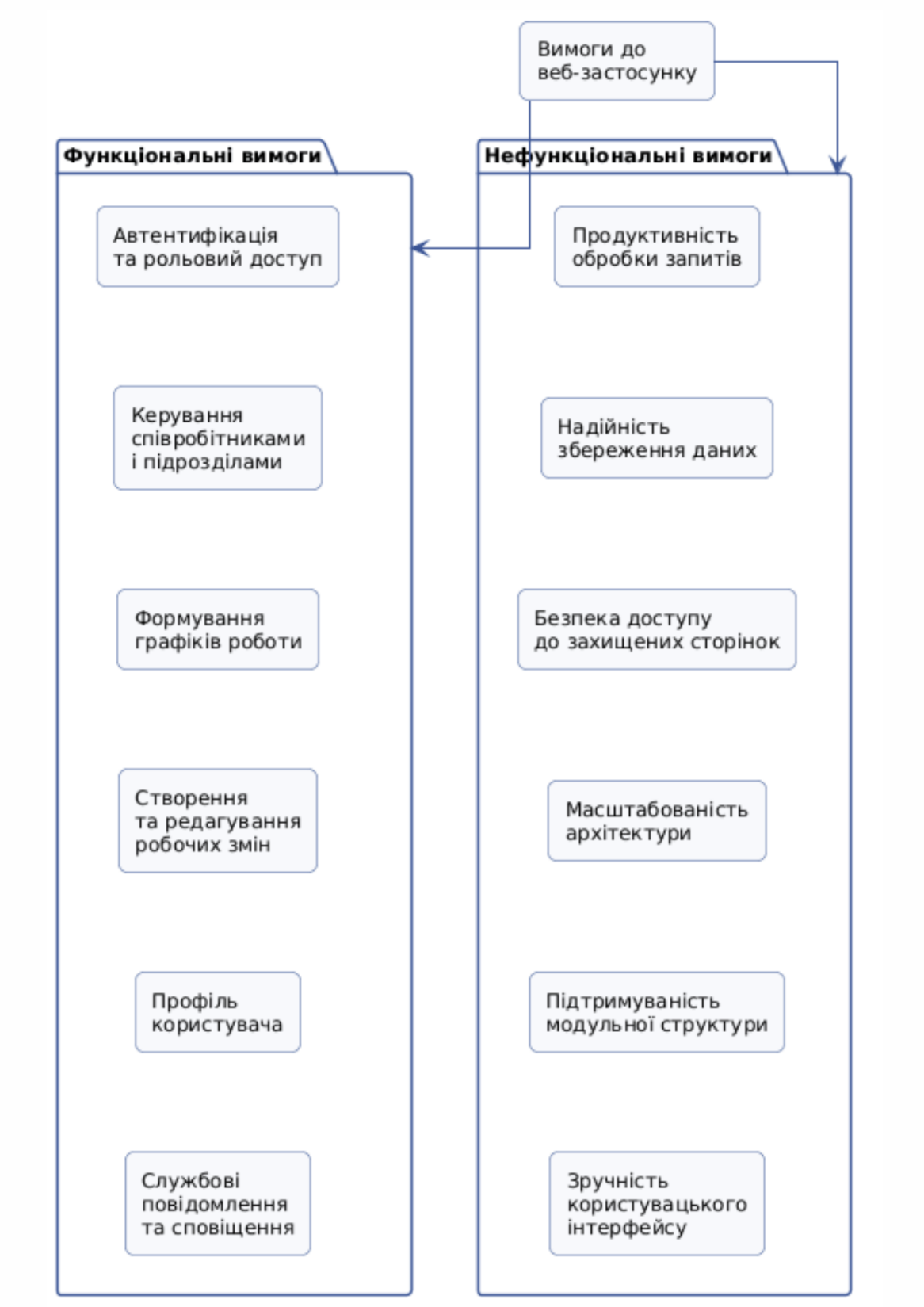

Рис. 3.2 Структура вимог до веб-застосунку

3.3 Проєктування ролей користувачів і сценаріїв використання

Проєктування ролей користувачів і сценаріїв використання визначає спосіб взаємодії людини із системою та задає межі доступу до її функціональності. Для веб-застосунку обліку робочих годин така частина проєктування має принципове значення, оскільки система працює з персональними даними, графіками роботи, змінами та службовими повідомленнями. У поточній реалізації рольова модель закладена на рівні серверної схеми даних через перелік Role, який містить два значення: Employe і Admin. Це означає, що архітектура системи спирається на дві базові внутрішні ролі - працівника та адміністратора.

Адміністратор є користувачем із розширеним набором повноважень. Його сценарії використання охоплюють доступ до захищеної частини веб-застосунку після проходження автентифікації, перегляд інформаційної панелі, роботу зі співробітниками, підрозділами, графіками роботи та змінами. Така модель випливає як із серверної модульної структури, де виділено AuthModule, UserModule, DepartmentModule, WorkShiftModule і WorkScheduleModule, так і з клієнтської маршрутизації, де для захищених сторінок передбачено dashboard, employees, profile, schedule і departments. Отже, адміністратор у спроєктованій системі виступає основним суб’єктом організації робочого часу та керування внутрішніми сутностями застосунку.

Працівник є користувачем із обмеженішим функціональним профілем. Його сценарії зосереджені на вході до системи, перегляді власної інформації, роботі з профілем, ознайомленні з графіком і змінами, а також отриманні сповіщень. На рівні клієнтського доступу ця логіка підкріплюється middleware, який перевіряє наявність сесії та допускає користувача до захищених маршрутів лише після успішної автентифікації. На рівні серверної моделі це узгоджується з тим, що користувач пов’язаний із власними записами workShift, workSchedule і notifications. Таким чином, сценарії працівника мають персоналізований характер і не передбачають повного адміністративного керування системою.

Окремо в архітектурі слід врахувати Telegram як додатковий канал взаємодії з працівником. Формально це не окрема внутрішня роль у схемі Role, але це окремий сценарний контур доступу до функцій системи. Серверна частина підтримує прив’язку Telegram-акаунта через шестизначний код, який користувач вводить боту після команди /start. Після успішної прив’язки бот дозволяє виконувати сценарії початку та завершення зміни. У разі натискання команди «Почати зміну» сервер перевіряє наявність користувача, запланованої зміни на поточну дату та створює запис workShift зі статусом PENDING. У разі команди «Закінчити зміну» виконується пошук активної зміни, обчислюється відпрацьований час і оновлюються поля endTime та totalHours. Це означає, що в проєктованій моделі сценарії використання не обмежуються браузерним інтерфейсом і частково винесені в зовнішнє повідомлювальне середовище.

З погляду логіки використання всі основні сценарії системи мають спільну точку входу - автентифікацію. Після неї система розподіляє можливості залежно від ролі користувача. Адміністратор переходить до сценаріїв організації структури підприємства та планування роботи персоналу. Працівник працює з персональними даними і власними записами. Telegram-сценарії є похідними від попередньої прив’язки акаунта й орієнтовані насамперед на оперативну роботу зі зміною. Саме така побудова забезпечує відповідність між предметною моделлю, механізмами доступу та фактичними функціями реалізованого проєкту.

Проєктування ролей користувачів і сценаріїв використання наведено на рис. 3.3.

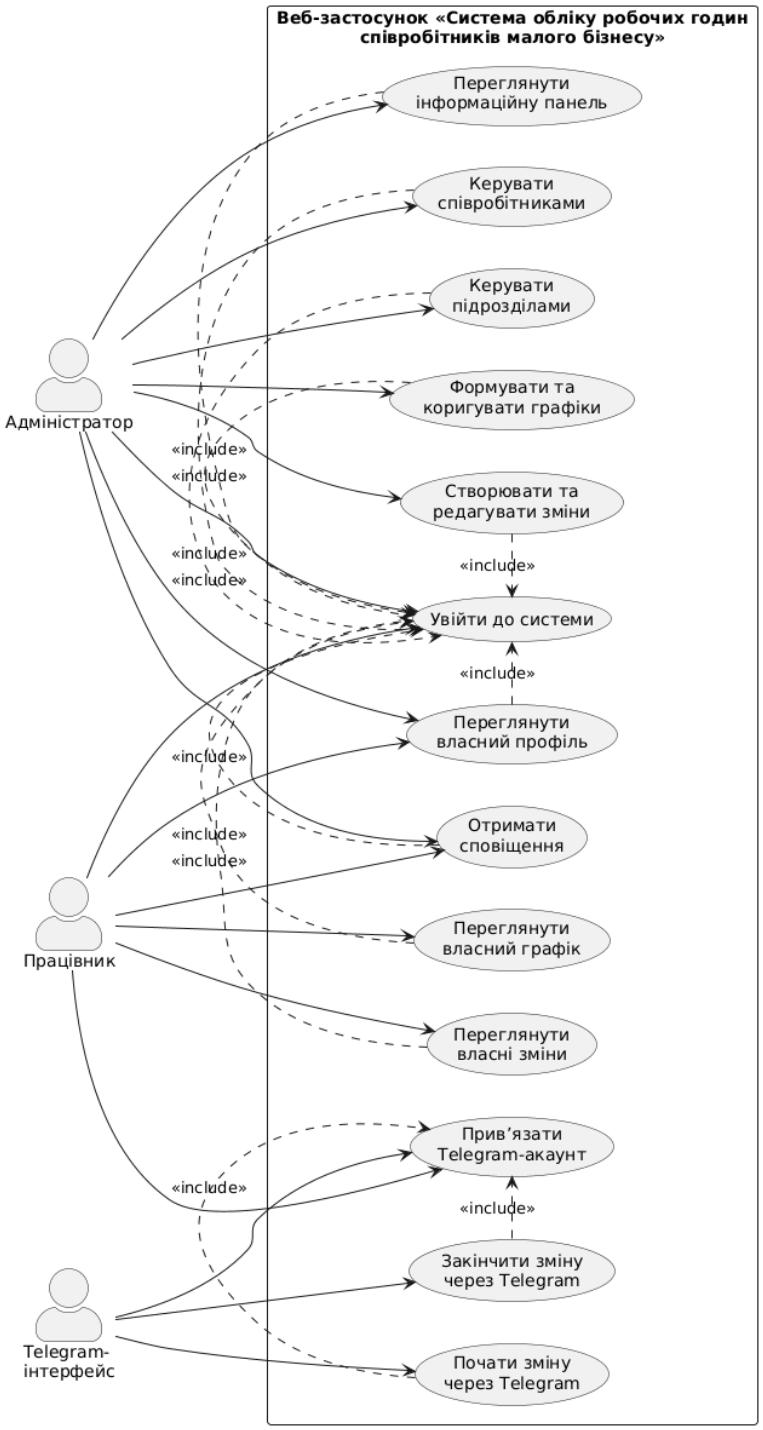

Рис. 3.3 Діаграма ролей користувачів і сценаріїв використання веб-застосунку

Подана схема відображає дворівневу внутрішню рольову модель системи та додатковий зовнішній сценарний контур через Telegram. Вона показує, що адміністратор працює з організаційними сутностями системи, працівник - із персоналізованими даними та змінами, а Telegram-інтеграція підтримує

оперативні дії, пов’язані з початком і завершенням зміни. Така модель є узгодженою з фактично реалізованою архітектурою проєкту.

3.3 Проєктування структури бази даних

Структура бази даних веб-застосунку спроєктована відповідно до логіки предметної області, у якій центральне місце займають користувачі, підрозділи, робочі зміни, графіки роботи та службові повідомлення. У проєкті використано реляційну модель, формалізовану в Prisma-схемі для PostgreSQL. Такий підхід дає змогу підтримувати цілісність зв’язків між сутностями, однозначно визначати ролі користувачів, фіксувати часові параметри роботи та розмежовувати планові й фактичні записи.

Базовою сутністю є User. Вона містить персональні дані користувача, зокрема ім’я, прізвище, електронну пошту, телефон, адресу, хеш пароля, дату народження, аватар і роль. Поле role реалізовано через перелік Role зі значеннями Employe і Admin, тому модель користувача одночасно виконує функцію ідентифікації особи та визначення рівня доступу до системи. Із користувачем пов’язані масиви workShift, workSchedule і notifications, що відображає його участь у змінах, графіках і механізмі сповіщень.

Сутність Department репрезентує підрозділ або окрему локацію малого бізнесу. У ній зберігаються назва, службовий telegramId, час відкриття й закриття у будні та вихідні дні, адреса, координати та ознака активності. Через зв’язки shiftDepartments, scheduleDepartments і scheduleLocks ця сутність інтегрується в усі процеси, пов’язані зі змінами, графіками та блокуванням календарного планування. Така модель є придатною для підприємств, у яких облік робочого часу ведеться не лише за працівниками, а й у межах окремих структурних одиниць.

Фактичний облік робочого часу реалізовано через модель WorkShift. Вона містить посилання на користувача і підрозділ, дату зміни, час початку, час завершення, загальну кількість відпрацьованих годин і статус. Статус зміни

задається переліком ShiftStatus, який містить значення PENDING, APPROVED і REJECTED.

Окремо модель WorkShift пов'язана з сутністю Tag через відношення ShiftTags, що дає змогу маркувати зміни за певними ознаками. Така побудова дозволяє трактувати зміну не як простий часовий запис, а як об'єкт адміністративного опрацювання з додатковими атрибутами класифікації.

Плановий рівень організації робочого часу винесено в модель WorkSchedule. Вона також пов’язана з користувачем і підрозділом, містить дату, часові межі та ознаку вихідного дня isDayOff. Поділ між WorkSchedule і WorkShift є змістовно виправданим, оскільки графік відображає заплановану зайнятість, тоді як зміна фіксує конкретний робочий запис зі статусом і підсумковою тривалістю. Для керування календарною стабільністю розкладу введено окрему сутність WorkScheduleLock, у якій зберігаються посилання на підрозділ, початок тижня та ознака блокування. Для пари departmentId і weekStart установлено унікальність, що унеможливлює дублювання блокувань одного й того самого періоду.

Механізм службового інформування реалізовано через модель Notification. У ній зберігаються посилання на користувача, заголовок, текст повідомлення, тип повідомлення, ознака прочитання і час створення. Тип визначається через перелік NotificationType зі значеннями INFO, WARNING, SUCCESS і ERROR. Зв’язок Notification із User налаштовано з каскадним видаленням, тому видалення користувача автоматично узгоджується з пов’язаними повідомленнями. Така модель підтримує не лише збереження службових записів, а й подальшу інтеграцію зі сценаріями подієвого інформування.

Отже, структура бази даних побудована як узгоджена реляційна модель, у якій кожна ключова сутність предметної області має окреме формалізоване відображення. Вона забезпечує розмежування між користувачами, підрозділами, графіками, змінами, блокуванням розкладу, тегами й повідомленнями, що створює коректну основу для подальшої реалізації серверної логіки та клієнтського інтерфейсу.

Структуру бази даних наведено на рис. 3.4.

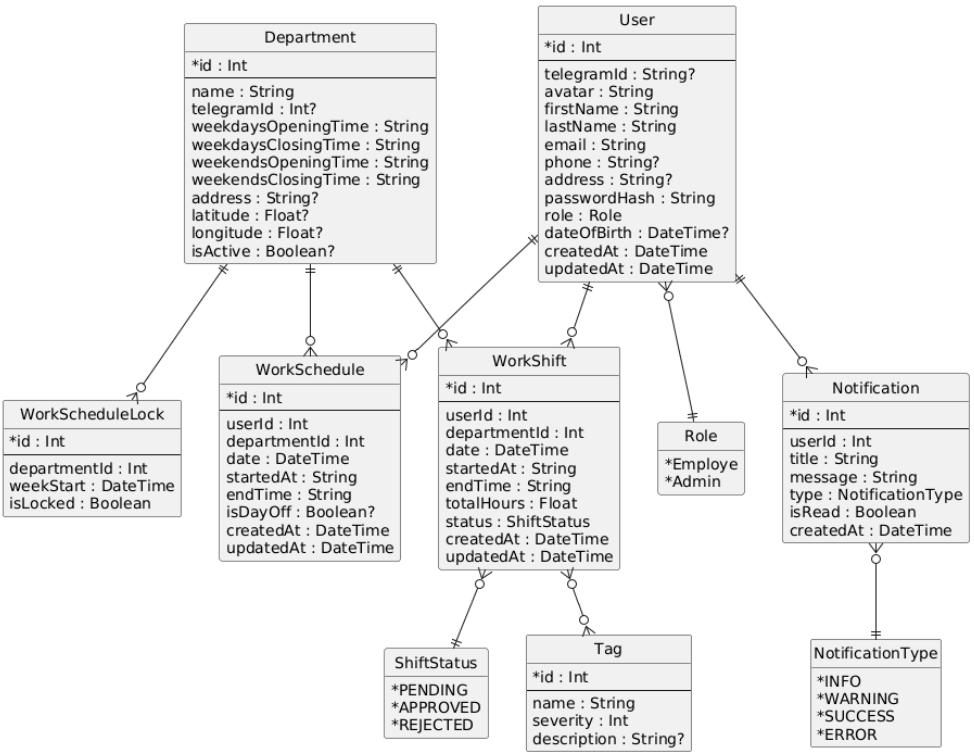

Рис. 3.4 Структура бази даних веб-застосунку «Система обліку робочих годин співробітників малого бізнесу»

3.5 Проєктування серверних модулів системи

Проєктування серверних модулів веб-застосунку ґрунтується на модульній архітектурі NestJS, у межах якої функціональність розподілено між окремими підсистемами відповідно до предметної логіки обліку робочого часу. У кореневому серверному каталозі src виділено модулі auth, department, events, file.storage, notifications, prisma, telegram, user, work.schedule, work.shift і work.shift.tag, а в AppModule підключено PrismaModule, AuthModule, UserModule, DepartmentModule, WorkShiftModule, WorkScheduleModule, WorkShiftTagModule, глобальний ConfigModule, а також TelegramModule через TelegrafModule.forRoot(...). Це свідчить, що серверна частина спроєктована як сукупність взаємопов’язаних модулів, а не як єдиний монолітний блок логіки.

Базовим інфраструктурним модулем системи є PrismaModule, який надає PrismaService для взаємодії з базою даних. Через нього серверні сервіси отримують доступ до моделей користувачів, підрозділів, змін, графіків, тегів і повідомлень. Саме тому в архітектурі серверних модулів PrismaModule доцільно розглядати як загальний шар доступу до даних, на який спираються прикладні підсистеми.

AuthModule виконує функцію автентифікації та побудований із залежністю від UserModule. Така залежність є логічною, оскільки перевірка облікових даних, визначення ролі користувача та подальше формування сесійного доступу безпосередньо пов’язані з відомостями про користувача. Проєктне рішення, за якого модуль автентифікації не працює ізольовано, а спирається на модуль користувачів, забезпечує узгодженість між механізмом входу до системи та предметною моделлю доступу.

UserModule є центральним прикладним модулем серверної частини, оскільки через нього проходить робота з даними користувачів, завантаженням файлів і Redis-клієнтом. У цьому модулі підключено ConfigModule, MulterModule.registerAsync(...), PrismaService, FileStorageService і провайдер REDIS_CLIENT, який створюється через ioredis. Таке проєктування показує, що модуль користувачів виконує ширшу функцію, ніж простий доступ до записів User: він також забезпечує інфраструктуру для файлових операцій і допоміжних механізмів, потрібних іншим модулям. Саме тому UserModule експортує UserService і REDIS_CLIENT, а інші модулі імпортують його як базову залежність.

DepartmentModule і WorkScheduleModule спроєктовані як модулі, пов’язані з організаційною структурою підприємства та плановим рівнем обліку робочого часу. DepartmentModule імпортує UserModule і використовує PrismaService, UserService та FileStorageService, а WorkScheduleModule також імпортує UserModule і працює через PrismaService, DepartmentService, UserService та FileStorageService. Це означає, що графік роботи в системі не розглядається

ізольовано: він проєктно пов’язаний із користувачами, підрозділами та спільним інфраструктурним шаром.

WorkShiftModule репрезентує модуль фактичних змін і має найбільш виразну інтеграцію з іншими серверними підсистемами. Він імпортує EventsModule, UserModule і NotificationsModule, а серед провайдерів використовує PrismaService, UserService, FileStorageService і DepartmentService. Така побудова демонструє, що робота зі змінами є центральним операційним процесом системи: створення або зміна запису про зміну пов’язується не лише з користувачем і підрозділом, а й із подієвим механізмом та системою сповіщень. Це відповідає предметній логіці застосунку, у якому зміна є не просто часовим записом, а подією, здатною породжувати службову реакцію системи.

WorkShiftTagModule виділено окремо для роботи з тегами змін. Він імпортує PrismaModule і UserModule, містить власні контролер і сервіс та експортує WorkShiftTagService. Виділення цього модуля в окрему підсистему є архітектурно виправданим, оскільки дозволяє не змішувати основну логіку змін із механізмом їх додаткової класифікації. Унаслідок цього серверна структура зберігає розділення між базовим обліком змін і допоміжною семантикою їх позначення.

Подієвий і повідомлювальний контур системи утворюють EventsModule, NotificationsModule і TelegramModule. EventsModule є глобальним і експортує EventsGateway. NotificationsModule також оголошено глобальним; він імпортує PrismaModule, UserModule, EventsGateway і TelegramModule, надає NotificationsService і NotificationsController та експортує сервіс сповіщень. TelegramModule імпортує PrismaModule і UserModule та містить TelegramUpdate і TelegramService. Таке проєктування означає, що система підтримує три рівні повідомлювальної логіки: збереження повідомлень у базі, передавання подій через gateway і дублювання окремих сценаріїв через Telegram.

Отже, серверні модулі системи спроєктовано як взаємопов’язану, але структурно розділену сукупність підсистем. AuthModule відповідає за автентифікацію, UserModule - за роботу з користувачами й допоміжною

інфраструктурою, DepartmentModule - за організаційну структуру, WorkScheduleModule - за графіки, WorkShiftModule - за фактичні зміни, WorkShiftTagModule - за класифікацію змін, NotificationsModule і EventsModule - за подієве інформування, а TelegramModule - за зовнішній канал взаємодії. Така модульна організація є придатною для подальшого розвитку системи, оскільки локалізує відповідальність і спрощує підтримання узгодженості між прикладною логікою та інфраструктурними сервісами.

Структуру серверних модулів наведено на рис. 3.5.

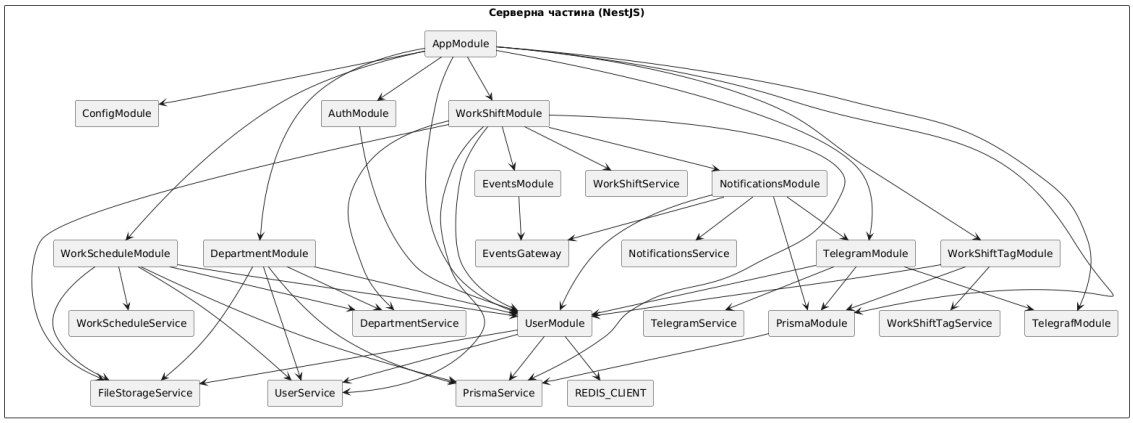

Рис. 3.5 Структура серверних модулів веб-застосунку

3.6 Проєктування клієнтської частини та маршрутизації

Клієнтська частина веб-застосунку спроєктована як окремий фронтенд-рівень, що забезпечує взаємодію користувача з основними функціями системи обліку робочих годин. Технологічну основу цього рівня становлять Next.js 15, React 19 і TypeScript 5, а також набір бібліотек для роботи зі станом, запитами, формами, календарними елементами, модальними вікнами, візуалізацією даних і подієвою взаємодією. Така конфігурація свідчить, що клієнтський рівень проєктувався не як набір ізольованих сторінок, а як узгоджена інтерфейсна система, у якій маршрутизація, доступ до серверних даних, локальний стан і допоміжні механізми взаємодії мають працювати в межах єдиної архітектури.

В основу клієнтської побудови покладено маршрутно-орієнтовану модель, характерну для Next.js. Із middleware випливає, що в системі виокремлено захищену частину з маршрутами /dashboard, /employees, /profile, /schedule і /departments, а також окрему зону /auth для сценаріїв автентифікації. Додатково передбачено маршрут /wip, який використовується для обмеження доступу залежно від типу пристрою. Такий поділ означає, що маршрутизація спроєктована за принципом функціонального розмежування: одна група маршрутів обслуговує вхід до системи, інша - роботу із захищеними даними, а окремий маршрут підтримує службовий режим доступу. Для системи обліку робочого часу це є доцільним, оскільки дозволяє відокремити публічний контур входу від внутрішнього робочого середовища адміністратора і працівника.

Проєктування клієнтської частини спирається на уявлення про різні сценарії користувацької роботи. Інформаційна панель виконує функцію стартового екрана захищеної частини системи, сторінка співробітників призначена для перегляду й адміністрування користувачів, сторінка підрозділів - для роботи зі структурою підприємства, сторінка графіка - для відображення та коригування робочого часу, а сторінка профілю - для персоналізованої взаємодії з даними користувача. На рівні архітектури це означає, що кожен маршрут відповідає окремому функціональному сегменту предметної області. Такий підхід зменшує змішування різнорідної логіки в межах одного інтерфейсного блоку й полегшує подальше розширення системи, оскільки нові сценарії можуть додаватися як окремі маршрутизовані елементи без руйнування чинної структури.

Маршрутизація доповнюється middleware-рівнем, який є ключовим елементом проєктування клієнтського доступу. У middleware реалізовано кілька послідовних перевірок: визначення типу пристрою за user-agent, контроль дозволеного доступу для ПК, планшета або мобільного пристрою через змінні середовища, перенаправлення на /wipy разі блокування типу пристрою, перевірку наявності cookie session, а також логіку розмежування між сторінками автентифікації й захищеними маршрутами. Якщо користувач уже має сесію і

переходить до /auth, система перенаправляє його на інформаційну панель; якщо сесії немає і користувач намагається відкрити захищений маршрут, його перенаправляють на сторінку входу. Це означає, що маршрутизація в проєкті спроєктована не як пасивна навігація між сторінками, а як механізм керованого доступу, тісно пов’язаний із сесійною моделлю безпеки.

Окремої уваги потребує внутрішня організація клієнтської логіки. Наявність у залежностях axios, @tanstack/react-queryi zustand дає підстави розглядати клієнтську частину як систему з чітким поділом між трьома шарами: інтерфейсним, сервісним і станоорієнтованим. Сервісний шар забезпечує звернення до серверних ендпоінтів; React Query виконує синхронізацію даних між сервером і клієнтом; Zustand підтримує локальний стан, пов’язаний із користувачем і рольовою логікою. У проєктувальному вимірі це означає, що маршрути клієнтської частини не повинні безпосередньо виконувати всю прикладну логіку, а мають взаємодіяти з окремими службовими компонентами, які відповідають за отримання, кешування та відображення даних. Така побудова є особливо придатною для системи, у якій одні й ті самі дані - наприклад, відомості про користувача або графік роботи - мають використовуватися в кількох маршрутах і бути узгодженими між собою.

Проєктування клієнтської частини також передбачає підтримку допоміжних сценаріїв взаємодії, які формують повноцінне робоче середовище. Використання react-hook-form вказує на орієнтацію системи на структуровану роботу з формами входу, профілю та записів; react-day-picker - на наявність календарного виміру в інтерфейсі; react-modal - на використання модальних діалогів; recharts - на потенційну візуалізацію даних інформаційної панелі; socket.io-client- на участь клієнта в подієвій взаємодії; sonner і motion - на підтримку інтерактивних повідомлень та анімованої поведінки елементів. Усе це свідчить, що клієнтська архітектура проєктувалася як багатокомпонентна система, у якій маршрути є лише зовнішнім каркасом, а внутрішня поведінка сторінок залежить від низки спеціалізованих бібліотек.

Отже, проєктування клієнтської частини та маршрутизації веб-застосунку побудовано на поєднанні трьох взаємопов’язаних принципів: функціонального розподілу сторінок, керованого доступу через middleware і відокремлення сервісної логіки від інтерфейсного рівня. Next.js визначає загальну структуру маршрутів, middleware контролює допуск до них, а React, TypeScript, Zustand, TanStack Query та Axios формують внутрішню організацію клієнтського середовища. У результаті фронтенд-рівень системи набуває чіткої архітектурної форми, придатної для підтримки ролей користувачів, роботи з динамічними даними та подальшого розширення функціональності без порушення базової логіки маршрутизації.

3.7 Проєктування інтерфейсу користувача

Проєктування інтерфейсу користувача веб-застосунку «Система обліку робочих годин співробітників малого бізнесу» підпорядковане логіці предметної області, у якій більшість дій пов’язана з переглядом, внесенням і коригуванням структурованих даних про працівників, підрозділи, графіки роботи та робочі зміни. Інтерфейс у такій системі не може розглядатися як декоративна оболонка програмного продукту, оскільки саме через нього користувач виконує основні операції адміністрування й контролю. З огляду на це клієнтську частину було спроєктовано так, щоб вона підтримувала чітку навігацію між функціональними розділами, відображала дані у зрозумілому структурованому вигляді та забезпечувала швидкий перехід від перегляду інформації до її редагування або уточнення.

Інтерфейсна модель застосунку спирається на маршрутну організацію клієнтської частини, у межах якої виділено окремі сторінки для інформаційної панелі, співробітників, підрозділів, графіка роботи, профілю та налаштувань. Така побудова відповідає поділу системи на функціональні сегменти й дає змогу користувачеві працювати не з перевантаженим універсальним екраном, а з окремими інтерфейсними просторами, кожен із яких пов’язаний із конкретним

типом дій. Проєктне рішення такого типу є доцільним для малого бізнесу, оскільки адміністративні сценарії вимагають швидкого доступу до різних видів інформації, а працівник, навпаки, потребує компактного інтерфейсу з доступом лише до персонально значущих даних.

Базовим принципом проєктування інтерфейсу стала рольова диференціація. Система орієнтована на дві основні категорії внутрішніх користувачів - адміністратора та працівника, і саме тому інтерфейс має бути придатним до двох різних моделей взаємодії. Для адміністратора ключовими є сценарії керування персоналом, підрозділами, графіками та змінами, отже інтерфейсна структура для цієї ролі повинна забезпечувати швидкий доступ до відповідних розділів, наявність елементів створення, редагування, перегляду та пошуку записів, а також можливість послідовного переходу між пов’язаними об’єктами. Для працівника центральними є сценарії перегляду власного графіка, власних змін, профілю та повідомлень, тому інтерфейс для нього має бути менш насиченим керувальними елементами й зосередженим на отриманні актуальної інформації без надмірного адміністративного функціоналу.

Навігаційна модель інтерфейсу спроєктована за принципом постійної доступності основних розділів. Наявність у структурі клієнтської частини компонентів Sidebar і Header вказує на використання стійкого навігаційного каркаса, у якому ключові маршрути не приховуються всередині багаторівневого меню, а залишаються доступними в межах основного інтерфейсного простору. Для системи обліку робочих годин це є виправданим рішенням, оскільки користувач постійно перемикається між пов’язаними сутностями: від співробітника до графіка, від підрозділу до змін, від профілю до сповіщень. Якщо навігація в такому застосунку побудована надто складно, це безпосередньо підвищує кількість зайвих дій під час щоденної роботи. Тому інтерфейс має підтримувати прямий доступ до основних сторінок і зберігати стабільну логіку розміщення навігаційних елементів.

Інформаційна панель проєктується як стартовий екран захищеної частини системи. Її призначення полягає не у виведенні надлишкової аналітики, а в

поданні зведеної інформації, яка дає користувачеві уявлення про поточний стан системи. У межах такої сторінки доцільно відображати узагальнені показники, пов’язані з кількістю співробітників, наявними змінами, станом графіка чи службовими подіями.

Наявність у клієнтських залежностях бібліотеки recharts свідчить, що інтерфейсна модель допускає використання графічних елементів для подання даних, однак у контексті цього застосунку їхня функція повинна залишатися допоміжною. Основний зміст інформаційної панелі має полягати в швидкому ознайомленні з поточними станами, а не в побудові складного аналітичного кабінету.

Сторінки співробітників і підрозділів проєктуються як робочі інтерфейси табличного або карткового типу, де користувач працює з множиною однорідних записів. Для такого формату принциповими є впорядковане подання даних, наявність дій редагування й перегляду, а також підтримка навігації між сторінками списку. Присутність у структурі клієнтської частини компонента PaginationSlider вказує, що в проєкті передбачено механізм роботи з багатоелементними вибірками. Отже, інтерфейс цих сторінок проєктується не як одноразове відображення всього масиву записів, а як керований список, придатний до поетапного перегляду. Для малого бізнесу це особливо доречно у випадках, коли кількість працівників або структурних одиниць перевищує межу, зручну для одноекранного представлення.

Сторінка графіка роботи має іншу логіку інтерфейсного проєктування, оскільки тут домінує не перелік статичних записів, а часовий вимір. Саме тому для побудови цього розділу доцільно використовувати календарно-табличний підхід, у якому користувач сприймає дані не лише як набір значень, а як розклад, прив’язаний до дат, часових меж і статусів. Використання react-day-picker у клієнтських залежностях підтверджує, що в проєкті закладено календарний компонент, придатний для вибору дат і побудови часових інтерфейсів. Отже, сторінка графіка роботи повинна бути спроєктована так, щоб користувач міг не лише побачити окремий запис, а й зрозуміти його місце в загальному розкладі. У

такому інтерфейсі визначального значення набувають наочність прив’язки до дати, розділення між плановими графіками та фактичними змінами, а також мінімізація візуального перевантаження.

Окремий інтерфейсний блок пов’язаний із формами. У системі такого типу форми використовуються під час автентифікації, редагування профілю, створення або оновлення записів про співробітників, підрозділи, зміни та, ймовірно, графіки роботи. Використання react-hook-form свідчить, що форма розглядається в проєкті як окремий структурований елемент інтерфейсу, а не як набір неузгоджених полів. Це має безпосередній вплив на проєктування: форма повинна бути лаконічною, поділеною на змістові поля, придатною до валідації й достатньо прозорою для користувача, який працює з адміністративними даними в умовах дефіциту часу. У системі обліку робочих годин це особливо актуально, оскільки помилка в часовому полі або в прив’язці зміни до підрозділу може вплинути на подальшу інтерпретацію розкладу.

Для вторинних сценаріїв взаємодії інтерфейс спроєктовано з використанням модальних вікон і повідомлень. Наявність react-modal і sonner дає підстави вважати, що в системі передбачено діалогові форми підтвердження, короткі сценарії редагування та відображення службових повідомлень без переходу на окремий екран. Таке рішення є виправданим для веб-застосунку внутрішнього користування, оскільки значна частина дій не потребує повної зміни контексту сторінки. Наприклад, редагування окремого запису або підтвердження службової дії доцільніше виконувати в модальному інтерфейсі, ніж відкривати новий маршрут. Водночас подібні елементи не повинні перевантажувати користувача й мають використовуватися лише для локальних сценаріїв, де збереження контексту є важливішим за розгортання повноцінного окремого екрану.

Проєктування інтерфейсу враховує також подієву взаємодію та сповіщення. Використання socket.io-client у фронтенді разом із серверною подієвою архітектурою означає, що користувацький інтерфейс повинен бути придатним до отримання нових повідомлень без ручного оновлення сторінки. Це

впливає на композицію інтерфейсу, оскільки в ньому має бути місце для ненав’язливого, але помітного відображення нових подій. У контексті цього застосунку події стосуються насамперед сповіщень, пов’язаних зі змінами, тому інтерфейс повинен підтримувати їхнє миттєве відображення без руйнування основного робочого сценарію.

Отже, проєктування інтерфейсу користувача веб-застосунку ґрунтується на поєднанні рольової диференціації, маршрутної організації, стабільної навігації, календарно-табличного подання часових даних, структурованої роботи з формами, локальних модальних сценаріїв і подієвого відображення сповіщень. Такий підхід відповідає предметній області обліку робочих годин і фактичній клієнтській архітектурі проєкту, у межах якої інтерфейс повинен залишатися зрозумілим для працівника, функціонально достатнім для адміністратора та придатним до подальшого розширення без перебудови базової навігаційної моделі.

3.8 Проєктування механізмів безпеки, перевірки доступу та обмеження ролей

Проєктування механізмів безпеки у веб-застосунку обліку робочих годин підпорядковане вимозі розмежування доступу до персональних і службових даних, захисту автентифікованої взаємодії та контролю сценаріїв використання залежно від ролі користувача. Для цієї системи безпека не обмежується лише перевіркою факту входу до застосунку, оскільки користувач працює з графіками, змінами, профілем, підрозділами та повідомленнями, а отже доступ до різних операцій має бути диференційованим. Архітектурно захисний контур побудовано як поєднання серверної сесійної автентифікації, ролевої моделі в базі даних, клієнтського middleware-контролю маршрутів і додаткових обмежень за типом пристрою.

Основу серверного механізму безпеки становить сесійна модель доступу. У main.ts підключено cookie-parser із секретом, express-session, connect-redis і ioredis, а сам застосунок налаштовує CORS з credentials: true, використовує ValidationPipe з transform і whitelist, а для cookie сесії задає httpOnly, secure, sameSite: 'lax', домен і строк життя. Сесії зберігаються в Redis через RedisStore, що означає винесення сесійного стану за межі процесу застосунку і підвищує стабільність авторизованої взаємодії. Така конфігурація є придатною для внутрішньої інформаційної системи, де доступ до захищених маршрутів має визначатися не локальним станом браузера, а серверним підтвердженням наявності чинної сесії. Використання глобальної валідації з whitelist додатково зменшує ризик передавання на сервер неочікуваних полів у вхідних запитах.

Клієнтський рівень перевірки доступу реалізовано через middleware.ts. У ньому спочатку виконується визначення типу пристрою на основі user-agent, після чого перевіряються прапорці ACCESS_PC_ENABLED, ACCESS_TABLET_ENABLED і ACCESS_MOBILE_ENABLED.

Якщо доступ для поточного типу пристрою заборонений, користувача перенаправляють на /wip; якщо заборони немає, але користувач уже перебуває на /wip, його повертають на інформаційну панель. Після цього middleware перевіряє наявність cookie session: для маршруту /authавторизований користувач перенаправляється на /dashboard, а для захищених маршрутів за відсутності сесії виконується перенаправлення на сторінку входу. Отже, маршрутизація спроєктована як механізм ранньої перевірки доступу, що відсікає неавторизовані переходи ще до рендерингу захищених сторінок.

Ролеве розмежування в системі закладено на рівні моделі даних. У Prisma-схемі користувач має поле role, а перелік Role містить два значення: Employe і Admin. Це означає, що в архітектурі передбачено базове двохрівневе розмежування доступу між працівником і адміністратором.

Працівник пов’язаний із власними записами WorkShift, WorkSchedule і Notification, тоді як адміністратор у прикладній логіці системи отримує доступ до ширшого набору операцій, пов’язаних із керуванням користувачами,

підрозділами, графіками та змінами. Такий підхід є змістовно виправданим для предметної області, оскільки роль виступає не додатковим атрибутом профілю, а одним із механізмів організації дозволених сценаріїв використання системи.

На клієнтському рівні рольова модель підтримується через Zustand-сховище user.store.ts. У ньому зберігається об’єкт поточного користувача та ознака isAdmin, яка визначається за значенням user.role === 'Admin'; стан зберігається у localStorage під ключем workcore-user-storage. Така побудова придатна для адаптації інтерфейсу до типу користувача: відображення доступних елементів меню, окремих кнопок, сторінок і допоміжних сценаріїв може виконуватися без повторного обчислення ролі в кожному компоненті. Водночас проєктувально важливо, що клієнтське сховище саме по собі не слугує остаточним механізмом захисту. Його функція полягає у відтворенні інтерфейсного стану, тоді як справжня авторизація має спиратися на серверну сесію і серверну бізнес-логіку. Саме такий розподіл відповідальності є коректним для веб-застосунку, у якому клієнтська частина повинна бути зручною, але не визначальною для безпекового контуру.

Окремого розгляду потребує захист облікових даних. У серверних залежностях наявний bcrypt, що вказує на використання хешування паролів замість збереження відкритих значень, а в моделі User визначено поле passwordHash, а не поле відкритого пароля. Це означає, що архітектура автентифікації спроєктована з урахуванням базової вимоги до захисту облікових записів: навіть у разі доступу до бази даних пароль не повинен зберігатися у читаному вигляді. Для внутрішньої системи обліку робочого часу таке рішення є обов’язковим, оскільки вона оперує персональними профілями працівників і доступом до службової інформації.

Додатковим елементом контролю доступу виступає Telegram-інтеграція, хоча вона не замінює основну веб-автентифікацію. У моделі User передбачено унікальний telegramId, а серверний стек містить nestjs-telegraf і telegraf, що вказує на зв’язування профілю користувача із зовнішнім каналом взаємодії. Проєктувально це означає, що доступ через Telegram повинен спиратися на

попередню прив’язку акаунта до вже існуючого користувача системи, а не створювати паралельний автономний контур ідентифікації. Такий підхід зберігає єдність моделі користувача і не допускає розриву між веб-роллю, сесійною моделлю та додатковими повідомлювальними сценаріями.

Отже, механізми безпеки, перевірки доступу та обмеження ролей у веб-застосунку спроєктовано як багаторівневу систему. Серверний рівень забезпечує сесії, збережені в Redis, cookie-параметри безпеки, валідацію вхідних даних і захист облікових записів через хешування паролів. Клієнтський рівень реалізує middleware-контроль маршрутів, обмеження за типом пристрою і рольову адаптацію інтерфейсу. Реляційна модель даних формалізує самі ролі, а додатковий Telegram-канал інтегрується в систему без руйнування її основної логіки доступу. Саме така архітектурна побудова є придатною для системи обліку робочих годин, у якій потрібно одночасно забезпечити зручність користування, захист персональних даних і контроль дозволених дій для різних категорій користувачів.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ

4.1 Реалізація клієнтської частини веб-застосунку

Клієнтська частина веб-застосунку реалізована як окремий фронтенд-рівень на основі Next.js 15, React 19 і TypeScript 5. У складі залежностей використано також Zustand для локального стану, TanStack React Query для синхронізації клієнта із серверними даними, Axios для HTTP-взаємодії, а також react-hook-form, react-day-picker, react-modal, recharts, socket.io-client, sass і sonner. Така комбінація технологій забезпечила побудову інтерфейсу, у якому поєднуються маршрутизація, форми, календарні елементи, сторінки керування сутностями та відображення службових повідомлень.

У межах реалізації клієнтської частини було сформовано захищений інтерфейс із функціональними маршрутами для інформаційної панелі, співробітників, профілю, графіка роботи та підрозділів. Окремо реалізовано контур автентифікації, а в middleware - перевірку наявності сесії, перенаправлення між сторінками входу і внутрішніми сторінками системи, а також механізм контролю доступу залежно від типу пристрою. Такий підхід дав змогу зробити клієнтський застосунок не просто набором сторінок, а цілісною інтерфейсною системою з контрольованою логікою доступу. Структуру реалізації клієнтської частини веб-застосунку наведено на рис. 4.1.

Сервісний шар клієнта побудовано як проміжний рівень між сторінками інтерфейсу та серверними ендпоінтами. Завдяки цьому отримання даних про працівників, підрозділи, зміни, графіки та профіль користувача не змішується безпосередньо з візуальною логікою компонентів. Така реалізація спрощує підтримання коду, підвищує керованість запитів і дає змогу повторно використовувати однакові сценарії звернення до серверної частини в різних екранних формах. Структуру реалізації серверної частини веб-застосунку наведено на рис. 4.2.

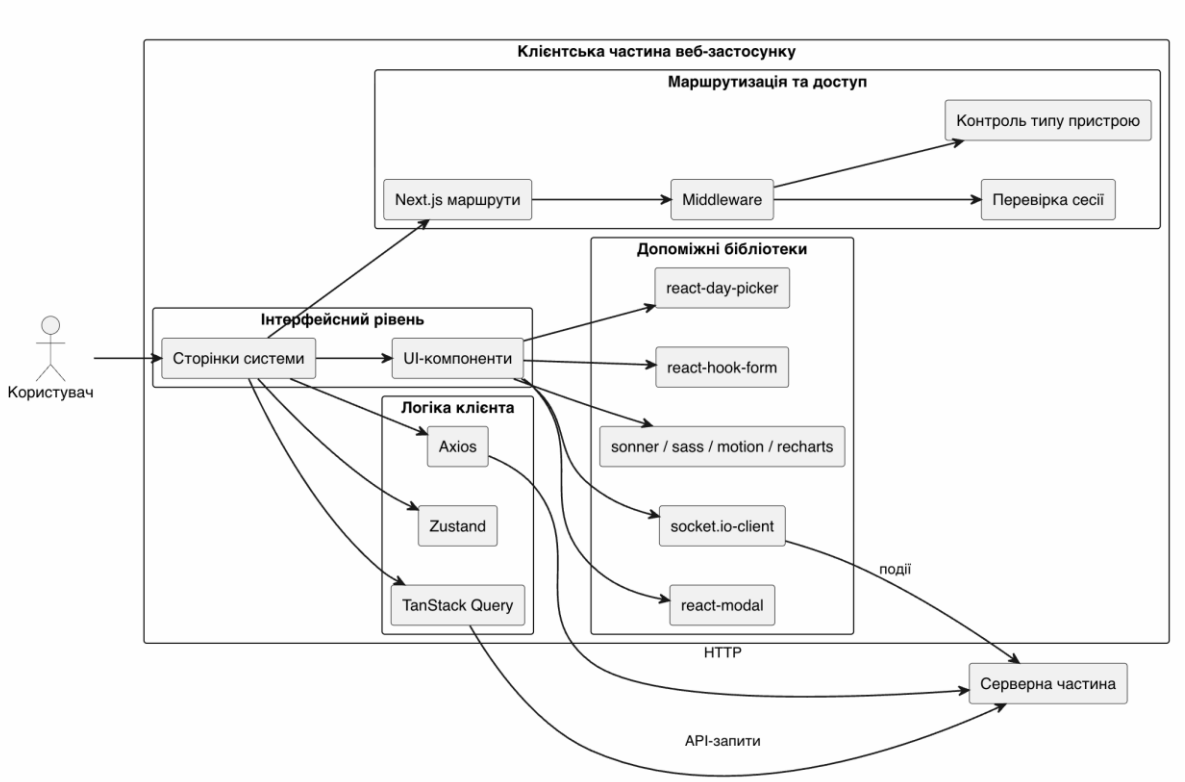

Рис. 4.1 Структура клієнтської частини веб-застосунку

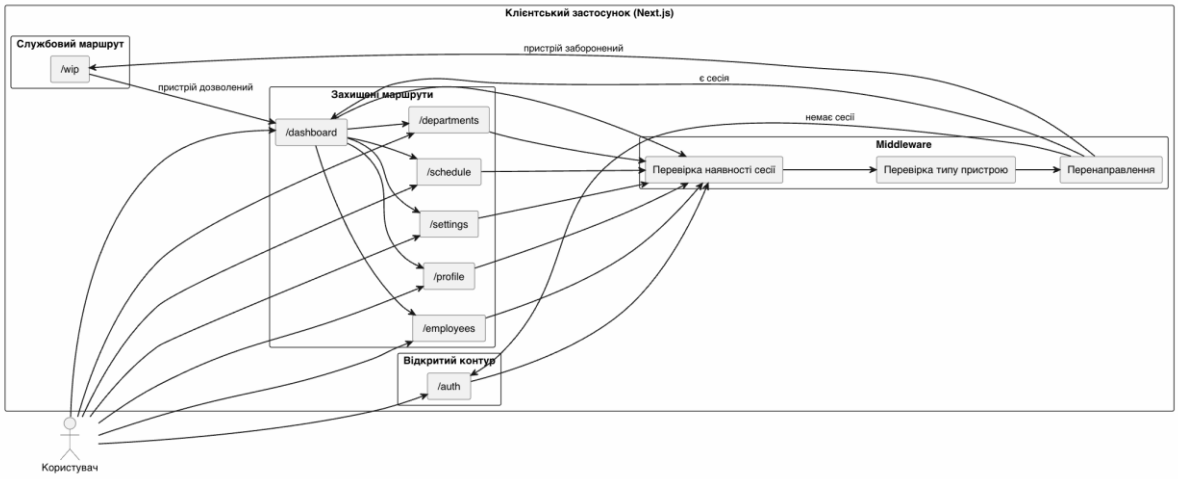

Рис. 4.2 Маршрутна організація клієнтського застосунку

4.2 Реалізація серверної частини веб-застосунку

Серверну частину реалізовано на основі NestJS. У проєкті підключено модулі для автентифікації, користувачів, підрозділів, змін, графіків, тегів змін, а також інфраструктурні компоненти для конфігурації, WebSocket-взаємодії та

Telegram-інтеграції. Така модульна організація дозволила розділити прикладну логіку за предметними сутностями і побудувати сервер як набір взаємопов’язаних, але структурно відокремлених підсистем.

На рівні прикладної реалізації сервер виконує обробку бізнес-логіки, перевірку прав доступу, взаємодію з базою даних і формування відповідей для клієнтської частини. У такій моделі клієнт не працює безпосередньо з даними, а звертається до серверних модулів, які забезпечують валідацію, узгодження зв’язків між сутностями та контроль за коректністю операцій. Це створює стабільну серверну основу для роботи з персоналом, графіками та змінами. Фрагмент реалізації модуля автентифікації користувачів наведено на рис. 4.3.

Суттєвою ознакою серверної реалізації є підтримка інфраструктурного поділу між прикладною логікою, збереженням даних і сесійним шаром. Для цього застосовано PostgreSQL як основне сховище структурованих даних і Redis як компонент підтримки сесій та допоміжної серверної взаємодії. Така побудова підвищує узгодженість системи й робить її придатною до розгортання в контейнеризованому середовищі. Фрагмент реалізації модулів керування підрозділами, робочими змінами та графіками роботи наведено на рис. 4.4.

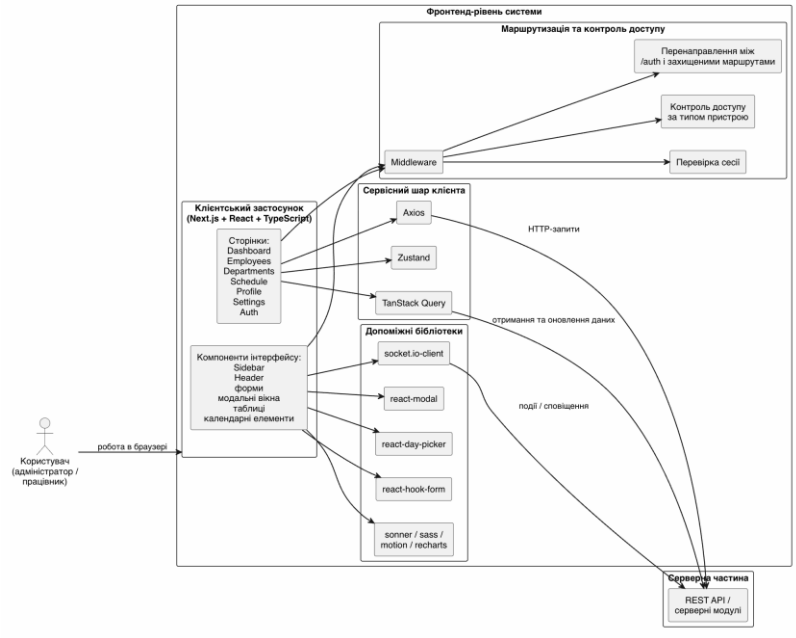

Рис. 4.3 Структура серверної частини веб-застосунку

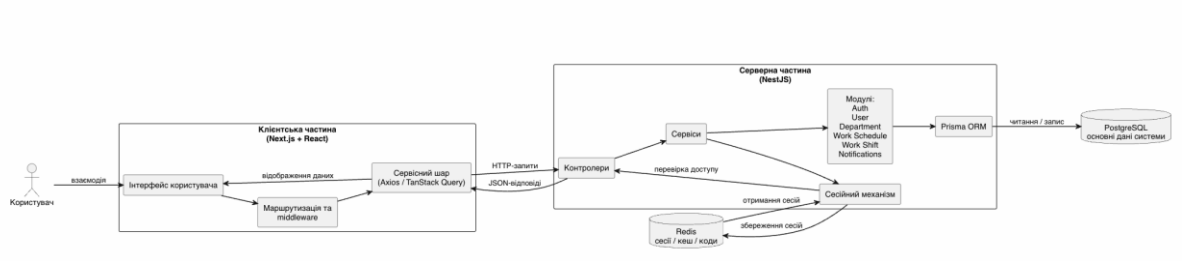

Рис. 4.4 Логіка взаємодії клієнта, сервера, бази даних і Redis

4.3 Реалізація модулів автентифікації та керування користувачами

Автентифікація в застосунку реалізована на основі сесійної моделі доступу. На сервері використано express-session, cookie-parser, connect-redis і ioredis, а на клієнтському рівні доступ до захищених маршрутів перевіряється через middleware за наявністю cookie session. У результаті користувач після входу працює не в межах одноразового локального стану браузера, а в моделі, де право доступу підтверджується серверною сесією.

Механізм керування користувачами спирається на модель User, що містить персональні дані, passwordHash, telegramId, контактні поля та роль. Перелік Role включає значення Employe і Admin, що реалізує двохрівневу рольову модель. Це дало змогу відокремити адміністративні сценарії від персоналізованих сценаріїв працівника і використовувати роль як основу для подальшого обмеження функцій системи.

У клієнтській частині роль користувача додатково підтримується через Zustand-сховище, де зберігається поточний профіль та ознака адміністративного доступу. Така реалізація дає змогу адаптувати інтерфейс до ролі користувача, приховувати або відкривати окремі елементи навігації та відтворювати стан профілю після повторного відкриття застосунку. При цьому остаточний контроль доступу зберігається на серверному й middleware-рівнях, що не дозволяє звести безпековий механізм лише до клієнтської логіки. До першої групи екранних форм належать базові користувацькі сторінки, які забезпечують початкову

взаємодію із системою, а саме вхід до веб-застосунку та перегляд персонального профілю користувача. Відповідні інтерфейси наведено на рис. 4.5 і рис. 4.6.

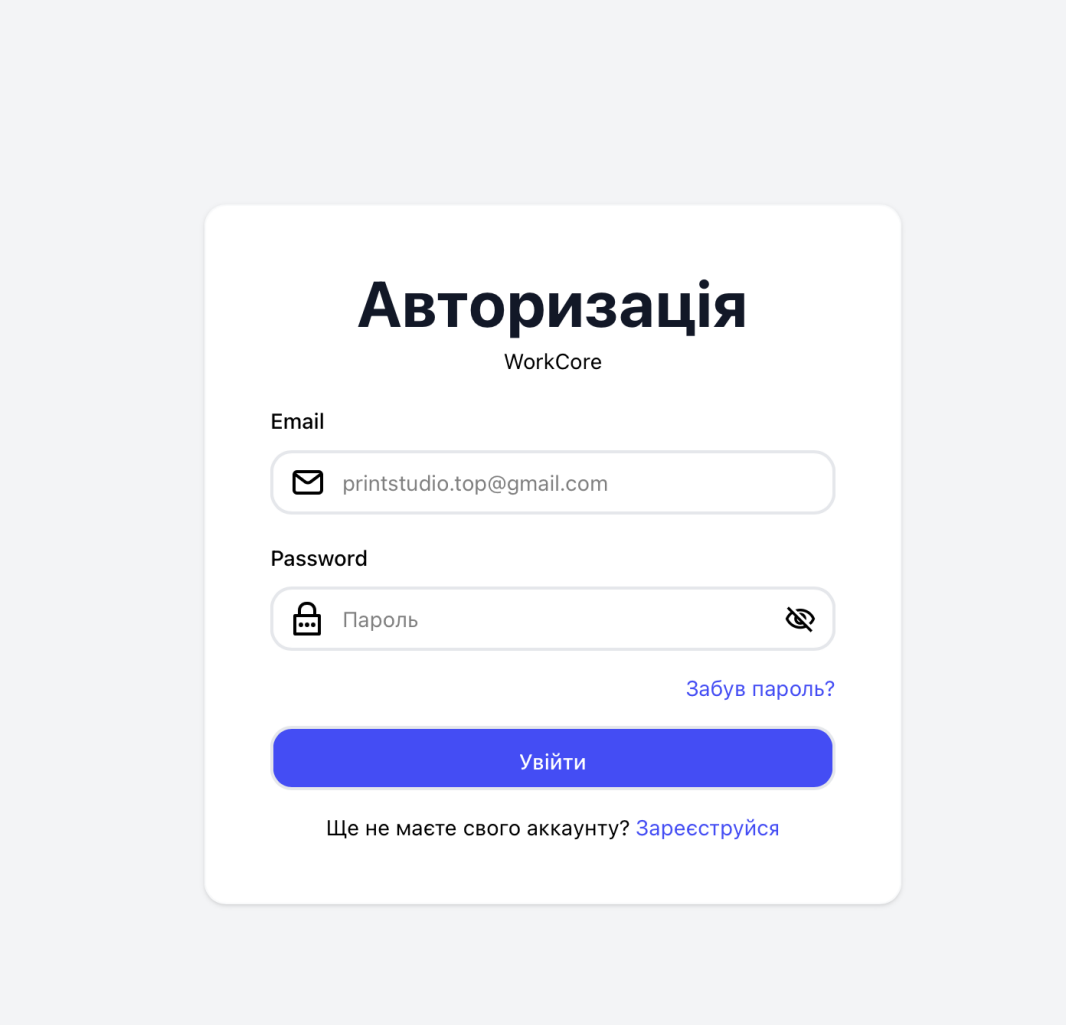

Рис. 4.5 Сторінка входу до веб-застосунку

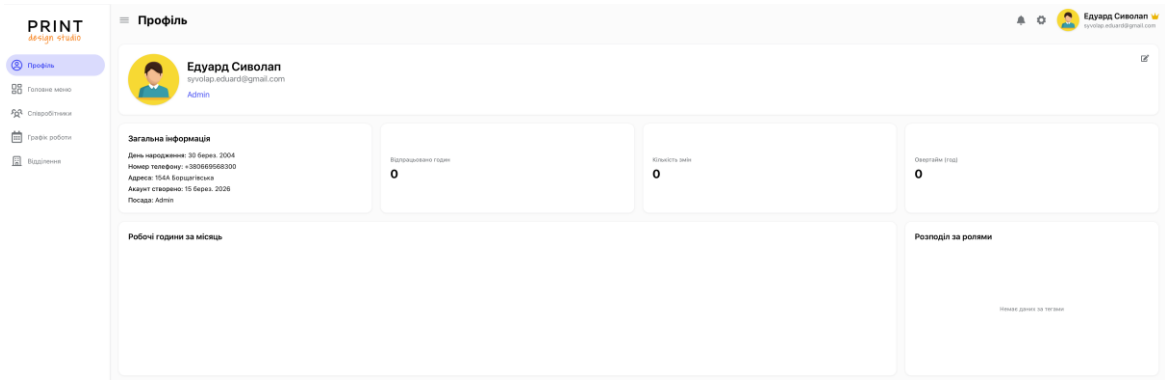

Рис. 4.6 Сторінка профілю користувача

4.4 Реалізація модулів підрозділів, робочих змін і графіків роботи

Модуль підрозділів реалізовано на основі моделі Department, у якій зберігаються назва, адресні дані, часові параметри роботи та координати. Така структура дозволяє використовувати підрозділ як повноцінну організаційну одиницю, до якої надалі прив'язуються зміни, графіки та блокування розкладу. Для малого бізнесу це є важливим, оскільки система обліку робочого часу повинна працювати не лише на рівні окремого працівника, а й на рівні конкретної точки або структурного сегмента підприємства.

Модуль робочих змін побудовано на основі моделі WorkShift, яка містить посилання на користувача і підрозділ, дату, час початку, час завершення, тривалість роботи і статус. Додатково зміни можуть пов'язуватися з тегами через модель Tag, що розширює можливості класифікації записів. Такий підхід дозволяє трактувати зміну не лише як часовий інтервал, а як окремий робочий об'єкт із внутрішнім станом опрацювання.

Модуль графіків роботи реалізовано через модель WorkSchedule, яка відокремлює плановий розклад від фактичних змін. У ній зберігаються посилання на користувача і підрозділ, дата, часові межі та ознака вихідного дня. Додатково використовується WorkScheduleLock, що дає змогу фіксувати блокування графіка на рівні підрозділу і тижня. У сукупності це формує гнучку модель, у якій графік і зміна не дублюють один одного, а представляють різні рівні обліку робочого часу. Друга група екранних форм відображає функціональні сторінки веб-застосунку, пов'язані з керуванням працівниками, графіками роботи, персональними налаштуваннями та додатковими можливостями взаємодії із системою. Приклади таких інтерфейсів наведено на рис. 4.7–4.10.

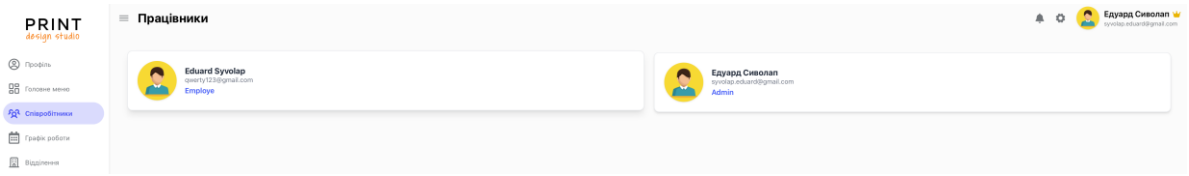

Рис. 4.7 Сторінка керування підрозділами

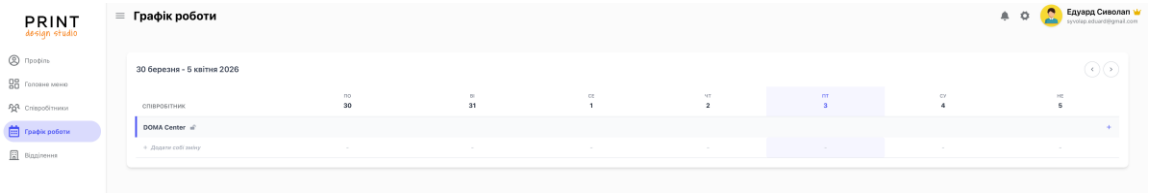

Рис. 4.8 Інтерфейс роботи з графіком змін

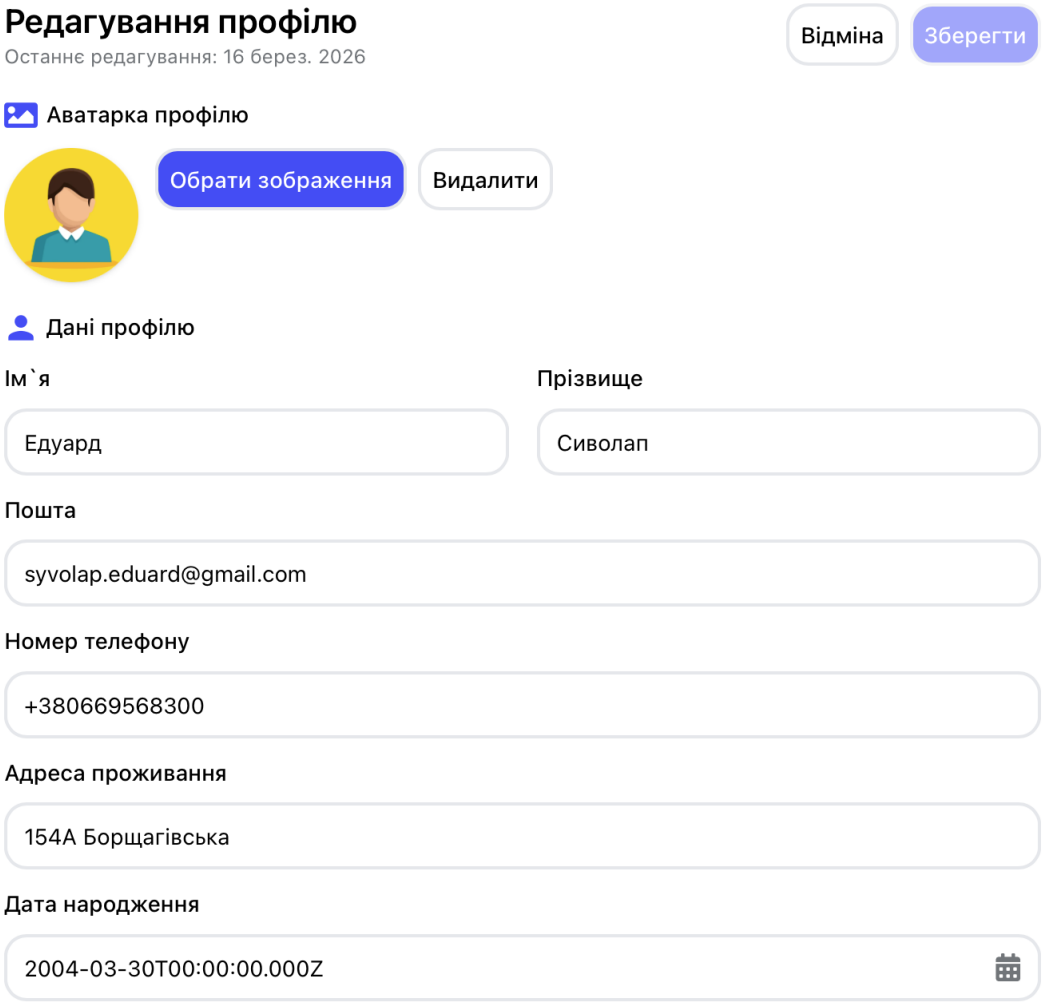

Рис. 4.9 Редагування профілю

4.5 Реалізація сповіщень і додаткових інтеграцій

Механізм сповіщень реалізовано як окремий серверний контур, що поєднує збереження повідомлень у базі даних і подієву доставку користувачеві. Для цього в моделі даних використано сутність Notification, а в серверному стеку - засоби WebSocket-взаємодії через @nestjs/platform-socket.io, @nestjs/websockets і socket.io. Це дозволило сформувати окремий канал подій, завдяки якому користувач може отримувати нові повідомлення без ручного оновлення сторінки.

Додатковий канал взаємодії реалізовано через Telegram. Серверна частина містить залежності nestjs-telegraf і telegraf, що вказує на інтеграцію бота в архітектуру системи. У прикладному вимірі Telegram використовується не лише для надсилання службових повідомлень, а й для сценаріїв прив’язки акаунта та роботи зі зміною. Це підвищує доступність системи для працівників, оскільки частина дій може виконуватися поза браузерним інтерфейсом.

Таким чином, реалізація сповіщень і додаткових інтеграцій розширює класичну клієнт-серверну модель застосунку. Система не обмежується лише сторінками веб-інтерфейсу, а підтримує подієву логіку в реальному часі та зовнішній повідомлювальний канал, що підсилює практичну цінність розробленого рішення для щоденної організації праці.

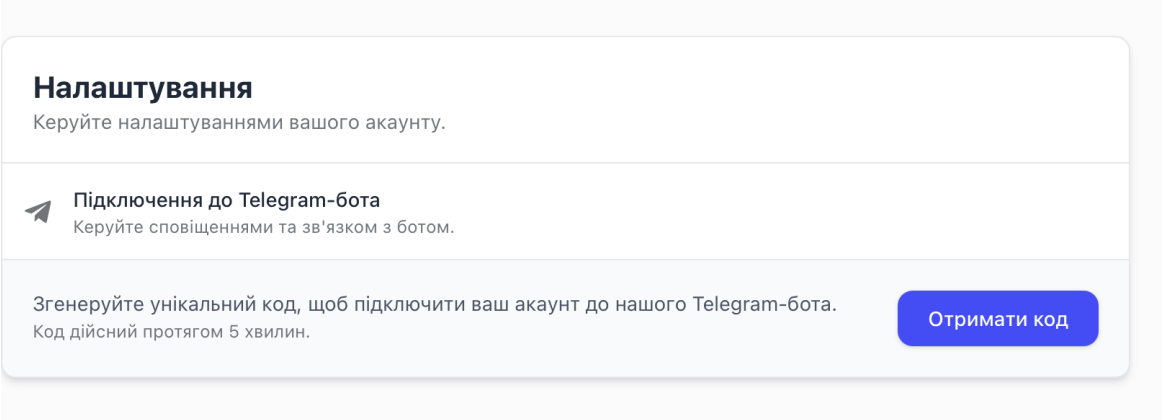

Рис. 4.10 Підключення сповіщень у Telegram

4.6 Реалізація користувацького інтерфейсу та основних екранних форм

Користувацький інтерфейс реалізовано як багатосторінкове робоче середовище з постійною навігацією, функціональними сторінками й допоміжними компонентами. У клієнтській частині присутні Sidebar, Header, модальні елементи, форми, пагінаційний компонент і засоби роботи з подіями. Це свідчить, що інтерфейс побудовано як систему взаємопов'язаних компонентів, а не як набір ізольованих шаблонів.

Основні екранні форми відтворюють логіку предметної області. Інформаційна панель виконує роль стартового інтерфейсу після входу до системи. Сторінки співробітників і підрозділів призначені для роботи зі списками сутностей і пов'язаними діями адміністрування. Сторінка графіка орієнтована на часовий вимір даних, а сторінка профілю - на персоналізовану інформацію користувача. Така реалізація відповідає маршрутам клієнтської частини і дозволяє зберегти функціональне розмежування між різними сегментами інтерфейсу.

У допоміжних сценаріях важливу роль відіграють форми, модальні вікна, повідомлення і календарні елементи. react-hook-form забезпечує керовану роботу з полями введення, react-day-picker підтримує календарну логіку, react-modal - локальні діалоги, а sonner - ненав'язливе відображення службових повідомлень. Це дозволяє інтерфейсу залишатися прикладним і зручним для щоденного використання.

4.7 Результат програмної реалізації веб-застосунку

У результаті програмної реалізації створено веб-застосунок, який поєднує клієнтський інтерфейс, серверну обробку, централізоване збереження даних і додаткові канали сповіщення. На клієнтському рівні система надає сторінки для роботи з основними сутностями предметної області, а на серверному - модульну

логіку обробки користувачів, підрозділів, змін, графіків і повідомлень. Для збереження даних використано PostgreSQL, для сесійного механізму - Redis, а для подієвої та зовнішньої взаємодії - Socket.IO і Telegram.

Реалізований застосунок покриває основні сценарії, визначені в попередніх розділах: вхід до системи, рольове розмежування доступу, роботу зі співробітниками й підрозділами, формування графіків, створення та опрацювання змін, перегляд профілю і отримання повідомлень. Таким чином, отриманий програмний результат відповідає заявленій темі та фактичній структурі проєкту.

Інфраструктурно система придатна до локального або узгодженого контейнеризованого запуску, оскільки архітектура проєкту з самого початку побудована з урахуванням розділення клієнта, сервера, бази даних і сесійного шару. Це підвищує цілісність реалізації і створює основу для подальшого розвитку функціональності без перебудови базової моделі застосунку.

4.8 Тестування функціональності веб-застосунку

Тестування функціональності веб-застосунку доцільно будувати навколо основних користувацьких сценаріїв. Для цього проєкту першочерговими є перевірка входу до системи, коректного перенаправлення між сторінками автентифікації й захищеними маршрутами, обмеження доступу без сесії, а також перевірка сценаріїв перегляду та редагування даних, пов’язаних зі співробітниками, підрозділами, графіками і змінами. Окремо повинні перевірятися отримання сповіщень і робота додаткового Telegram-каналу. Такий підхід повністю відповідає фактичній побудові застосунку, у якому middleware, серверні модулі та модель даних прямо визначають набір критичних функцій, що потребують перевірки.

З технічного погляду проєкт підтримує тестовий і відлагоджувальний контур також на рівні серверної частини, оскільки в package.json передбачено сценарії test, test:watch, test:cov, test:debug і test:e2e. Це свідчить про готовність

серверного рівня до використання інструментів тестування, навіть якщо в межах прикладної перевірки основний акцент зроблено на функціональних сценаріях користувача. Для бакалаврської роботи найбільш доцільним є поєднання сценарної перевірки інтерфейсу з перевіркою ключових серверних реакцій на запити користувача.

У межах функціонального тестування доцільно перевіряти не лише позитивні сценарії, а й граничні випадки: спробу доступу без авторизації, використання забороненого типу пристрою, некоректні вхідні дані у формах, відсутність прив’язки Telegram-акаунта, створення зміни без відповідного графіка або відображення порожніх списків сутностей. Саме такі ситуації найкраще виявляють стійкість системи до реальних умов використання.

4.9 Аналіз результатів тестування

Аналіз результатів тестування повинен бути спрямований не лише на фіксацію факту працездатності системи, а й на оцінювання того, наскільки реалізація покриває ключові вимоги, сформульовані для веб-застосунку. У цьому проєкті такими критичними зонами є коректна робота сесійного доступу, узгодженість рольової моделі, стабільність операцій зі співробітниками, підрозділами, графіками і змінами, а також надійність відображення подій через сповіщення та Telegram. Оскільки ці сценарії прямо відповідають архітектурі застосунку, саме вони мають становити основу підсумкового аналізу тестування.

З погляду функціональної повноти проєкт демонструє узгодженість між клієнтською й серверною частинами. Маршрутизація фронтенду, серверні модулі, реляційна модель даних і додаткові інтеграції працюють у межах однієї предметної логіки. Це означає, що система не складається з ізольованих фрагментів, а реалізує цілісний цикл: від автентифікації та доступу до захищених сторінок - до роботи зі змінами, графіками і повідомленнями. Саме така узгодженість є головним результатом тестової перевірки для застосунку цього типу.

Підсумкова оцінка результатів тестування дає підстави розглядати реалізований веб-застосунок як працездатне програмне рішення для обліку робочих годин співробітників малого бізнесу. Архітектура проєкту підтримує подальше розширення функціональності, а реалізовані модулі вже формують завершений прикладний контур системи. Це дає змогу перейти від етапу перевірки базової функціональності до подальшого розвитку інтерфейсу, аналітичних компонентів і засобів взаємодії з користувачем.

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи стала розробка веб-застосунку для обліку робочих годин співробітників малого бізнесу, призначеного для централізованого керування користувачами, підрозділами, графіками роботи, робочими змінами, профілями, службовими повідомленнями та додатковими каналами взаємодії. Реалізоване програмне рішення поєднує клієнтську й серверну частини, базу даних, сесійний механізм, рольову модель доступу та засоби контейнеризованого розгортання. Функціональність застосунку орієнтована на практичні потреби малого бізнесу, де облік робочого часу часто здійснюється за допомогою таблиць, повідомлень або розрізнених цифрових інструментів, що ускладнює контроль актуальності даних, координацію змін і доступ до відомостей про зайнятість персоналу.

За першим розділом проведено аналіз предметної області обліку робочих годин співробітників малого бізнесу. Встановлено, що специфіка такого обліку пов’язана з необхідністю одночасного ведення даних про працівників, підрозділи, графіки роботи, робочі зміни, часові інтервали та статуси опрацювання змін. Окремо визначено цільову аудиторію веб-застосунку, до якої належать адміністратори, керівники малого бізнесу, відповідальні особи в межах підрозділів і працівники, що потребують доступу до персонального графіка та призначених змін. Проаналізовано програмні аналоги Clockify, Toggl Track і Jibble, що дало змогу встановити їхні функціональні можливості та обмеження. Clockify має розвинені засоби time tracking, attendance і reporting, однак є надмірно універсальним для вузьких задач малого бізнесу. Toggl Track більше орієнтований на проєктний облік часу, аналітику та інтеграції. Jibble є найближчим за функціональністю до задач обліку змін і відвідуваності, проте залишається зовнішнім сервісом із наперед заданою моделлю роботи. На підставі порівняння аналогів обґрунтовано потребу у власному веб-застосунку, адаптованому до внутрішньої структури малого підприємства, та сформульовано функціональні й нефункціональні вимоги до системи.

За другим розділом обґрунтовано вибір засобів реалізації веб-застосунку. Для організації розробки використано Visual Studio Code як середовище роботи з клієнтською та серверною частинами, GitHub як засіб керування версіями, Docker і Docker Compose як інструменти контейнеризації та локального розгортання. Клієнтську частину побудовано з використанням TypeScript, Next.js, React, Zustand, TanStack Query, Axios та допоміжних бібліотек, що забезпечило модульність інтерфейсу, підтримку маршрутизації, роботу зі станами й обмін даними із сервером. Серверну частину реалізовано на основі NestJS, що дало змогу побудувати структуровану модульну архітектуру з окремими компонентами для автентифікації, користувачів, підрозділів, змін, графіків, тегів і сповіщень. Для взаємодії з базою даних використано Prisma ORM, для збереження структурованих даних — PostgreSQL, для підтримки сесійного механізму — Redis. Додатково враховано використання Socket.IO для подієвих сповіщень і Telegram-інтеграції як допоміжного каналу взаємодії користувача із системою.

За третім розділом виконано проєктування веб-застосунку. Архітектуру системи побудовано як клієнт-серверну модель, у якій клієнтська частина відповідає за взаємодію користувача з інтерфейсом, серверна частина — за обробку запитів і бізнес-логіку, PostgreSQL — за збереження основних даних, Redis — за підтримку сесій, а додаткові інтеграційні компоненти — за сповіщення й обмін подіями. Спроєктовано модель вимог до веб-застосунку, ролі користувачів і сценарії використання для адміністратора, працівника та Telegram-інтерфейсу. Визначено структуру бази даних із сутностями User, Department, WorkShift, WorkSchedule, WorkScheduleLock, Notification і Tag, що забезпечують відображення предметної області в реляційній моделі. Спроєктовано серверні модулі системи, клієнтську маршрутизацію, структуру сторінок інтерфейсу та механізми безпеки, зокрема перевірку сесії, розмежування прав доступу й обмеження функціональності відповідно до ролі користувача.

За четвертим розділом виконано програмну реалізацію та тестування веб-застосунку. Клієнтська частина містить сторінки входу, профілю, інформаційної панелі, співробітників, підрозділів, графіків роботи, налаштувань та інших функціональних екранів. Серверна частина охоплює модулі автентифікації, користувачів, підрозділів, робочих змін, графіків роботи, тегів, сповіщень і Telegram-взаємодії. Реалізовано сесійний механізм доступу з використанням Redis, cookie-based взаємодії та role-based обмеження функцій для адміністратора і працівника. У системі забезпечено створення, редагування та перегляд даних про співробітників, підрозділи, робочі зміни й графіки роботи. Реалізовано подієві сповіщення через Socket.IO та дублювання окремих повідомлень через Telegram. Telegram-інтеграція дає змогу прив'язати акаунт користувача й підтримувати сценарії, пов'язані з початком і завершенням зміни. Контейнеризоване розгортання через Docker Compose забезпечує запуск клієнтської частини, серверної частини, PostgreSQL і Redis як окремих сервісів.

Тестування підтвердило працездатність основних сценаріїв веб-застосунку: входу до системи, доступу до захищених сторінок, перегляду профілю, роботи з користувачами, підрозділами, графіками, робочими змінами та сповіщеннями. Перевірка показала, що система коректно обмежує доступ відповідно до ролі користувача, забезпечує взаємодію між клієнтською і серверною частинами, зберігає дані в базі та підтримує основні операції адміністрування робочого часу. Практичний результат полягає у створенні програмного рішення, яке дає змогу малому бізнесу перейти від фрагментарного ведення графіків і часових записів до централізованого цифрового обліку. Веб-застосунок зменшує кількість ручних операцій, підвищує впорядкованість доступу до інформації, спрощує координацію між адміністратором і працівниками та формує єдине середовище для роботи з даними про зайнятість персоналу. Архітектура системи допускає подальше розширення функціональності за рахунок розвитку аналітичних засобів, деталізації ролей користувачів, удосконалення механізмів звітності та розширення зовнішніх інтеграцій.

ПЕРЕЛІК ПОСИЛАНЬ

1. Axios Docs: Getting Started [Електронний ресурс]. - Назва з екрана. - Дата звернення: 03.04.2026.
2. Clockify™ - FREE Time Tracking Software [Електронний ресурс]. - Назва з екрана. - Дата звернення: 03.04.2026.
3. Clockify Features [Електронний ресурс]. - Назва з екрана. - Дата звернення: 03.04.2026.
4. Docker Docs [Електронний ресурс]. - Official Docker documentation. - Дата звернення: 03.04.2026.
5. Jibble: FREE Time Tracking Software [Електронний ресурс]. - Назва з екрана. - Дата звернення: 03.04.2026.
6. Jibble: FREE Time Clock Software [Електронний ресурс]. - Назва з екрана. - Дата звернення: 03.04.2026.
7. Documentation | NestJS - A progressive Node.js framework [Електронний ресурс]. - Official NestJS documentation. - Дата звернення: 03.04.2026.
8. Next.js Docs [Електронний ресурс]. - Official Next.js documentation. - Дата звернення: 03.04.2026.
9. PostgreSQL: Documentation [Електронний ресурс]. - Official PostgreSQL documentation. - Дата звернення: 03.04.2026.
10. Prisma Documentation [Електронний ресурс]. - Official Prisma documentation. - Дата звернення: 03.04.2026.
11. Prisma. Overview of Prisma Schema [Електронний ресурс]. - Назва з екрана. - Дата звернення: 03.04.2026.
12. React [Електронний ресурс]. - Official React documentation. - Дата звернення: 03.04.2026.
13. Redis docs [Електронний ресурс]. - Official Redis documentation. - Дата звернення: 03.04.2026.
14. Introduction | Socket.IO [Електронний ресурс]. - Official Socket.IO documentation. - Дата звернення: 03.04.2026.

15. Overview | TanStack Query React Docs [Електронний ресурс]. - Official TanStack Query documentation. - Дата звернення: 03.04.2026.

16. Telegram Bot API [Електронний ресурс]. - Official Telegram API documentation. - Дата звернення: 03.04.2026.

17. Toggl Track: Time Tracking Software for Any Workflow [Електронний ресурс]. - Назва з екрана. - Дата звернення: 03.04.2026.

18. Toggl Track Features [Електронний ресурс]. - Назва з екрана. - Дата звернення: 03.04.2026.

19. TypeScript Documentation [Електронний ресурс]. - Official TypeScript documentation. - Дата звернення: 03.04.2026.

20. Zustand: Introduction [Електронний ресурс]. - Official Zustand documentation. - Дата звернення: 03.04.2026.