Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра мультимедійних систем

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**Розробка ядра CRM системи із візуалізацією процесу виконання проєктів для компанії Omega Telecom**»

Виконала: студентка 4-го року навчання

Спеціальності

121 «Інженерія програмного забезпечення»

Матвієнко Ірина Валентинівна

Керівник Борозенний Сергій Олександрович, старший викладач

Рецензент _____

(прізвище та ініціали)

Кваліфікаційна робота захищена з оцінкою _____

Секретар ЕК _____

«____» _____ 2023р.

Київ – 2023

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра мультимедійних систем

Освітній ступінь бакалавр

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри мультимедійних систем

доцент О. П. Жежерун

«____» _____2023 року

ЗАВДАННЯ

ДЛЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Матвієнко Ірині Валентинівні

1. Тема роботи «**Розробка ядра CRM системи із візуалізацією процесу виконання проєктів для компанії Omega Telecom**»
керівник роботи Борозенний Сергій Олександрович, старший викладач
затверджені наказом вищого навчального закладу від
«____» _____2023 року № __
2. Строк подання студентом роботи: 24 травня 2023
3. План роботи

   Вступ

   Розділ 1. Аналіз предметної області

   Розділ 2. Проєктування системи

   Розділ 3. Вибір інструментарію

   Розділ 4. Опис реалізації

   Висновки

   Список використаних джерел

ГРАФІК ПІДГОТОВКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	жовтень			
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	жовтень - листопад			
3.	Складання плану кваліф. роботи та узгодження з науковим керівником	листопад			
4.	Написання розділів роботи	листопад – квітень			
5.	Проміжний контроль виконання роботи	лютий			
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	лютий - квітень			
	Розділ 1 (Аналіз предметної області)	січень			
	Розділ 2 (Проєктування системи)	березень			
	Розділ 3 (Вибір інструментарію)	березень			
	Розділ 4 (Опис реалізації)	квітень			
7.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	квітень – початок травня			
8.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності	середина травня			
9.	Подання на зовнішню рецензію	середина травня			
10.	Підготовка до захисту кваліфікаційної роботи на засіданні кафедри: написання доповіді та виготовлення ілюстративного матеріалу	до 9 травня			
11.	Попередній захист кваліфікаційної роботи на засіданні кафедри	до 10 травня			
12.	Подання кваліфікаційної роботи на кафедру з усіма супроводжувальними документами	до 24 травня			
13.	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	згідно з розкладом роботи ЕК			

Графік узгоджено 15 жовтня 2022 р.

Науковий керівник Борозенний Сергій Олександрович

Виконавець кваліфікаційної роботи Матвієнко Ірина Валентинівна

ЗМІСТ

ВСТУП

Для успішної роботи компаній їм потрібно забезпечувати різноманітні внутрішні процеси та взаємодію між ними. Завдання різних відділів мають бути визначені та задокументовані, а отже, ця інформація має десь зберігатися і бути зручно доступною для працівників. Використання для цієї мети файлових каталогів, таблиць Excel та чатів призводить до порушення цілісності даних, несанкціонованого доступу, нерозуміння своїх задач та плутанини в безлічі документів. Кращим рішенням є використання CRM системи – налагодженої програмної системи для управління взаємовідносинами з клієнтами та оптимізації бізнес-процесів [1].

CRM системи надають компаніям переваги, як-от: централізоване збереження даних про контрагентів і працівників та управління ними, прийняття обґрунтованих рішень на основі аналітичних даних про ефективність роботи. CRM системи покращують комунікацію та співпрацю між співробітниками, бо надають спільну платформу для відстеження оновлень прогресу в реальному часі. Також CRM системи підвищують продуктивність роботи та дозволяють зекономити час шляхом автоматизації рутинних та повторюваних завдань. Важливою частиною CRM систем є візуалізація бізнес-процесів, що полегшує сприйняття великої кількості інформації, яку необхідно опрацьовувати.

Omega Telecom – компанія, що надає телекомунікаційні послуги. Компанія не є винятком і потребує специфічної CRM системи для ефективної організації каналів зв’язку. Робочі процеси компанії Omega Telecom є об’єктом дослідження даної роботи. Предмет – розробка веб-застосунку для автоматизації робочих процесів цієї компанії.

Постановка задачі – спроєктувати та розробити ядро CRM системи у вигляді веб-застосунку для компанії Omega Telecom. Система повинна автоматизувати та візуалізувати процес виконання проєктів, що складаються із послідовності завдань.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Огляд предметної області

Основним напрямком діяльності компанії Omega Telecom є організація каналів зв'язку. Побудова каналу зв'язку – складний проєкт, який складається з невеликих чітко визначених завдань з дедлайнами, кожне з яких може виконати один працівник певного відділу або компанія-контрагент за укладеним договором. Завдання залежать одне від одного – вони можуть виконуватися послідовно або паралельно. Виконання деяких завдань має бути ретельно перевірене кваліфікованими працівниками та, відповідно, підтверджене або відхилене. Завдання може мати компанію-замовника, тоді воно виконується за умовами, прописаними в договорі. Виконання всього проєкту контролюється відповідальним за проєкт працівником – тим, хто створив проєкт, або його керівником, який може відхилити виконання проєкту, якщо щось піде не за планом. За вартістю всіх завдань вкінці обраховується вартість виконання проєкту. Керівник компанії повинен отримувати звіти по роботі працівників та виконанню проєктів.

1.2. Аналіз існуючого рішення, що використовується

Компанія Omega Telecom вже використовує власну CRM систему, яка, проте, містить баги та не реалізовує деякі важливі для компанії вимоги, що стало зрозумілим завдяки проведеному опитуванню двох працівників компанії.

В системі присутній пункт меню Проєкти, але ним не користуються, бо працівники не розуміють його відмінності від Завдання, та інтерфейс користувача заплутаний. Зазвичай створюють одне Завдання, яке містить всі підзавдання, необхідні для побудови каналу. Перехід від завершеного до нового підзавдання відбувається шляхом комунікації в чаті, який є при кожному Завданні (виходить, це один чат на цілий великий проєкт), а позначити завдання як послідовні чи паралельні можна лише за домовленістю в чаті чи прописавши це текстом в полі Опис; також для того, аби зрозуміти,

на якому етапі виконання знаходиться проєкт, потрібно уважно прочитати чат чи навіть уточнювати у співробітників). Завдання містить лише поля Назва, Опис, Виконавець та Дедлайн, отже, немає можливості автоматичного обрахування вартості проєкту та обов'язкового прикріплювання договорів. Важливе поле Дедлайн завдання написане маленьким шрифтом десь праворуч, натомість працівники компанії хотіли б, щоб це поле на сторінці певного завдання одразу привертало до себе увагу і було зображене червоним кольором. Функціональність генерації звітів у системі також відсутня.

Із корисної функціональності системи, в базі даних зберігаються працівники та контрагенти, а система надає можливості зручного пошуку та виконання CRUD-операцій з цими сутностями. Також список завдань можна фільтрувати за рівнем відповідальності за них працівника – поточного користувача, проте фільтри часом застосовуються некоректно. В системі присутній суттєвий баг – в списку періодично дублюються ті самі завдання, які ніяк не реагують на кліки мишкою.

Доступу до коду існуючої системи немає і туди довелося б вносити дуже багато змін, тому було прийняте рішення розробити нову систему з нуля.

1.3. Аналіз існуючих аналогів CRM систем

Розглянемо декілька існуючих аналогів CRM систем.

SendPulse – CRM система для автоматизації маркетингу [1]. Основним спрямуванням є залучення нових клієнтів, комунікація з ними через чат-боти у соціальних мережах та керування угодами. Візуалізація процесу виконання угод відбувається за допомогою Kanban-дошки, де картки-угоди можна переносити по колонках-статусах та фільтрувати. Функціональність керування угодами є безкоштовною, але за вбудовану можливість комунікації з клієнтами через різні канали зв'язку необхідно платити. Із недоліків також можна зазначити відсутність розподілення відповідальності за процес виконання угод між працівниками – до угоди прикріплюється один працівник-

виконавець, але всі працівники-користувачі можуть редагувати всі угоди та змінювати їхній статус. Ще одним недоліком є те, що угода – це одна сутність, яка по суті є аналогом проєкту, але не розподіляється на менші завдання. Відсутня функціональність генерації звітів.

Monday work management – CRM система, що покриває всі аспекти робочих процесів в одному комплексному рішенні [2]. Надає інструменти для керування проєктами, завдання можна переглядати у вигляді списку та Kanban-дошки за статусами і з фільтрацією. Із переваг можна виокремити можливість зберігати документи, візуалізацію звітів по проєктах за допомогою графіків та наявність шаблонів для створення проєктів. Із недоліків безкоштовний тариф охоплює лише два користувачі, а більша кількість користувачів, керування контактами, візуальні канали продажів та інша важлива функціональність є платною [1].

KeepinCRM – українська CRM система для малого та середнього бізнесу, що дозволяє керувати продажами та підрядниками, полегшити комунікацію з клієнтами, вести фінансовий облік [3]. Основним спрямуванням є продаж фізичних товарів, бо є інтеграція з Новою Поштою та УкрПоштою (та багатьма іншими сервісами), оптимізація процесу доставки та робота з інформацією про товари на складах [1]. Однак, присутня і візуалізація угод та завдань за допомогою Kanban-дошки та списку з фільтрацією (але завдання не групуються за проєктами, проєкти як такі відсутні). Із переваг є тригери для автоматизації виконання угод та генерація звітів. Основними недоліками є лише один користувач та дуже обмежена функціональність в безкоштовному тарифі та лише два користувачі в платному тарифі. Тому інші працівники не мають змоги спостерігати за прогресом завдань та угод, оновлювати статус своїх завдань, користуватися списком підрядників.

Отже, більшість існуючих CRM систем є платними, але не гарантують реалізацію специфічних вимог компанії до автоматизації бізнес-процесів, включно з процесом виконання проєктів. Зокрема, в розглянутих CRM

системах відсутнє підтвердження виконання завдань більш кваліфікованими працівниками, немає обов’язкового прикріплення договорів із контрагентами, які є виконавцями та замовниками. В CRM системах зазвичай використовується Kanban-дошка як спосіб візуалізації процесу виконання завдань (угод). При такому підході працівник бачить всі завдання компанії, навіть ті, які його не стосуються, йому складно зрозуміти необхідний порядок виконання завдань в межах одного проєкту і коли саме їх потрібно починати виконувати. Кращим рішенням буде використання діаграм у власній розроблюваній системі для візуалізації послідовності завдань в проєкті та залежності завдань одне від одного. Це також забезпечить гнучку зміну порядку завдань, зручне додавання нових завдань зі зв’язками з існуючими завданнями та полегшить планування великих проєктів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ

2.1. Опис груп користувачів

Система буде мати таких користувачів:

- Працівник.
- Керівник відділу – може виконувати всі дії, що і працівник, але несе відповідальність за виконання завдань та проєктів.
- Адміністратор.

2.2. Технічні вимоги користувачів до функціональності системи

Кожен користувач повинен мати можливість вирішувати такі задачі:

- додавати проєкти із послідовністю завдань;
- змінювати назву своїх створених незавершених проєктів;
- видаляти свої створені проєкти, якщо їх виконання ще не розпочато;
- розпочинати виконання своїх створених проєктів;
- переглядати список доступних проєктів (свої створені проєкти, проєкти із завданнями свого відділу), виконувати пошук за назвою та фільтрацію списку проєктів за станом прогресу та своїм рівнем відповідальності за проєкт;
- переглядати детальну інформацію по доступних проєктах, включно із послідовністю завдань цього проєкту;
- додавати нове завдання в свій створений незавершений проєкт у послідовність завдань, лише якщо після нього немає завдань або якщо виконання наступних завдань ще не розпочато;
- редагувати та видаляти завдання свого створеного незавершеного проєкту, якщо їх виконання ще не розпочато;

- додавати нотатки до своїх створених завдань та завдань свого відділу, що належать до незавершених проєктів;
- додавати та видаляти зв’язки послідовності між завданнями свого створеного незавершеного проєкту;
- позначати завдання як виконане, якщо він призначений виконавцем завдання і до завдання дійшла черга виконання;
- переглядати список доступних завдань (свої створені завдання, завдання свого відділу) з усіх доступних проєктів, виконувати пошук за назвою та фільтрацію списку завдань за станом прогресу та своїм рівнем відповідальності за завдання;
- переглядати детальну інформацію по доступних завданнях;
- переглядати список працівників компанії, виконувати пошук за ПІБ та email і фільтрацію списку працівників за відділами;
- переглядати список компаній-контрагентів, виконувати пошук за назвами;
- переглядати свій профіль;
- змінювати пароль для входу в систему.

Керівник відділу, окрім вище зазначених задач, доступних для всіх користувачів, повинен мати можливість вирішувати такі задачі:

- змінювати назву незавершених проєктів, за які він відповідальний (проєктів, які створив він сам або його підлеглі);
- видаляти проєкти, за які він відповідальний, якщо їх виконання ще не розпочато;
- розпочинати виконання проєктів, за які він відповідальний;

- позначати проєкт в процесі виконання, за який він відповідальний, як провальний, щоб виконання проєкту припинилося;
- переглядати список доступних проєктів, в які також входять проєкти, за які він відповідальний;
- додавати нове завдання в незавершений проєкт, за який він відповідальний, у послідовність завдань, лише якщо після нього немає завдань або якщо виконання наступних завдань ще не розпочато;
- редагувати та видаляти завдання незавершеного проєкту, за який він відповідальний, якщо їх виконання ще не розпочато;
- додавати нотатки до завдань проєкту, за який він відповідальний;
- додавати та видаляти зв’язки послідовності між завданнями незавершеного проєкту, за який він відповідальний;
- позначати завдання, за яке він відповідальний, як виконане, якщо до завдання дійшла черга виконання;
- переглядати список доступних завдань, в які також входять завдання, за які він відповідальний, і завдання проєктів, за які він відповідальний;
- за замовчуванням бути виконавцем завдань, які має виконувати його відділ, якщо виконавець не призначений;
- призначати виконавців завдань, за які він відповідальний, якщо їх виконання ще не розпочато.

Адміністратор, окрім вище зазначених задач, доступних для всіх користувачів, повинен мати можливість вирішувати такі задачі:

- змінювати назву всіх незавершених проєктів;
- видаляти всі проєкти;
- розпочинати виконання всіх проєктів;

- позначати будь-який проєкт в процесі виконання як провальний, щоб виконання проєкту припинилося;
- переглядати список всіх проєктів;
- переглядати детальну інформацію по всіх проєктах, включно із послідовністю завдань цього проєкту;
- додавати нове завдання в будь-який створений незавершений проєкт у послідовність завдань, лише якщо після нього немає завдань або якщо виконання наступних завдань ще не розпочато;
- редагувати та видаляти завдання будь-якого незавершеного проєкту, якщо їх виконання ще не розпочато;
- додавати та видаляти зв’язки послідовності між завданнями будь-якого незавершеного проєкту;
- позначати завдання як виконане, якщо до завдання дійшла черга виконання;
- переглядати список всіх завдань;
- переглядати детальну інформацію по всіх завданнях;
- додавати, редагувати, видаляти та переглядати інформацію про відділи;
- додавати, редагувати, видаляти та переглядати інформацію про посади по відділах;
- додавати, редагувати, видаляти та переглядати інформацію про працівників;
- додавати, редагувати, видаляти та переглядати інформацію по компаніях-контрагентах;
- отримувати звіти по роботі всіх працівників за весь час роботи або за певний період – такі показники по кожному працівнику: кількість

виконаних завдань, кількість запланованих для виконання завдань, кількість виконаних проєктів під відповідальністю, кількість провалених проєктів під відповідальністю, кількість запланованих для виконання проєктів під відповідальністю;

- отримувати звіти по виконанню всіх проєктів за весь час роботи або за певний період: загальна кількість виконаних проєктів, загальна кількість провалених проєктів, загальна сума на витрачених коштів (на виконані завдання); такі показники по кожному проєкту: відповідальний працівник, статус (виконано чи провалено), дата завершення виконання чи провалення відповідно, час виконання, чи прострочений дедлайн (для виконаних проєктів), сума витрачених коштів.

2.3. Архітектура застосунку

Веб-застосунки найчастіше реалізовують одну з трьох архітектур: моноліт, клієнт-серверну архітектуру чи мікросервіси.

Монолітна архітектура – це підхід до розробки програмного продукту, при якому весь застосунок є єдиним інтегрованим блоком коду з однією базою даних, а всі компоненти взаємодіють між собою через виклики функцій чи методів класів. Моноліт може бути зручним підходом для невеликих проєктів, але не підходить для складніших проєктів через низьку гнучкість та обмеження в масштабованості застосунку, сильну залежність views від бізнес-логіки.

Мікросервісна архітектура є підходом до розробки програмного продукту, при якому застосунок розбивається на невеликі незалежні сервіси, які працюють разом та комунікують між собою через мережу. Розроблюваний застосунок не настільки великий, щоб розподіляти його на мікросервіси, які додають складності у комунікації між собою, розгортанні, реалізації безпеки для кожного сервісу окремо та мають проблему дублювання коду.

Найбільш релевантним для розроблюваного застосунку є клієнт-серверна архітектура – модель обчислень, в якій сервер зберігає та надає більшість ресурсів та послуг, що запитує клієнт, який реалізує графічний інтерфейс взаємодії з користувачем. Перевагами цієї архітектури є розподілення реалізації бізнес-логіки та інтерфейсу користувача, повний контроль централізованої мережі над процесами та діяльністю, просте спільне використання ресурсів на різних платформах, доступ для користувачів до будь-якого файлу з центрального сховища у будь-який час [4]. Обрана клієнт-серверна архітектура застосунку зображена на рис. 2.1.

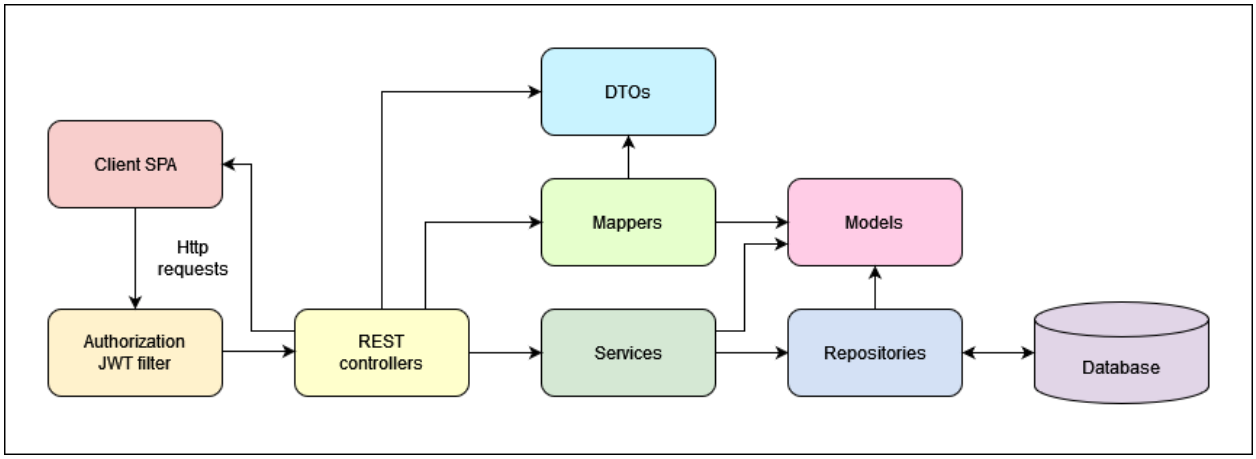

Рис. 2.1 Клієнт-серверна архітектура застосунку

Клієнт є Single Page Application (SPA), що використовує одну HTML-сторінку для відображення контенту, а динамічне оновлення вмісту цієї сторінки відбувається за допомогою JavaScript відповідно до дій користувача та отриманих даних у форматі JSON з серверу. Такий підхід дозволяє створювати більш динамічні та зручні у використанні веб-додатки, зменшуючи час завантаження і таким чином підвищуючи продуктивність їх роботи [5]. Логіка відправлення http запитів до серверу виокремлена в сервіси, до яких звертаються компоненти графічного інтерфейсу.

Http запит з JWT токеном з клієнта відправляється на сервер, де проходить авторизацію з допомогою JWT фільтра. Якщо токен валідний і користувач має

право доступу до ресурсу, відбувається перехід до відповідного REST контроллера.

Архітектуру серверної частини розроблено за допомогою патерну Controller-Service-Repository. Контроллер перехоплює http-запити після авторизації і відкриває клієнтам їм доступ до бізнес-логіки. Сервіс реалізовує всю бізнес-логіку, включно зі зміною стану сутностей. Репозиторій відповідає за збереження та отримання інформації з бази даних [6]. В якості допоміжних рівнів використовуються DTO (Data Transfer Object) та маппер (mapper). DTO потрібен для передачі даних між клієнтом та сервером – містить поля, які потрібні для передачі, але відрізняється від сутностей (може не мати певних полів сутностей або мати додаткові поля). Маппер відповідає для перетворення об'єктів сутностей на об'єкти DTO та навпаки. Чіткий розподіл відповідальності між рівнями полегшує розуміння, супровід, розширення та тестування коду, прискорює процес розробки.

При проєктуванні серверної частини також використано інверсію контролю, а точніше, впровадження залежностей. Інверсія контролю (Inversion of control або IoC) – принцип проєктування, який передає контроль над об'єктами або частинами програми контейнеру або фреймворку для досягнення слабкого зв’язування. Впровадження залежностей (dependency injection) - патерн, що реалізовує інверсію контролю, де контроль, який інвертується, полягає у встановленні залежностей об'єкта ззовні. Використання цього патерну надає такі переваги: відокремлення виконання задачі від її реалізації, спрощення заміни різних реалізацій, простіше тестування програми шляхом ізоляції компоненту або його заміни на тестову реалізацію (mock) [7].

2.4. ER-модель

ER-модель – інформаційна модель концептуального рівня “сутність-зв’язок”, що відображає бізнес-логіку предметної області (рис. 2.2). У кожної сутності є унікальний ідентифікатор – id.

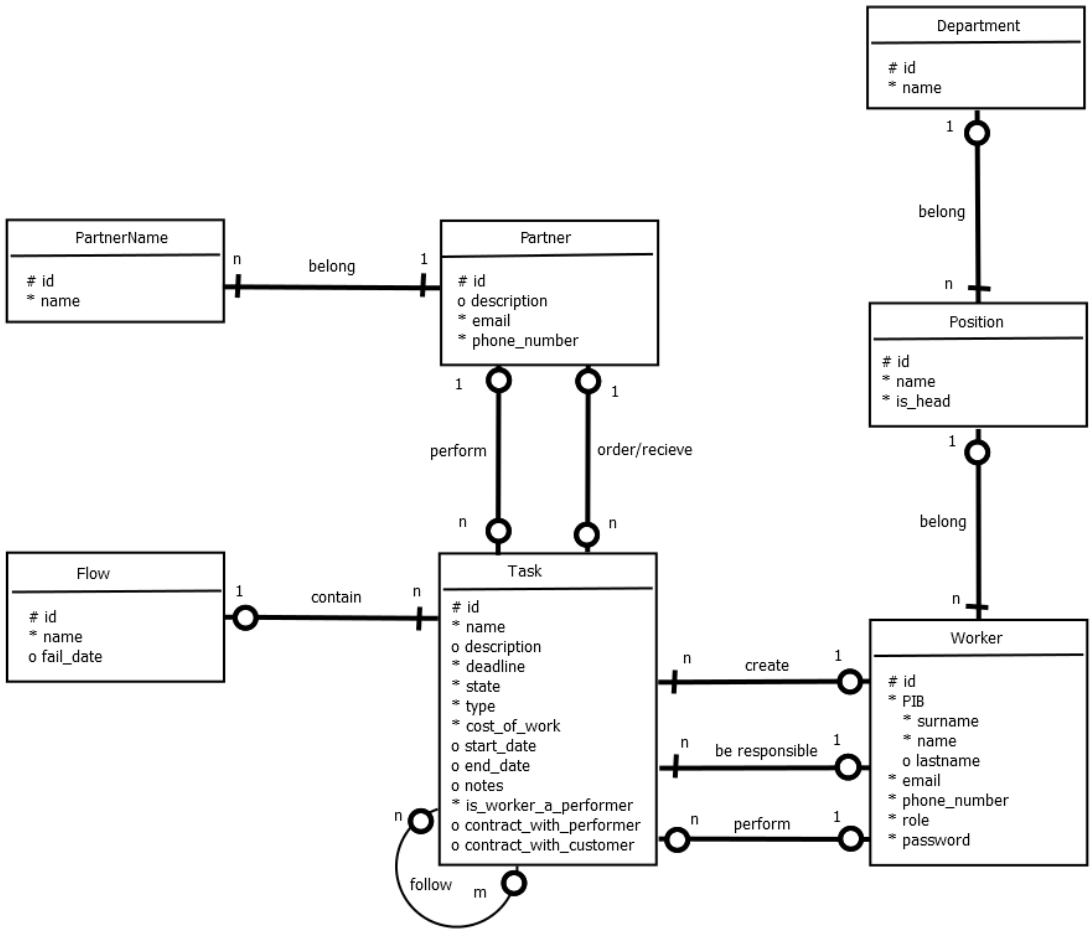

Рис. 2.2 ER-модель

Працівник (Worker), він же користувач, має одну з трьох ролей: працівник, керівник відділу, адміністратор (належить тільки керівнику компанії). Працівник обов’язково обіймає одну посаду; якщо посада є керівною, він має роль керівник відділу. Працівник може створювати та виконувати багато завдань і бути відповідальним за багато завдань (лише якщо він керівник відділу).

Посада (Position) обов’язково належить одному відділу та її можуть обіймати багато працівників.

Відділ (Department) може містити багато посад, але корпоративними обмеженнями контролюється наявність лише однієї керівної посади на відділ.

Завдання (Task) належить до одного з двох типів: звичайне виконання чи підтвердження прийому попередніх завдань. Завдання перебуває в одному з чотирьох станів: нове (виконання ще не розпочато), в процесі, виконано, провалено. Завдання обов’язково належить до одного проєкту. Завдання може мати багато попередніх (обов’язково мають бути виконані раніше) та наступних (обов’язково мають бути виконані пізніше) завдань. Завдання обов’язково має одного працівника, який його створив, та одного відповідального працівника. Виконавцем завдання може бути або працівник, або компанія-контрагент, через це зв’язок “perform” з кожною з двох моделей є необов’язковим, проте корпоративними обмеженнями контролюється обов’язкова наявність одного і тільки одного з них. Якщо виконавцем є компанія-контрагент, корпоративними обмеженнями контролюється обов’язкове прикріплення документу-договору про виконання робіт цією компанією. У завдання може бути один замовник – компанія-контрагент, тоді корпоративними обмеженнями контролюється обов’язкове прикріплення документу-договору про виконання робіт компанією Omega Telecom для компанії-замовника.

Проєкт (Flow) може містити багато завдань.

Компанія-контрагент (Partner) може мати багато назв, але обов’язково хоча б одну. Компанія-контрагент може бути виконавцем і замовником багатьох завдань.

Назва компанії-контрагента (PartnerName) обов’язково належить до однієї компанії-контрагента.

2.5. Діаграми станів

Для кращого розуміння зміни станів основних сутностей – Проєкту та Завдання варто використати діаграми станів. Діаграма станів – UML діаграма, що описує переходи між станами об’єкту системи, спричинені певними тригерами; використовується для моделювання життєвого циклу об’єкту [8].

2.5.1. Діаграма станів для сутності Проєкт

На рис. 2.3 зображено діаграму станів для сутності Проєкт. Проєкт може перебувати в чотирьох станах: новий (стан першого завдання – нове), в процесі, виконано, провалено.

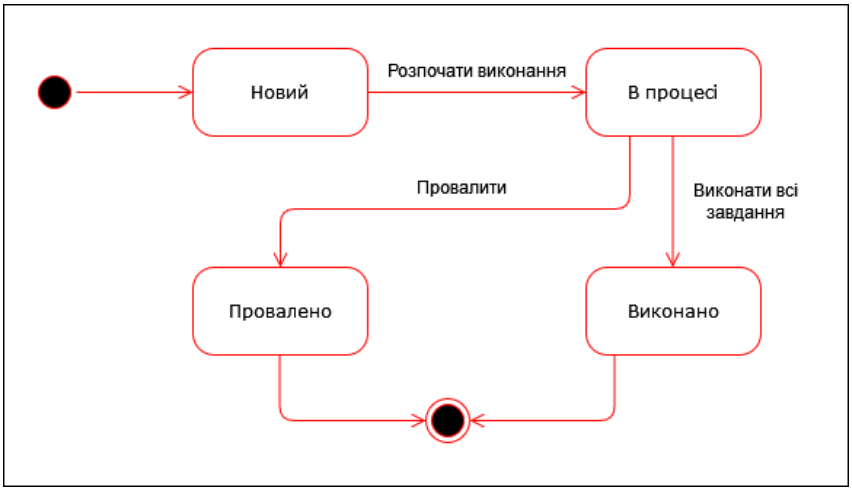

Рис. 2.3 Діаграма станів для сутності Проєкт

Визначення станів є релевантним лише для клієнтської частини застосунку та для генерації звітів на сервері. В базі даних зберігається лише значення fail_date – дата провалення проєкту, яка приймає значення null, якщо проєкт не провалено. Інші стани проєкту визначаються через стани його завдань. Зі стану новий до стану в процесі проєкт переходить, коли розпочинається його виконання (стан першого завдання стає в процесі). Зі стану в процесі проєкт може перейти до стану виконано або до стану провалено. До стану виконано проєкт переходить у разі виконання всіх завдань (стан усіх завдань проєкту – виконано). До стану провалено проєкт переходить, якщо відповідальний за проєкт працівник вирішує його провалити, бо щось пішло не так і подальше виконання не має сенсу.

2.5.2. Діаграма станів для сутності Завдання

На рис. 2.4 зображено діаграму станів для сутності Завдання. Можливі стани для завдання було перераховано вище, в розділі 2.4.

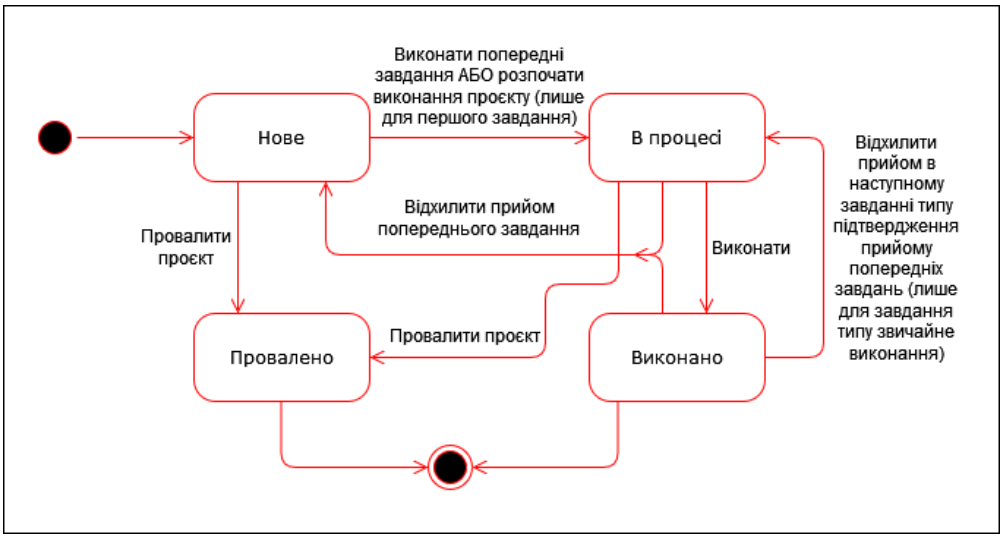

Рис. 2.4 Діаграма станів для сутності Завдання

Зі стану нове завдання може перейти до стану в процесі або до стану провалено. Зі стану нове до стану в процесі завдання переходить у двох випадках. Якщо це перше завдання проєкту, воно переходить у стан в процесі, коли розпочинається виконання проєкту. Будь-яке інше завдання переходить у стан в процесі у разі виконання всіх попередніх завдань (стан всіх попередніх завдань – виконано). Зі стану нове та зі стану в процесі до стану провалено завдання переходить у разі провалу проєкту (fail_date ≠ null). Зі стану в процесі завдання також може перейти до стану виконано шляхом його виконання (якщо тип завдання – підтвердження виконання попередніх завдань, виконанням завдання вважається підтвердження або відхилення прийому попередніх завдань). Якщо завдання А має тип звичайне виконання і вимагає підтвердження прийому одним або декількома наступними завданнями (які мають тип підтвердження виконання попередніх завдань), то зі стану виконано воно може знову перейти до стану в процесі у разі відхилення його прийому в наступному завданні. Тоді всі наступні завдання для завдання А повертаються до стану нове зі станів в процесі та виконано.

РОЗДІЛ 3. ВИБІР ІНСТРУМЕНТАРІЮ

3.1. Вибір інструментарію для розробки серверної частини застосунку

Для розробки серверної частини застосунку обрано об’єктно-орієнтовану мову програмування Java (в тому числі, через статичну типізацію) та Spring Boot, що є надбудовою над Spring. Spring — це фреймворк, що забезпечує комплексну інфраструктурну підтримку для розробки Java-додатків з використанням інверсії контролю; містить вбудовані корисні модулі, як-от: Spring MVC та Spring Security, що значно скорочують час розробки застосунку. Spring Boot – проєкт-надбудова над фреймворком Spring, який усуває типові конфігурації, необхідні для налаштування Spring-додатка, спрощує процес розробки та розгортання Spring-додатків [9].

Spring Boot дозволяє швидко підняти автономний, готовий до запуску в продакшн застосунок. Анотації надають можливість не писати boilerplate код, бо він може згенеруватися автоматично. Доступна велика кількість залежностей, які можна легко підключити в файл pom.xml та використовувати по всьому додатку. Перевагою також є зручність роботи з базою даних з використанням Object Relational Mapping (ORM).

Спеціальні залежності starter (дескриптори залежностей певної функціональності) спрощують конфігурацію додатка та збірки. Наприклад, Spring Boot Starter Data JPA спрощує роботу з реляційними базами даних, включно з автоматичною генерацією репозиторіїв з інтерфейсів. Spring Boot Starter Web додає необхідні бібліотеки для розробки веб-додатків зі Spring Boot, налаштовує такі аспекти веб-додатків, як обробка запитів та маршрутизація. Spring Boot Starter Security підключає фреймворк Spring Security, що використовується для аутентифікації, авторизації та шифрування паролів. Spring Boot Starter Mail використовується для надсилання електронних листів.

Завдяки наведеним та іншим перевагам для розробки серверу обрано саме Spring Boot, а не, наприклад, Node.js, що є іншим популярним серед розробників рішенням.

Використовуватиметься бібліотека Lombok для автоматичної генерації boilerplate коду для класів, як-от: геттери, сеттери, конструктори, методи equals(), toString() та інші. Для роботи з системою керування базами даних PostgreSQL використовуватиметься залежність Postgresql. Для роботи з JWT-токенами як способом авторизації використовуватиметься залежність Java Jwt з пакету OAuth0. Бібліотека IText використовуватиметься для створення документів у форматі pdf.

3.2. Вибір інструментарію для розробки клієнтської частини застосунку

Для розробки клієнтської частини застосунку обрано мову програмування JavaScript (JS) як мову, яка найчастіше використовується для створення динамічних інтерактивних веб-сторінок, та React. React – це JavaScript-бібліотека для розробки інтерфейсів користувача, що використовує підхід «компонентів» – розбиття інтерфейсу на малі, незалежні від інших частин блоки, що робить код більш організованим та легшим у розумінні [10]. React також використовує віртуальний DOM (Document Object Model), що дозволяє роботу з інтерфейсом в режимі реального часу завдяки мінімальному оновленню DOM відповідно до стану компонентів; це забезпечує більш швидку та зручну взаємодію користувачів з веб-додатком. На відміну від традиційних фреймворків, як Vue та Angular, React використовує html всередині JS, а точніше, JSX – мову розмітки, розширення синтаксису JS. Розміщення JSX поруч із відповідною логікою візуалізації дозволяє легко створювати, підтримувати та видаляти компоненти React [10]. Загалом, React простий для вивчення та зручний у використанні.

Використовуватиметься Bootstrap та React-bootstrap для стилів. Bootstrap – фреймворк для розробки веб-додатків, який надає зручний синтаксис для шаблонних дизайнів для швидкої побудови адаптивних веб-сторінок на основі

HTML, CSS та JS [11]. React-bootstrap - це бібліотека компонентів, які розроблені на основі Bootstrap, адаптована для використання з React [12].

Redux – бібліотека, призначена для керування станом JS-додатків; забезпечує передбачуваність та централізоване збереження стану [13]. React Redux – підтримувана розробниками Redux бібліотека, яка надає засоби для інтеграції Redux в React-додатки [14].

Самостійна реалізація діаграм послідовності завдань у проєктах зайняла б дуже багато часу та створила б такі додаткові кропіткі задачі, як красиве розміщення завдань за рівнями в правильній послідовності, перемальовування діаграми при додаванні та видаленні завдань та інші. Тому було прийняте рішення використати для цієї задачі сторонню бібліотеку GoJS. GoJS – JS-бібліотека для швидкої побудови інтерактивних діаграм в браузері на основі Canvas [15]. GoJS React – JS-бібліотека з компонентами React для GoJS-діаграм, розроблена для спрощення використання GoJS в React-додатках [16].

РОЗДІЛ 4. ОПИС РЕАЛІЗАЦІЇ

4.1. Реалізація серверної частини застосунку

4.1.1. Структура серверної частини

Структуру класів серверної частини розроблено відповідно до архітектури, зображеної на рис. 2.1 та описаної в розділі 2.3, з деякими додатковими класами (рис. 4.1). Пакет models містить моделі, repositories – інтерфейси репозиторіїв, services – сервіси (інтерфейси та імплементації), controllers – контроллери, mapstruct – DTOs та маппери. Пакет config призначений для класів-конфігурацій. Пакет exceptions містить класи-помилки для різних виняткових ситуацій та клас-обробник помилок GlobalExceptionHandler. Пакет security містить класи, що відповідають за налаштування безпеки застосунку. В пакеті utils розміщені допоміжні класи-утиліти.

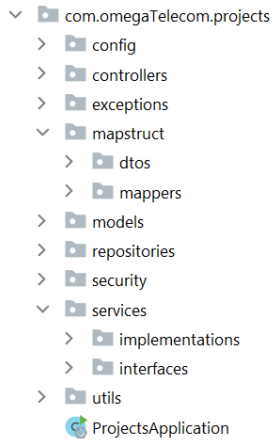

Рис. 4.1 Структура серверної частини

Розглянемо взаємозв’язок класів, які необхідні для створення нової посади. У класі-контроллері PositionController викликається метод addPosition(), який витягає з тіла http-запиту об’єкт класу-DTO PositionPostDto. За допомогою методу об’єкту автоматично згенерованого за інтерфейсом PositionMapper класу-маппера PositionMapperImpl, з positionPostDto створюється новий об’єкт класу-моделі Position, який передається в метод addPosition() об’єкту класу-сервісу PositionServiceImpl, що реалізовує інтерфейс PositionService (рис. 4.2).

```
@PostMapping
@PreAuthorize("hasAnyAuthority('ADMIN')")
public PositionGetDto addPosition(@Valid @RequestBody PositionPostDto positionPostDto){
    return positionMapper.positionToPositionGetDto(
            positionService.addPosition(
                    positionMapper.positionPostDtoToPosition(positionPostDto)));
}
```

Рис. 4.2 Метод addPosition() класу PositionController

У методі addPosition() сервісу за допомогою методів processString() та checkName() об'єкту класу Utils відбувається видалення зайвих пробілів із назви посади та перевіряється, чи назва не порожня. Далі відбувається пошук посади з такою самою назвою у відділі, куди потрібно додати нову посаду, за допомогою методу findByNameAndDepartment() об'єкту автоматично згенерованого за інтерфейсом PositionRepository класу-репозиторію PositionRepositoryImpl. Якщо така посада вже існує, викидається помилка PositionIllegalArgumentException із текстовим описом помилки, який зберігається як константа в класі Values. Також ця помилка, але вже з іншим текстовим описом, викидається, якщо посада, яку потрібно додати, є керівною, але у відділі вже є керівна посада. Якщо ж посада пройшла перевірку, вона зберігається в базі даних за допомогою методу save() репозиторію, а її новий об'єкт повертається назад в метод addPosition() контроллера (рис. 4.3). Там з об'єкту за допомогою маппера створюється новий об'єкт класу-DTO PositionGetDto та повертається з методу (рис. 4.2). Схожим чином оброблюються інші запити, що стосуються посади та решти моделей.

```
@Override
public Position addPosition(Position position) {
    String name = utils.processString(position.getName());
    utils.checkName(name);
    Optional<Position> positionWithSuchNameInSameDepartment =
            positionRepository.findByNameAndDepartment(name, position.getDepartment().getId());
    if (positionWithSuchNameInSameDepartment.isPresent())
        throw new PositionIllegalArgumentException(Values.POSITION_WITH_SUCH_NAME_ALREADY_EXISTS);
    position.setName(name);
    if (position.isHead()) {
        for (Position pos: getAll()) {
            if (pos.isHead() && pos.getDepartment().equals(position.getDepartment()))
                throw new PositionIllegalArgumentException(
                        Values.HEAD_POSITION_ALREADY_EXISTS_IN_THIS_DEPARTMENT);
        }
    }
    return positionRepository.save(position);
}
```

Рис. 4.3 Метод addPosition() класу PositionService

4.1.2. Безпека

Перед тим, як запит потрапить на обробку у відповідний контроллер, він має пройти ланцюжок фільтрів для аутентифікації та авторизації. Конфігурація цього ланцюжка відбувається в класі SecurityConfig, в методі filterChain(), де власна реалізація фільтру з JWT-токеном – JwtAuthorizationFilter встановлюється в ланцюжок фільтрів Spring Security перед UsernamePasswordAuthenticationFilter (рис. 4.4).

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf().disable();
    http.headers().frameOptions().disable();
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.exceptionHandling().accessDeniedHandler((
            (request, response, accessDeniedException) -> ResponseHelper.setResponse(response,
                    FORBIDDEN.value(), new ApiErrorResponse(accessDeniedException.getMessage())));
    http.addFilterBefore(new JwtAuthorizationFilter(jwtConfig.getSecret()),
            UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
```

Рис. 4.4 Метод filterChain() класу SecurityConfig

Для того, щоб фільтр JwtAuthorizationFilter викликався лише один раз для кожного запиту, він наслідується від абстрактного класу OncePerRequestFilter та реалізовує метод doFilterInternal() [17], де обробляється JWT-токен із заголовка Authorization запиту (рис. 4.5). JWT (JSON Web Token) - це стандарт, який визначає безпечний та компактний спосіб передачі даних разом з підписом між двома сторонами. Токен верифікується за допомогою алгоритму HMAC256 та декодується (процес аутентифікації) [18], після чого можна провести безпосередньо процес авторизації – перевірити, що користувачу з його роллю дозволено виконувати цей запит (дозволені ролі для кожного запиту визначаються за допомогою анотації @PreAuthorize над методами контролерів, як на рис. 4.2). Якщо авторизація пройшла успішно, створюється об’єкт класу UsernamePasswordAuthenticationToken та встановлюється у поточний SecurityContext, що використовується фреймворком для зберігання інформації про поточного користувача [19]; викликається наступний фільтр з ланцюжка.

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
                                FilterChain filterChain) throws IOException, ServletException {
    String token = request.getHeader(HttpHeaders.AUTHORIZATION);
    if (token != null) {
        try {
            Algorithm algorithm = Algorithm.HMAC256(SECRET.getBytes());
            JWTVerifier verifier = JWT.require(algorithm).build();
            DecodedJWT decodedJWT = verifier.verify(token);
            String username = decodedJWT.getSubject();
            Collection<GrantedAuthority> roles = List.of(
                    new SimpleGrantedAuthority(decodedJWT.getClaim( s: "role").asString()));
            UsernamePasswordAuthenticationToken authenticationToken =
                    new UsernamePasswordAuthenticationToken(username, credentials: null, roles);
            SecurityContextHolder.getContext().setAuthentication(authenticationToken);
            filterChain.doFilter(request, response);
        } catch (JWTVerificationException exception) {
            ResponseHelper.setResponse(response, FORBIDDEN.value(),
                    new ApiErrorResponse(Values.TOKEN_VALIDATION_FAILED));
        }
    } else {
        filterChain.doFilter(request, response);
    }
}
```

Рис. 4.5 Метод doFilterInternal() класу JwtAuthorizationFilter

Один ендпойнт (endpoint) серверу не потребує відправлення з JWT-токеном. Це ендпойт login, призначений якраз для того, щоб користувач отримав новий JWT-токен за електронною поштою та паролем для виконання інших запитів. Відбувається перевірка, чи користувач з такою поштою та паролем існує в системі (порівняння паролю за допомогою методу matches() об’єкту класу BCryptPasswordEncoder, оскільки в базі даних зберігається зашифрований пароль користувача). Якщо такий користувач існує, за допомогою методу generateToken() об’єкту класу JwtGenerator генерується JWT-токен [18], використовуючи електронну пошту, ПІБ та роль користувача та повертається як відповідь ендпойнту (рис. 4.6).

```
public String generateToken(Worker user) {
    try {
        long nowMillis = System.currentTimeMillis();
        return JWT.create()
                .withSubject(user.getContacts().getEmail())
                .withClaim( name: "pib", user.getPib().toString())
                .withClaim( name: "role", user.getRole().toString())
                .withIssuedAt(new Date(nowMillis))
                .withExpiresAt(new Date(nowMillis + JWT_TTL_MILLIS))
                .sign(Algorithm.HMAC256(config.getSecret()));
    } catch (JWTCreationException exception) {
        throw new RuntimeException(Values.YOU_NEED_TO_ALLOW_ENCRYPTION_ALGORITHM);
    }
}
```

Рис. 4.6 Метод generateToken() класу JwtGenerator

В системі відсутній ендпойнт для реєстрації, оскільки всіх користувачів-працівників додає адміністратор (який додається вручну до бази). При додаванні нового працівника на сервері генерується випадковий пароль довжиною 8 символів, який записується в базу в зашифрованому вигляді.

4.1.3. Відправка електронних листів

Перед збереженням в базу даних нового користувача із зашифрованим паролем, на його електронну пошту надсилається лист-сповіщення про створення аккаунту, що також містить пароль, щоб користувач міг увійти в систему. Для цього використовується метод sendEmail() об'єкту класу EmailServiceImpl, що реалізовує інтерфейс EmailService (рис. 4.7). В методі створюється повідомлення – об'єкт класу SimpleMailMessage, ініціалізуються його поля відправник, отримувач, тема та текст листа і викликається метод send() об'єкту класу JavaMailSender для відправки цього повідомлення [20].

```
@Override
public void sendEmail(MessageForEmail messageForEmail) {
    SimpleMailMessage message = new SimpleMailMessage();
    String from = messageForEmail.getFrom() == null ? defaultFrom : messageForEmail.getFrom();
    message.setFrom(from);
    message.setTo(messageForEmail.getTo());
    message.setSubject(messageForEmail.getSubject());
    message.setText(messageForEmail.getText());
    mailSender.send(message);
}
```

Рис. 4.7 Метод sendEmail() класу EmailServiceImpl

4.1.4. API

Нижче наведено реалізовані API ендпойнти серверної частини застосунку.

Облікові записи:

- POST /api/login
- PUT /api/password – зміна паролю поточного користувача.

Працівники:

- GET /api/workers
- GET /api/workers/:id

- GET /api/workers/me – отримання інформації про поточного користувача.
- POST /api/workers
- PUT /api/workers/:id
- DELETE /api/workers/:id

Відділи:

- GET /api/departments
- GET /api/departments/:id
- POST /api/departments
- PUT /api/departments/:id
- DELETE /api/departments/:id

Посади:

- GET /api/positions
- GET /api/positions/:id
- POST /api/positions
- PUT /api/positions/:id
- DELETE /api/positions/:id

Компанії-контрагенти:

- GET /api/partners
- GET /api/partners/:id
- POST /api/partners
- PUT /api/partners/:id
- DELETE /api/partners/:id

Завдання:

- PATCH /api/tasks/:id/notes – оновлення нотаток завдання;
- PATCH /api/tasks/:id/performer – призначення нового виконавця завдання;
- POST /api/tasks/:id/finish – завершення виконання завдання типу звичайне виконання;
- POST /api/tasks/:id/approve_previous – підтвердження прийому попередніх завдань (завершення виконання завдання типу підтвердження виконання попередніх завдань);
- POST /api/tasks/:id/reject_previous – відхилення прийому попередніх завдань (завершення виконання завдання типу підтвердження виконання попередніх завдань).

Проєкти:

- GET /api/flows/all_allowed – отримання всіх доступних для користувача проєктів (розділ 2.2);
- GET /api/flows/:id
- POST /api/flows
- PUT /api/flows/:id
- DELETE /api/flows/:id
- POST /api/flows/start – початок виконання проєкту;
- POST /api/flows/fail – позначення проєкту як провалений.

Звіти:

- GET /api/reports/workers/generate – генерація звіту у форматі pdf по роботі працівників в проміжок часу між початковою та кінцевою датами, що задаються параметрами;

- GET /api/reports/flows/generate – генерація звіту у форматі pdf по виконанню проєктів в проміжок часу між початковою та кінцевою датами, що задаються параметрами;
- GET /api/reports/workers – отримання раніше згенерованого звіту (файлу у форматі pdf) по роботі працівників;
- GET /api/reports/flows – отримання раніше згенерованого звіту (файлу у форматі pdf) по виконанню проєктів.

4.2. Реалізація клієнтської частини застосунку

4.2.1. Структура клієнтської частини

Структуру файлів клієнтської частини зображено на рис. 4.8. Основним файлом є index.js, в ньому створюється компонент App зі шляхами. На тому самому рівні файлової структури розміщені однойменні файли стилів. В папці components зберігаються всі JSX-компоненти додатку, згруповані по папках за призначенням. Наприклад, в папці common знаходяться допоміжні компоненти, які використовуються в декількох більших компонентах, як-от: Input для введення значень текстових полів, SearchBar для пошуку та Loader, що показується під час завантаження списку об’єктів з бекенду. Іншим прикладом може слугувати папка Workers, вміст якої схожий на вміст інших папок, що відповідають компонентам для певної сутності предметної області. В папці Workers знаходяться компоненти для роботи з сутністю Працівник: AddWorkerModal для додавання нового працівника, UpdateWorkerModal для редагування інформації про працівника, Workers для відображення списку працівників та WorkerItem як елемент цього списку. Папка services зберігає сервіси для кожної сутності із запитами до серверної частини. Папка store відповідає за роботу зі станом додатку. В папці utils зберігаються константи та допоміжні функції-утиліти.

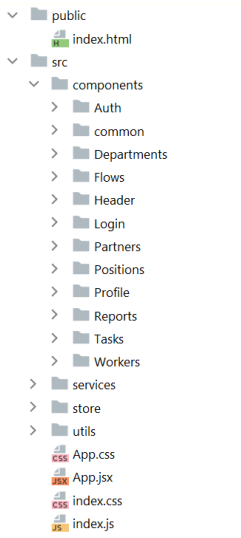

Рис. 4.8 Структура клієнтської частини

4.2.2. Діаграми GoJS для інтерактивного відображення завдань проєкту

Діаграми GoJS містять та відображають частини (parts) – ноди (nodes) і зв'язки (links), які є візуалізацією (view) даних, що зберігаються та управляються моделлю (model). Тому GoJS має model-view архітектуру (рис. 4.9) [21]. Дані в моделі – це масиви JS-об'єктів, що мають необхідні для бізнес-логіки властивості, тобто є можливість динамічно змінювати дані в моделі, наприклад, при зміні стану компоненту React, і отримувати автоматичну перемальовку діаграми. В моделі ноди зберігаються в масиві nodeDataArray, а зв'язки – в масиві linkDataArray.

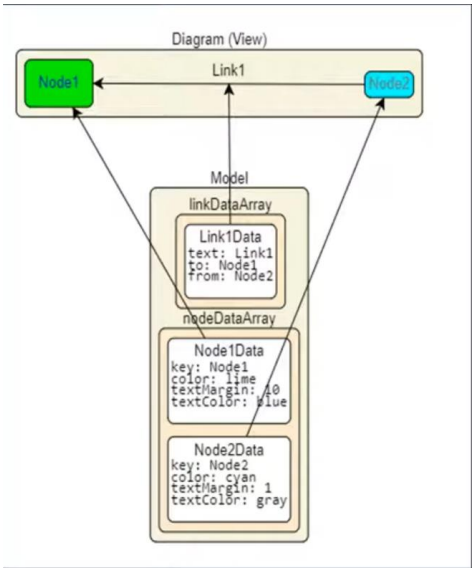

Рис. 4.9 Model-view архітектура бібліотеки GoJS

Діаграма – це JS-об’єкт в пам’яті, що асоціюється з div-елементом HTML точно встановленого розміру. У клієнті, написаному на чистому JS, діаграма створюється так: go.GraphObject.make(go.Diagram, <div-елемент>), але завдяки бібліотеці react-gojs, немає потреби вручну створювати div-елемент. Замість цього можна додати компонент ReactDiagram, передавши туди параметрами всі необхідні налаштування діаграми (рис. 4.10).

```
<ReactDiagram
    ref={this.diagramRef}
    divClassName='diagram-component'
    style={this.diagramStyle}
    initDiagram={this.initDiagram}
    nodeDataArray={this.props.nodeDataArray}
    linkDataArray={this.props.linkDataArray}
    modelData={{canRelink: false}}
    skipsDiagramUpdate={this.props.skipsDiagramUpdate}
/>
```

Рис. 4.10 Компонент ReactDiagram

Бібліотека GoJS надає можливість створювати діаграми з різними типами моделей та схемами розміщення елементів (layouts). Обрано модель GraphLinksModel, яка передбачає ноди та зв’язки, що містять поля from і to з ключами початкової та кінцевої ноди відповідно. Обрано layout ForceDirectedLayout для того, щоб наступні завдання-ноди автоматично розміщувалися нижче, ніж їх попередні, та щоб завдання розміщувалися не хаотично, а чіткою структурою по горизонтальних рівнях.

Для створення необхідних для ініціалізації компоненту ReactDiagram функцій та методів створено клас-обгортку – компонент DiagramWrapper [22]. В ньому створюються шаблони, що визначають стиль нод і зв’язків. В шаблоні ноди задається, що нода складається з основної панелі Panel та кнопки Shape форми PlusLine для додавання наступного завдання (рис. 4.11).

```
diagram.nodeTemplate =
    $(go.Node, 'Spot',
        {selectionObjectName: "BODY"...},
        $(go.Panel, 'Auto', {name: "BODY"},
            $(go.Shape, 'RoundedRectangle',
                {name: 'SHAPE'...},
            ),
            $(go.Panel, "Table",
                {...},
                $(go.RowColumnDefinition,
                    {alignment: go.Spot.Left...}),
                textTemplate( {sourceprop, row, column, columnSpan, isTextBig, transformTextFunc): {
                    sourceprop: 'id', row: 0, column: 0}),
                textTemplate( {sourceprop, row, column, columnSpan, isTextBig, transformTextFunc): {
                    sourceprop: 'department', transformTextFunc: (department) => department.name,
                    row: 4, column: 0, columnSpan: 2}),
                /* ... more textTemplate() calls */
            ),
        ),
        $("Button",
            $(go.Shape, "PlusLine", {width: 10, height: 10}),
            {name: "ADD_TASK_BTN"...}
        ));
```

Рис. 4.11 Шаблон нод-завдань для GoJS діаграми

Панель є заокругленим прямокутником RoundedRectangle та містить таблицю Table із комірками для тексту з властивостями завдання, стиль кожної з яких задається за допомогою функції textTemplate() (рис. 4.12). В цій функції, окрім стилю, що задається масивом properties, текстовому блоку TextBlock встановлюється прив’язка Binding до певної властивості завдання sourceprop. Щоб при зміні значення цієї властивості у завдання, зміни відображались на ноді, трансформуючи значення за допомогою функції transformTextFunc().

```
const textTemplate = ({sourceprop, row, column, columnSpan, isTextBig :boolean = false,
                       transformTextFunc = (text) => text}) => {
    let font = isTextBig ? 'bold 1rem' : '.9rem';
    const properties = {margin: 4, row: row, column: column, editable: false,
        font: font + ' Roboto, sans-serif'};
    if (columnSpan)
        properties.columnSpan = columnSpan;
    return $(go.TextBlock,
        properties,
        new go.Binding( targetprop: 'text', sourceprop, conv: t => {
            return transformTextFunc(t);
        })
    );
}
```

Рис. 4.12 Шаблон відображення текстового атрибуту на ноді в GoJS діаграмі

В компоненті ReactDiagram також створюються контекстні меню для нод і зв’язків та обробники подій для діаграми, які задають реакцію застосунку на взаємодію користувача з діаграмою – подвійний клік на об’єкті

ObjectDoubleClicked та додавання зв’язку LinkDrawn (рис. 4.13). Окрім цього, в компоненті ReactDiagram визначаються властивості діаграми, наприклад, що переміщати та копіювати ноди заборонено.

```
componentDidMount() {
    const diagram = this.diagramRef.current.getDiagram();
    if (diagram instanceof go.Diagram) {
        diagram.addDiagramListener( name: 'LinkDrawn', this.handleDiagramEvent);
        diagram.addDiagramListener( name: 'ObjectDoubleClicked', this.handleDiagramEvent);
    }
}

handleDiagramEvent (e) {
    const name = e.name;
    switch (name) {
        case 'LinkDrawn': {
            console.log('LinkDrawn');
            this.props.onLinkDrawn(e.subject);
            break;
        }
        case 'ObjectDoubleClicked': {
            this.props.onNodeTaskDoubleClicked(e.subject);
            break;
        }
        default: break;
    }
}
```

Рис. 4.13 Обробники подій для GoJS діаграми

Компонент DiagramWrapper створюється в компонентах FlowProfile та CreateFlow для інтерактивного відображення послідовностей завдань існуючих проєктів та проєкту при створенні. Із цих двох компонентів в компонент DiagramWrapper передаються масиви завдань та зв’язків між завданнями, які зберігаються як стан (state) компонентів та динамічно змінюються. Динамічно змінюються як масиви (дані моделі), так і графічні елементи на самій діаграмі при кожній зміні, внесеній до послідовності завдань. Наприклад, ці два методи використовуються для додавання до діаграми нового завдання та нового зв’язку відповідно: diagram.model.addNodeData(task), diagram.model.addLinkData(link).

4.2.3. Механізм store

Для забезпечення централізованого збереження та регламентованої зміни стану застосунку використовується механізм store, що надається бібліотекою Redux. Механізм дозволяє використовувати спільні дані в різних компонентах застосунку замість того, щоб завантажувати їх кожен раз з серверу. Store – це

контейнер, який зберігає глобальний стан застосунку (state) [21]. Для зручності роботи взаємодія зі state розподіляється на логічні частини; створюються папки, що називаються так само, як ці частини, і кожна з яких містить набір файлів, назви та призначення яких дублюється в кожній папці. Щоб отримати доступ до частини state, використовуються функції-селектори: const getWorkers = (state) => state.workers.entities, що в свою чергу використовуються в компоненті React за допомогою хука useSelector(): const workers = useSelector(getWorkers). Зміна state відбувається за допомогою функції reducer, що комбінується з окремих менших функцій-reducers.

Розглянемо детальніше роботу з частиною state – працівниками (workers); взаємодія з іншими частинами state відбувається схожим чином. Той самий список всіх працівників потрібен в багатьох компонентах React – Workers, FlowProfile, TaskProfile, AddUpdateTaskModal та інших, тому він зберігається в глобальному state. Щоб спричинити оновлення стану, потрібно створити JS-об’єкт, що описує дію, яка відбулась в застосунку – action [21]. Цей об’єкт створюється за допомогою функції з файлу actionCreators.js. Наприклад, для додавання нового працівника це буде функція workerAdded() (рис. 4.14).

```
export const workerAdded = (worker) => ({
    type: ADD_WORKER,
    payload: worker,
});
```

Рис. 4.14 Функція workerAdded()

Типи дій, доступні для workers, зберігаються в файлі actionTypes.js – отримати список працівників, додати, оновити, видалити працівника, почати завантажувати список працівників, отримати помилку при завантаженні, сповістити про потребу перезавантаження списку працівників (рис. 4.15).

```
export const GET_WORKERS = 'workers/workersGet';
export const ADD_WORKER = 'workers/workerAdded';
export const UPDATE_WORKER = 'workers/workerUpdated';
export const DELETE_WORKER = 'workers/workerDeleted';
export const START_LOADING_WORKERS = 'workers/workersStartedLoading';
export const ERROR_WHILE_LOADING_WORKERS = 'workers/workersGotErrorWhileLoading';
export const NEED_RELOAD_WORKERS = 'workers/workersNeedReload';
```

Рис. 4.15 ActionTypes для працівників

Об’єкт action необхідно відправити за допомогою функції dispatch() до store, де за допомогою кореневого reducer та workersReducer з файлу reducer.js (рис. 4.16) обраховується наступний state на основі попереднього стану та action. Store сповіщає підписників, що state було оновлено, щоб інтерфейс користувача оновився відповідно до нових даних [23].

```
export default function workersReducer(state : {entities: any[], gotData: number} = workersInitialState, action) {
    switch (action.type) {
        case START_LOADING_WORKERS:
            return {...state, gotData: STARTED_LOADING}
        case ERROR_WHILE_LOADING_WORKERS:...
        case NEED_RELOAD_WORKERS:...
        case GET_WORKERS:
            return {gotData: LOADED_SUCCESSFULLY, entities: action.payload}
        case ADD_WORKER:
            return {...state, entities: [...state.entities, action.payload]}
        case UPDATE_WORKER: {
            const { id } = action.payload;
            return {...state, entities: state.entities.map((worker) => {
                    if (worker.id !== id) return worker;
                    return action.payload;})};
        }
        case DELETE_WORKER:
            return {...state, entities: state.entities.filter((worker) => worker.id !== action.payload)}
        default:
            return state;
    }
}
```

Рис. 4.16 Функція workersReducer()

Допоміжним кроком в механізмі store є функції з роллю thunk, що зберігаються в однойменному файлі – middleware для асинхронних функцій. Вони об’єднують в собі отримання даних з серверу і сповіщення про це store. Наприклад, функція saveNewWorker() повертає асинхронну функцію saveNewWorkerThunk(), де викликається асинхронна функція addWorker() сервісу workerService для відправки нового працівника на сервер і збереження в базі (рис. 4.17). Залежно від відповіді сервера, store сповіщається про помилку чи успішне додавання нового працівника за допомогою функції dispatch(). Виклик функції saveNewWorker() з компоненту React відбувається так, як звичайна відправка action: dispatch(saveNewWorker(worker)).

```
export function saveNewWorker(worker) {
    return async function saveNewWorkerThunk(dispatch) {
        const response = await addWorker(worker);
        if (response.success === "false") {
            dispatch(generalErrorSet(response.error));
        } else {
            dispatch(workerAdded(response));
        }
    };
}
```

Рис. 4.17 Функція saveNewWorker()

4.3. Інтерфейс та користувацький досвід

Розробка інтерфейсу проводилася з розрахунку зручного користування, включно з адаптивністю до різних розмірів екранів, із використанням спокійного синього кольору як основного.

Сторінка логіну містить форму для вводу електронної пошти та паролю користувача (рис. 4.18). Введені в поле паролю символи змінені з міркувань безпеки.

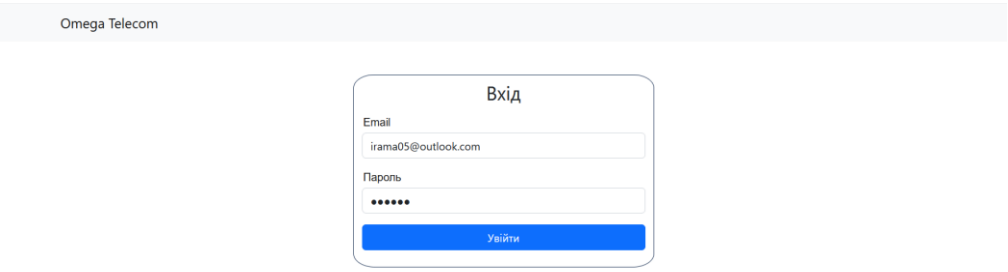

Рис. 4.18 Сторінка логіну

Якщо були введені коректні дані, користувач переходить на сторінку із доступними завданнями (рис. 4.19). Для завдань наявний пошук та фільтри за станом прогресу (Нове, В процесі, Виконано, Провалено) та рівнем відповідальності користувача (Я виконавець, Моя відповідальність, Мною створено, Завдання мого відділу, Завдання з проєкту під моєю відповідальністю). Кожне завдання як елемент списку відображає базову інформацію та має кнопку з посиланням на сторінку з детальною інформацією про нього.

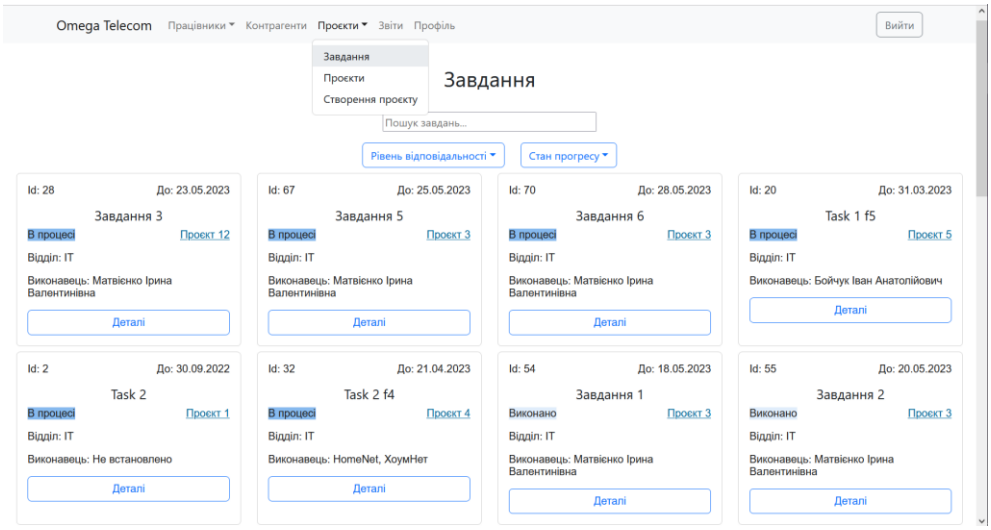

Рис. 4.19 Сторінка із завданнями

Також згори можна побачити меню з кнопкою Вийти та посиланнями на сторінки проєктів, створення проєкту, працівників, компаній-контрагентів, профілю; сторінки відділів з посадами та звітів доступні тільки для користувача з роллю адміністратора.

На сторінці детальної інформації окремого завдання (рис. 4.20) можна додати нотатки, встановити нового виконавця завдання (лише для завдання, виконання якого ще не розпочалося) та залежно від типу завдання завершити виконання завдання або підтвердити/відхилити виконання попередніх завдань за допомогою кнопок (лише якщо завдання в прогресі та користувачу це дозволено за вимогами, розділ 2.2).

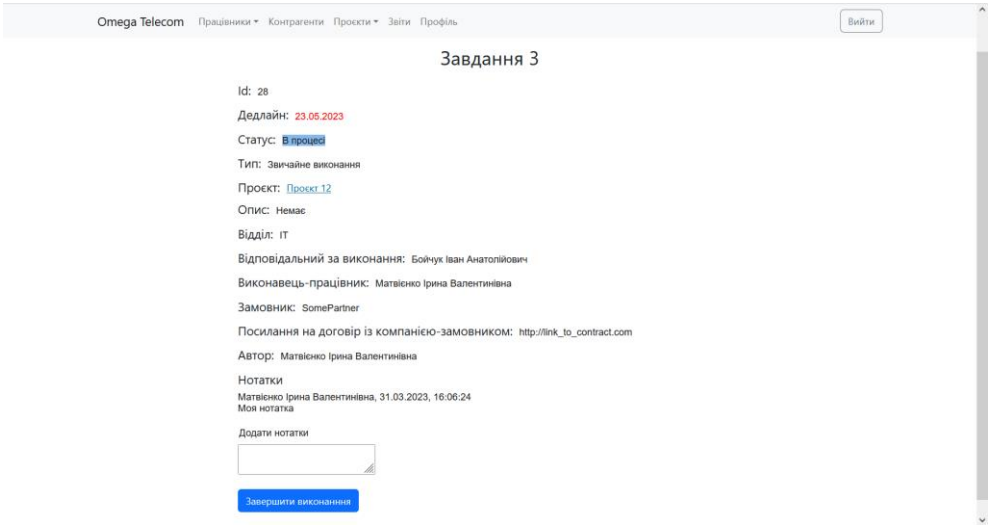

Рис. 4.20 Сторінка детальної інформації окремого завдання

З меню можна перейти на сторінку з доступними проєктами (рис. 4.21). Для проєктів наявний пошук та фільтри за станом прогресу (Новий, В процесі, Виконано, Провалено) та рівнем відповідальності користувача (Моя відповідальність, Мною створено, Інші). Кожен проєкт як елемент списку відображає базову інформацію та має кнопку з посиланням на сторінку з детальною інформацією про нього. В лівому верхньому куті є кнопка для переходу на сторінку додавання нового проєкту.

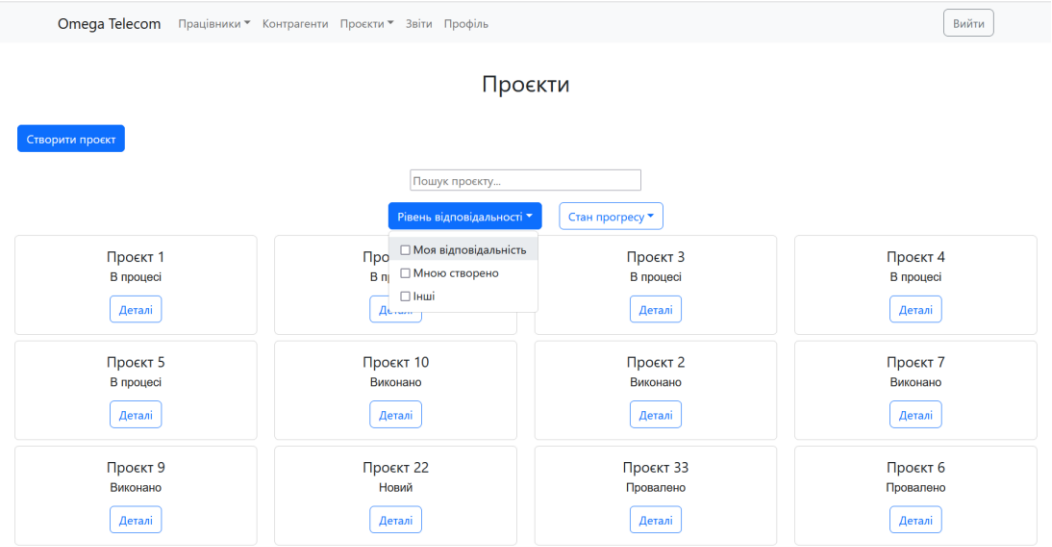

Рис. 4.21 Сторінка з проєктами

На сторінці детальної інформації окремого проєкту розміщено діаграму із послідовністю завдань проєкту (рис. 4.22). Те, що завдання А обов'язково має виконуватися раніше, ніж завдання В, позначено зв'язком-стрілкою від завдання А до завдання В. За допомогою подвійного кліку лівою кнопкою миші на завданні в діаграмі можна перейти на сторінку цього завдання, якщо сторінка не знаходиться в режимі редагування (рис. 4.20). Наступні дії можна виконувати лише якщо ввімкнено режим редагування за допомогою перемикача. За допомогою кліку правою кнопкою миші на завданні в діаграмі відкривається меню для вибору дій, які можна виконати над завданням (лише якщо воно нове і якщо користувачу це дозволено за вимогами, розділ 2.2): відредагувати за допомогою форми в діалоговому вікні (рис. 4.23), видалити чи додати наступне завдання за допомогою форми (рис. 4.23). Наступне

завдання також можна додати натиснувши на кнопку у формі плюса в правій частині завдання на діаграмі. Якщо після завдання А звичайного типу є хоч одне завдання-підтвердження, то всі інші наступні завдання для завдання А теж мають бути підтвердженнями. Після завдання-підтвердження можуть бути лише завдання звичайного типу. Для того, щоб додати новий зв'язок між двома завданнями, потрібно натиснути лівою кнопкою миші з курсором на першому завданні та, не відпускаючи кнопку, протягнути курсор до другого завдання, додаючи таким чином стрілку. Для видалення зв'язку потрібно натиснути на нього правою кнопкою миші та натиснути кнопку Видалити, що з'явиться (якщо у завдання лише один зв'язок із попереднім завданням, його видаляти заборонено). Відредагувавши послідовність завдань та, можливо, змінивши назву проєкту, потрібно натиснути на кнопку Оновити під діаграмою для збереження змін. Поруч є кнопки Видалити та одна з двох кнопок Розпочати виконання або Провалити виконання, що виконують відповідні дії над проєктом.

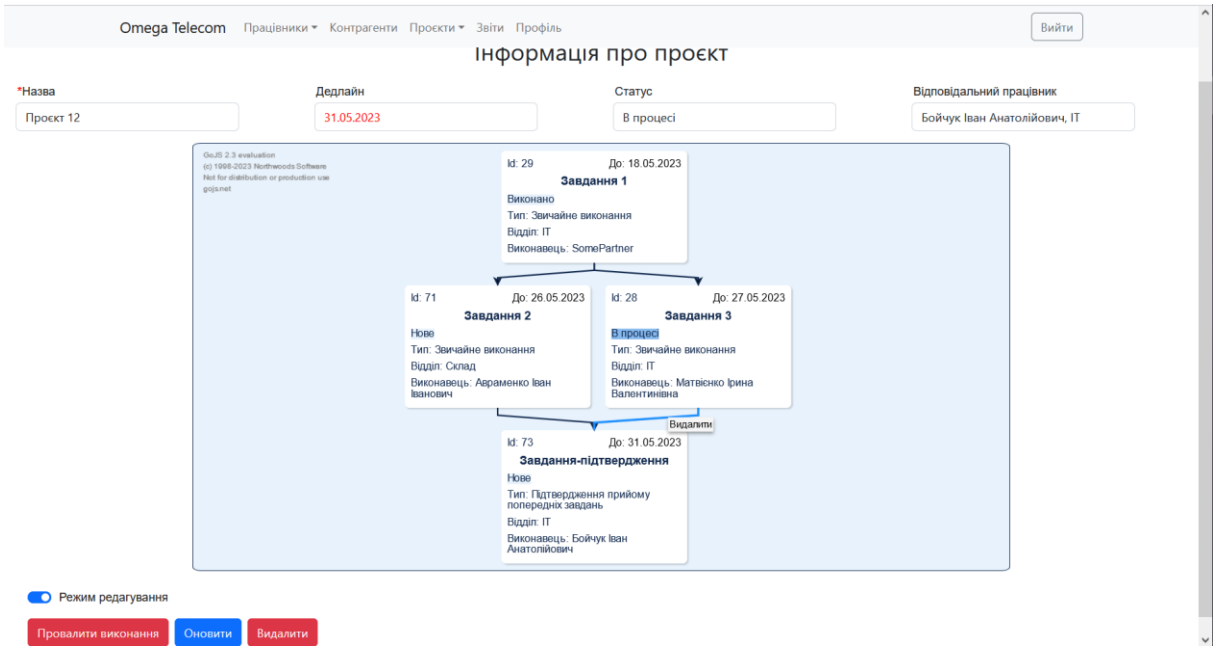

Рис. 4.22 Сторінка детальної інформації окремого проєкту

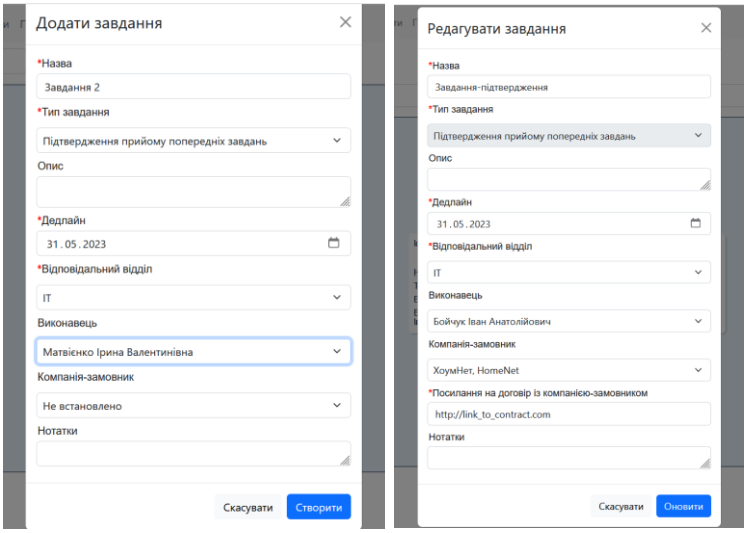

Рис. 4.23 Форми додавання та редагування завдання

На сторінці створення нового проєкту потрібно ввести назву проєкту, а також додати послідовність завдань за допомогою діаграми (рис. 4.24). Щоб додати перше завдання, потрібно натиснути кнопку під діаграмою Додати завдання, далі заповнити необхідні поля форми в діалоговому вікні (рис. 4.23) та натиснути кнопку Створити. Додавши всі потрібні завдання і зв’язки, треба натиснути кнопку Зберегти, щоб новий проєкт зберігся в базі даних і відбувся перехід на сторінку з проєктами (рис. 4.21).

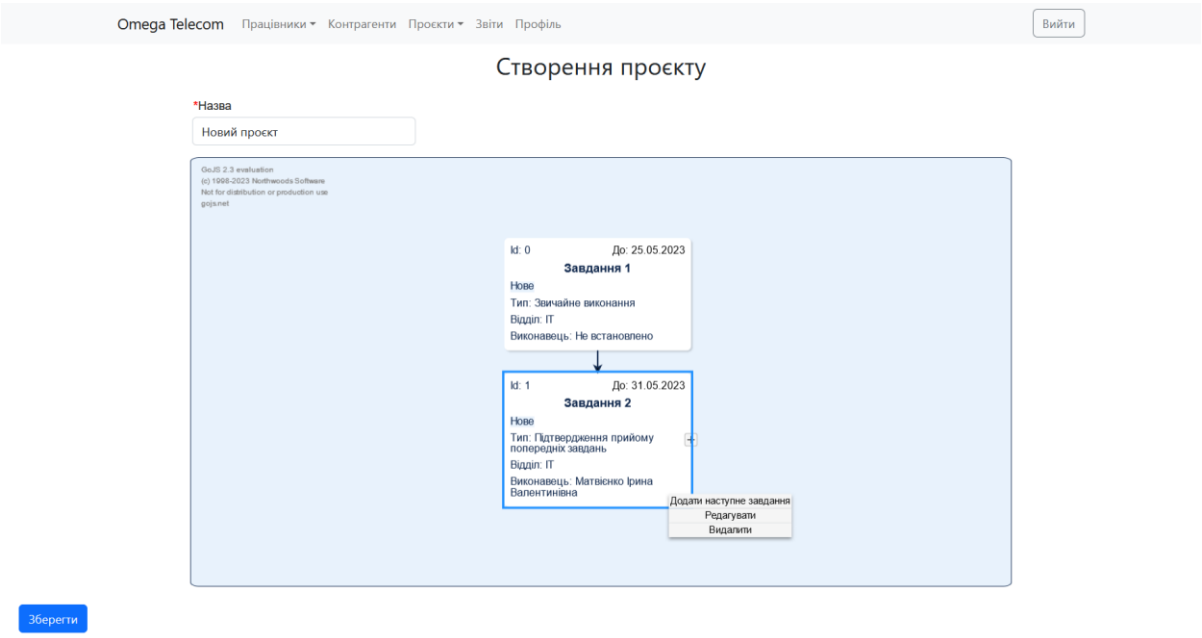

Рис. 4.24 Сторінка створення нового проєкту

З меню можна перейти на сторінку зі списком працівників компанії, для якого доступний пошук та фільтрація за відділом (рис. 4.25). Користувач з роллю адміністратор також може додавати та редагувати інформацію про працівників за допомогою форм (рис. 4.26). При додаванні нового працівника, йому на вказаний email приходить лист зі згенерованим паролем та рекомендацією якнайшвидше його змінити.

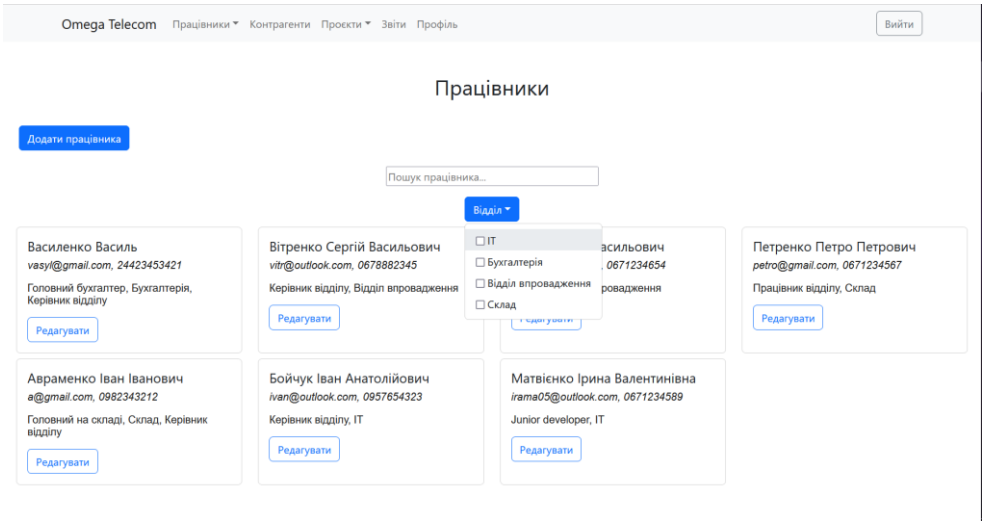

Рис. 4.25 Сторінка зі списком працівників

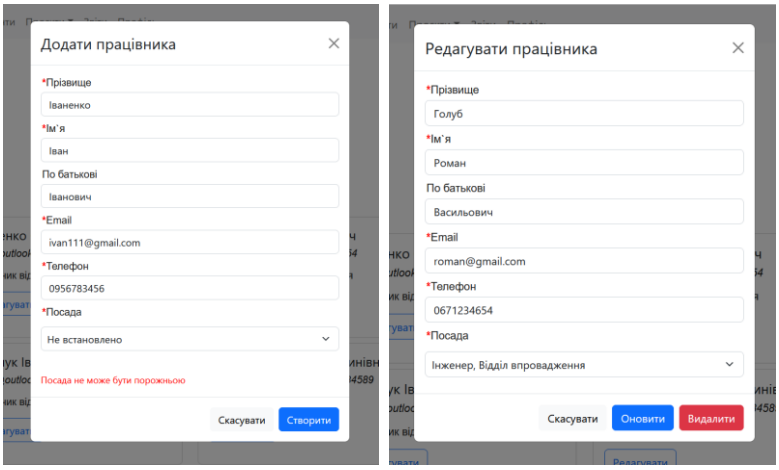

Рис. 4.26 Форми додавання та редагування працівника

З меню також можна перейти на сторінку зі списком компаній-контрагентів, для якого доступний пошук (рис. 4.27). Користувач з роллю адміністратор може додавати та редагувати інформацію про компанії-контрагентів, за допомогою форм, включно з можливістю додавання/редагування/видалення декількох назв для компанії (рис. 4.28).

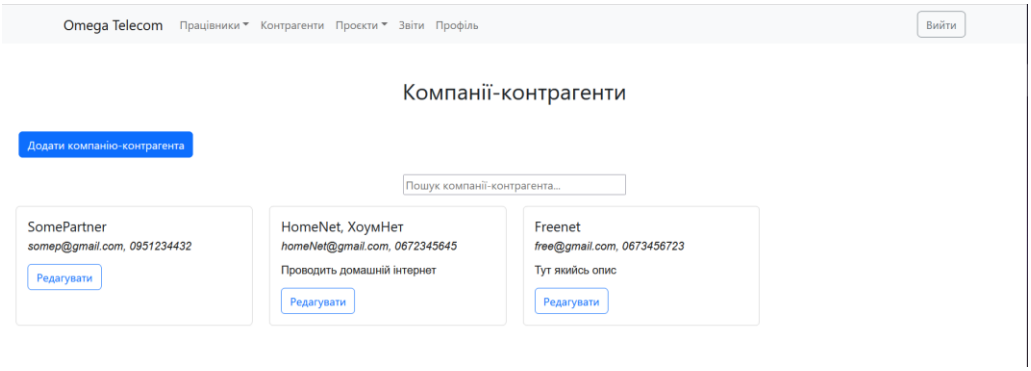

Рис. 4.27 Сторінка зі списком компаній-контрагентів

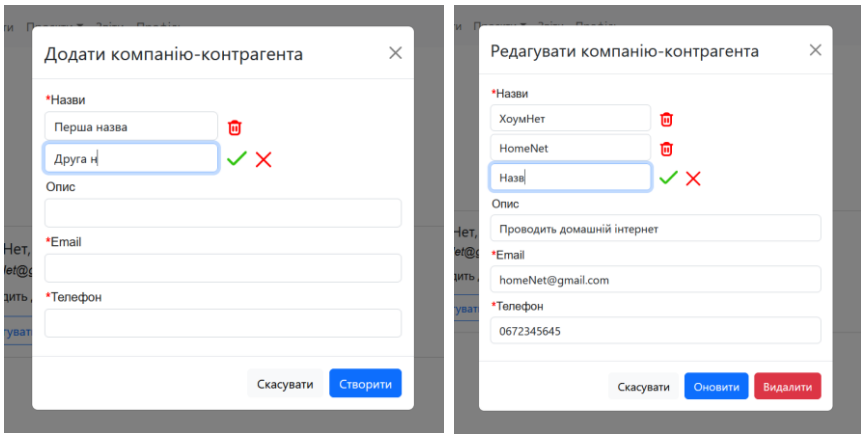

Рис. 4.28 Форми додавання та редагування компанії-контрагента

На сторінці профілю користувач може переглянути інформацію про себе, а також змінити пароль, ввівши існуючий пароль та новий пароль двічі (рис. 4.29).

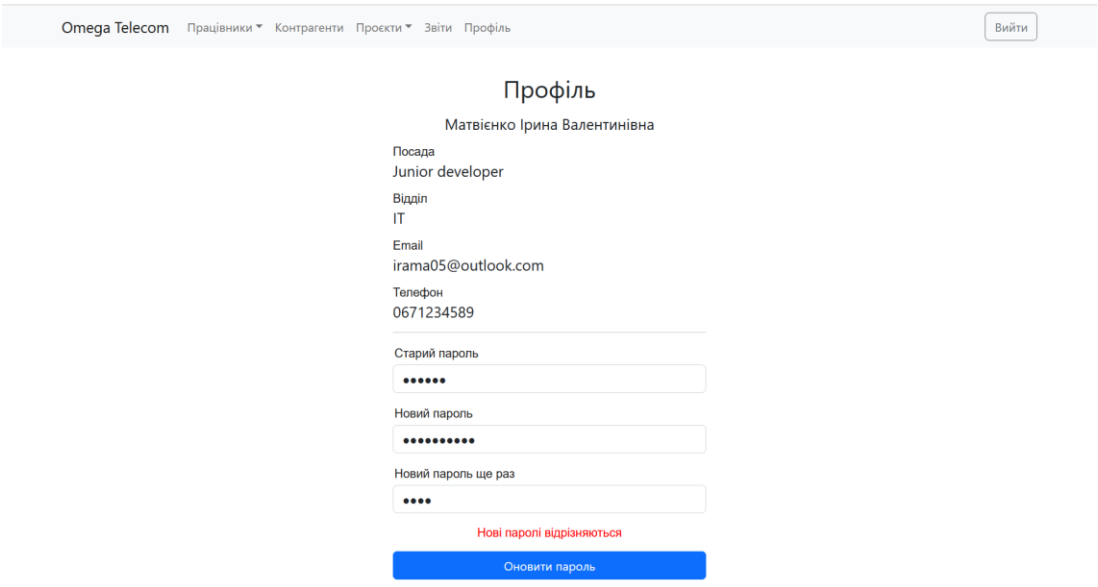

Рис. 4.29 Профіль користувача

Адміністратор з меню може перейти на сторінку зі списком посад, які згруповані за відділами (рис. 4.30). Тут він може додавати новий відділ, редагувати назви відділів та видаляти відділи, а також додавати до відділу нову посаду (у відділі може бути лише одна керівна посада), редагувати назви посад та видаляти посади, якщо їх не обіймають працівники.

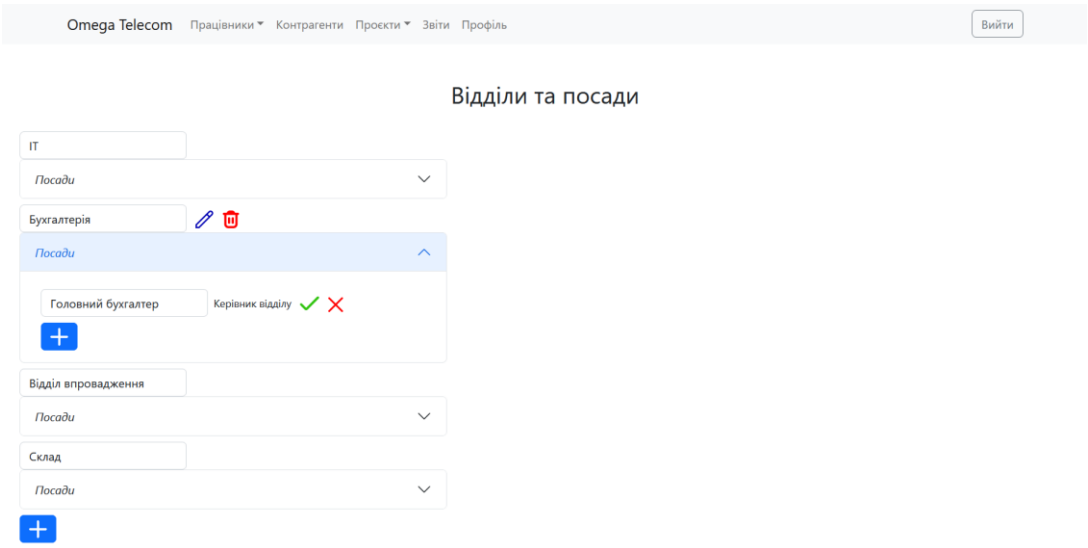

Рис. 4.30 Сторінка з відділами та посадами

На сторінці зі звітами адміністратор може згенерувати та переглянути звіти по роботі всіх працівників та виконанню проєктів за весь час роботи або за певний період, ввівши дати початку та закінчення періоду та натиснувши на відповідні кнопки (рис. 4.31).

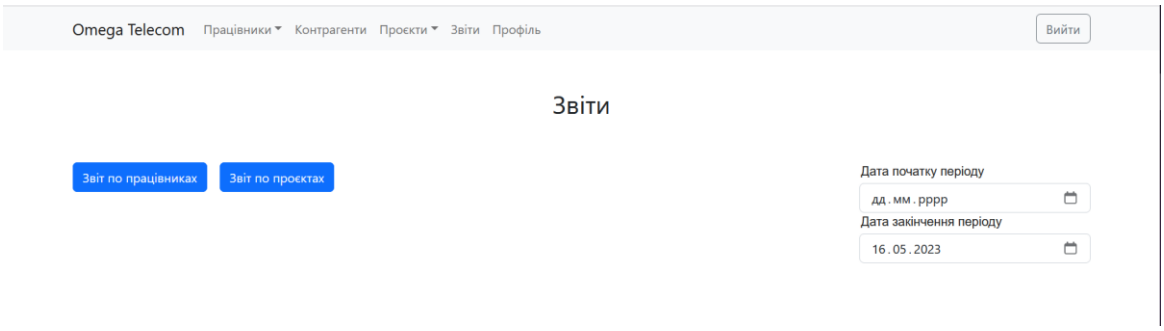

Рис. 4.31 Сторінка для генерації звітів

Звіти відкриваються на окремих сторінках, їх можна завантажити у форматі pdf та надрукувати (рис. 4.32-4.33).

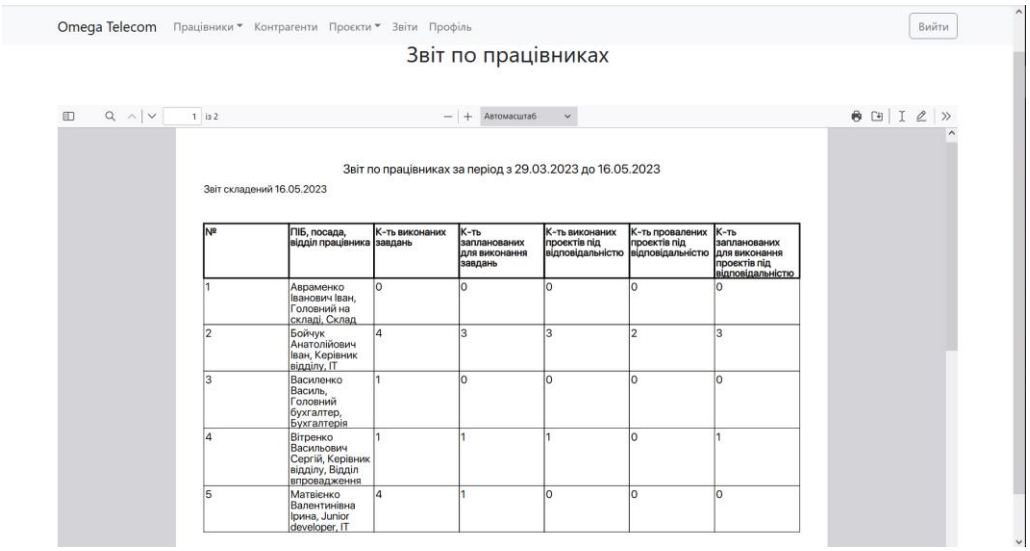

Рис. 4.32 Сторінка зі звітом по працівниках

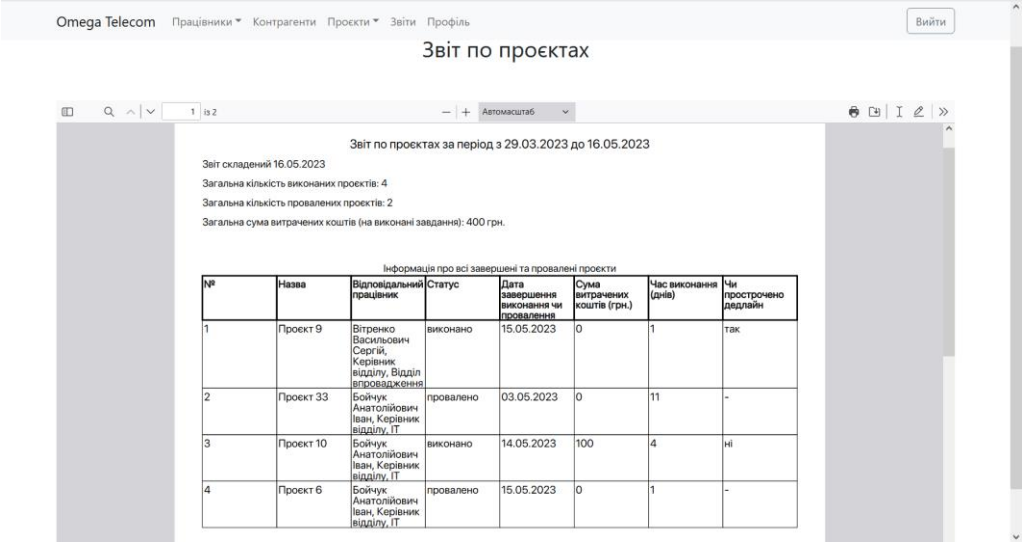

Рис. 4.33 Сторінка зі звітом по проєктах

ВИСНОВКИ

Отже, було проведено аналіз існуючих рішень, визначено групи користувачів та їх вимоги до функціональності системи, спроєктовано архітектуру застосунку, створено ER-модель та дві діаграми станів, визначено інструменти для розробки застосунку. З’ясовано, що бібліотека GoJS має детальну документацію, достатню кількість готових діаграм-прикладів із коментарями до коду, зручна у використанні та надає величезний набір параметрів для кастомізації діаграм.

Розроблено ядро CRM-системи у вигляді веб-застосунку із повним запланованим функціоналом, що покриває всі визначені вимоги користувачів для трьох ролей із чітким розподіленням прав доступу. Застосунок візуалізує послідовність завдань та поточну стадію виконання проєктів за допомогою діаграм GoJS, що спрощує контроль над прогресом проєкту та покращує розуміння для працівників, коли потрібно виконувати свої задачі. Реалізоване автоматичне формування звітів полегшує оцінку продуктивності роботи окремих працівників та компанії в цілому.

Розроблений веб-застосунок готовий до впровадження у роботу компанії Omega Telecom. За потреби, система може бути легко розширена додатковою функціональністю (наприклад, шаблонами послідовності завдань для проєктів) завдяки чіткому розподілу відповідальності між її рівнями та класами. Система також може використовуватися (можливо, з певними модифікаціями) в інших компаніях, робота яких побудована у форматі виконання проєктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ЩО таке CRM: визначення | SendPulse Україна. *SendPulse*. URL: https://sendpulse.ua/support/glossary/crm (дата звернення: 18.05.2023).
2. Monday work management – the work management software to maximize business efficiency. *monday.com*. URL: https://monday.com/work-management (дата звернення: 18.05.2023).
3. KeepinCRM. *KeepinCRM*. URL: https://keepincrm.com/ (дата звернення: 18.05.2023).
4. What is client server architecture? - differences, types, example. *Intellipaat Blog*. URL: https://intellipaat.com/blog/what-is-client-server-architecture/?US (дата звернення: 18.05.2023).
5. SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення: 18.05.2023).
6. Collings T. Controller-Service-Repository. *Medium*. URL: https://tom-collings.medium.com/controller-service-repository-16e29a4684e5 (дата звернення: 18.05.2023).
7. Inversion of Control and Dependency Injection with Spring | Baeldung. *Baeldung*. URL: https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring (дата звернення: 18.05.2023).
8. UML - statechart diagrams. *Online Courses and eBooks Library*. URL: https://www.tutorialspoint.com/uml/uml_statechart_diagram.htm (дата звернення: 18.05.2023).
9. A Comparison Between Spring and Spring Boot | Baeldung. *Baeldung*. URL: https://www.baeldung.com/spring-vs-spring-boot (дата звернення: 18.05.2023).

10. Офіційний сайт React. *React*. URL: https://react.dev/ (дата звернення: 18.05.2023).

11. What is Bootstrap?. *Hostinger Tutorials*. URL: https://www.hostinger.com/tutorials/what-is-bootstrap/ (дата звернення: 18.05.2023).

12. Офіційний сайт React-Bootstrap. *React-Bootstrap · React-Bootstrap Documentation*. URL: https://react-bootstrap.github.io/ (дата звернення: 18.05.2023).

13. Офіційний сайт Redux. | Redux. *Redux*. URL: https://redux.js.org/ (дата звернення: 18.05.2023).

14. Офіційний сайт React Redux. *React Redux*. URL: https://react-redux.js.org/ (дата звернення: 18.05.2023).

15. Офіційний сайт GoJS. *GoJS - Build Interactive Diagrams for the Web*. URL: https://gojs.net/latest/index.html (дата звернення: 18.05.2023).

16. GitHub - NorthwoodsSoftware/gojs-react: A set of React components to manage GoJS Diagrams, Palettes, and Overviews. *GitHub*. URL: https://github.com/NorthwoodsSoftware/gojs-react (дата звернення: 18.05.2023).

17. What Is OncePerRequestFilter? | Baeldung. *Baeldung*. URL: https://www.baeldung.com/spring-onceperrequestfilter (дата звернення: 18.05.2023).

18. Managing JWT With Auth0 java-jwt | Baeldung. *Baeldung*. URL: https://www.baeldung.com/java-auth0-jwt (дата звернення: 18.05.2023).

19. Manually authenticate user with Spring Security | Baeldung. *Baeldung*. URL: https://www.baeldung.com/manually-set-user-authentication-spring-security (дата звернення: 18.05.2023).

20. Jeong S. Send Email with Spring Boot And Gmail. *Medium*. URL: https://medium.com/@Seonggil/send-email-with-spring-boot-and-gmail-27c14fc3d859 (дата звернення: 18.05.2023).

21. GoJS JavaScript Diagramming Library. What's in a GoJS JavaScript Application? | GoJS Beginner Tutorial #1, 2018. *YouTube*. URL: https://www.youtube.com/watch?v=dsC7Tf5mDO8 (дата звернення: 18.05.2023).

22. GoJS and React -- Northwoods Software. *GoJS - Build Interactive Diagrams for the Web*. URL: https://gojs.net/latest/intro/react.html (дата звернення: 18.05.2023).

23. Redux Fundamentals, Part 1: Redux Overview | Redux. *Redux*. URL: https://redux.js.org/tutorials/fundamentals/part-1-overview (дата звернення: 18.05.2023).