НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри
_____ Сергій Стіренко
(підпис)

“__” _____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Сервіс для навчання нових співробітників на підприємстві

Виконав (-ла): студент (-ка) 4 курсу, групи ІВ-71
(шифр групи)

Мазан Ян Владиславович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент Сергієнко А. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) проф. д.т.н. Сімоненко В.П. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Мазана Яна Владиславовича

1. Тема проєкту *Сервіс для навчання нових співробітників на підприємстві*
керівник проєкту Сергієнко Анастасія Анатоліївна, асистент ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 11.05 2021 року № 1139-с
2. Термін здачі студентом закінченого проєкту *27 травня 2021 р.*
3. Вихідні дані до проєкту *технічна документація, прикладний програмний інтерфейс.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Опис предметної області, аналіз наявних програмних рішень, розробка

архітектури сервісу для навчання нових співробітників на підприємстві, тестування та впровадження сервісу

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) діаграма класів, діаграма розгортання застосунку, блок-схеми алгоритмів роботи програми

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормконтроль	Сімоненко В. П.		

7. Дата видачі завдання 15.12.2020

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2020-15.12.2020*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.03.2021*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2021-25.03.2021*	
4.	*Розробка структур окремих підсистем*	*25.03.2021-5.04.2021*	
5.	*Програмна реалізація системи*	*5.04.2021-15.04.2021*	
6.	*Оформлення пояснювальної записки*	*15.04.2021-20.05.2021*	
7.	*Захист програмного продукту*	*25.04.2021*	
8.	*Передзахист*	*23.05.2021*	
9.	*Захист*	*15.06.2021*	

Студент-дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

Анотація

В бакалаврському дипломному проєкті спроєктовано і реалізовано вебсервіс для навчання нових співробітників на підприємстві.

Застосунок надає змогу користувачу реєструвати підприємство, додавати до нього нових співробітників, управляючи їх ролями. Менеджер на підприємстві може створювати шаблони для початкових робочих задач, назначати їх новому співробітнику та слідкувати за його прогресом. Співробітник, у свою чергу, отримує можливість дізнатися деталі робочих завдань та отримати відповіді на питання.

Клієнтська та серверна частини сервісу реалізовані мовою TypeScript у середовищі NodeJS з використанням технологій React та Nest.js та розгорнуто у хмарному сервісі AWS для забезпечення подальшої масштабованості застосунку.

Annotation

In the given bachelor's thesis, the web service for training new employees at the enterprise is designed and implemented.

The application allows a user to register an enterprise, add new employees and manage their roles. Manager at the enterprise can create templates for initial work tasks, assign them to a new employee and monitor his progress. The employee, in turn, gets the possibility to learn the details of work tasks and get answers to questions.

The client and server components of the service are implemented in the TypeScript language in the NodeJS environment using React and Nest.js technologies, and deployed to the AWS cloud service to provide future scalability of the application.

ОПИС АЛЬБОМУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Додаток
	А4		Завдання на дипломний проєкт	2	
	А4	ІАЛЦ.467200.001 ВП	Відомість проєкту	2	
	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	4	
	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	75	
	А4	ІАЛЦ.467200.006 Д3	Функціональна схема	1	
	А4	ІАЛЦ.467200.005 Д2	Структурна схема	1	
	А4	ІАЛЦ.467200.004 Д1	Принципова схема	1	
	А4	ІАЛЦ.467200.007 Д4	Лістинг програми	10	

Зм.	Арк.	№ докум.	Підпис	Дата	*ІАЛЦ.467200.001 ОА*		
Розроб.		Мазан Я. В.			*Сервіс для навчання нових співробітників на підприємстві* Опис альбому	Лит.	Арк. 1 / Аркушів 1
Перев.		Сергієнко А. А.					
Реценз.						НТУУ «КПІ», ФІОТ, ІВ-71	
Н. Контр.		Сімоненко В. П.					
Затверд.		Стіренко С. Г.					

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня бакалавр

на тему: «Сервіс для навчання нових співробітників на підприємстві»

Київ – 2021

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку програмного забезпечення для потреб бізнесу. Область застосування: використання для управління бізнес-процесами на підприємстві.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка сервісу для навчання нових співробітників на підприємстві, його розгортання у хмарному середовищі та тестування роботи сервісу.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики розробки вебсервісів, проєктування програмного забезпечення, наукові статті, публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до продукту, що розробляється

1. Можливість доступу через будь-який сучасний веббраузер
2. Зручний користувацький інтерфейс
3. Надійність зберігання даних

4. Можливість подальшого розширення та доповнення розроблюваного продукту

5.2. Вимоги до програмного забезпечення

1. Установлене середовище Node.js 8-ї версії та вище
2. Операційна Unix-подібна система або Windows вище 7-ї версії

5.3. Вимоги до апаратної частини

1. Комп’ютер на базі процесору Intel Core i3 і вище
2. Оперативної пам’яті не менше 2 Гбайт
3. Вільного місця на жорсткому диску не менше 1 Гбайт
4. Стабільне підключення до Інтернету

6. ЕТАПИ РОЗРОБКИ

	Дата
Затвердження теми роботи	15.12.2020
Аналіз завдання та огляд існуючих рішень	15.03.2021-5.04.2021
Розробка архітектури та загальної структури системи	5.04.2021-10.04.2021
Програмна реалізація системи	10.04.2021-1.05.2021
Доопрацювання і налагодження системи	1.05.2021-6.05.2021
Оформлення пояснювальної записки	2.05.2021-20.05.2021
Передзахист та проходження нормативного контролю	25.05.2021
Захист дипломного проєкту	15.06.2021

Пояснювальна записка

до дипломного проєкту
на тему: «Сервіс для навчання нових співробітників на підприємстві»

Київ – 2021

ЗМІСТ

Перелік скорочень

SaaS	програмне забезпечення як послуга (англ. Software as a service)
HR	менеджер управління персоналом (англ. Human resources)
MVC	модель-вигляд-контролер (англ. Model-View-Controller)
API	прикладний програмний інтерфейс (англ. Application Programming Interface)
IoC	інверсія управління (англ. Inversion of Control)
DI	впровадження залежностей (англ. Dependency Injection)
SPA	односторінковий застосунок (англ. Single Page Application)
ORM	об’єктно-реляційне відображення (англ. Object-relational mapping)
JSON	запис об'єктів JavaScript (англ. JavaScript Object Notation)
JWT	JSON вебтокен (англ. JSON Web Token)

ВСТУП

У зв’язку зі збільшенням складності ринку праці, вимог до кваліфікації співробітників та їх інтеграції в команду, найм та адаптація нових співробітників на підприємстві є актуальною проблемою в сучасному світі. У перші дні на робочому місці новий співробітник стикається зі специфічними деталями роботи, які можуть значно відрізнятися в залежності від підприємства та які не вивчаються попередньо у закладі здобуття освіти, а також з колегами, з якими потрібно налагодити робочі стосунки. Підприємству вигідно оптимізувати процес звикання нового співробітника до робочих процесів, щоби мінімізувати можливість помилок з його сторони, для підтримки корпоративної культури компанії та щоб збільшити загальну продуктивність.

Мета дипломного проєкту — розробити програмний продукт для полегшення процесу адаптації нових співробітників. Новий співробітник повинен розуміти специфіку поставлених перед ним робочих задач, мати можливість познайомитись із колегами та отримати відповіді на робочі запитання. Менеджер, водночас, повинен мати можливість оцінити нового співробітника, як він справляється з робочими задачами та комунікує з колегами, надати йому робочі завдання та отримати відгук щодо якості процесу адаптації на робочому місці.

Для досягнення мети пропонується розробити SaaS-застосунок — сервіс, розміщений у хмарному сховищі, до якого підприємство та співробітники можуть отримати захищений доступ та використовувати в процесі навчання.

В дипломній роботі оглянуто існуючі програмні рішення та визначено функціонал, який повинен містити розроблюваний сервіс. Також оглянуто програмні можливості реалізації сервісу і визначені найбільш вдалі технології.

РОЗДІЛ 1
ОГЛЯД НАЯВНИХ СЕРВІСІВ НАВЧАННЯ НОВИХ СПІВРОБІТНИКІВ НА ПІДПРИЄМСТВІ

1.1. Опис предметної області

Організаційна соціалізація (або онбоардинг, від англ “onboarding”) — процес, протягом якого нові співробітники залучаються до процесів роботи підприємства, отримують знання, здобувають навички та вивчають поведінкові принципи, які потрібні для досягнення успіху на робочому місці. Специфіка необхідних технічних та соціальних навичок відрізняється на кожному підприємстві, у чому і полягає різниця від професійної соціалізації (від англ. “occupational socialization”), під якою мається на увазі вивчення навичок, необхідних для конкретної окремої професії (лікар, співробітник ресторанного бізнесу, програміст тощо) [1].

Після прийняття співробітника на роботу йому необхідно вивчити всі деталі робочого процесу, які переважно значно відрізняються залежно від підприємства. Під час перших робочих днів співробітник повинен як виконати певний набір базових робочих задач: налаштувати робоче місце, вивчити специфічні правила поведінки під час взаємодії з клієнтом тощо, так і мати можливість отримати відповіді на питання, отримавши допомогу, та познайомитись із іншими співробітниками [2]. Водночас, менеджер повинен мати можливість відслідковувати прогрес нового співробітника та контролювати виконання ним базових робочих задач, щоби оцінити його якість роботи під час випробувального терміну (рис. 1.1). Тому виокремлення періоду адаптації співробітника та правильна його організація дає змогу підприємству збільшити власну продуктивність, покращити взаємодію нового співробітника з колегами, допомогти йому краще запам’ятати специфіку роботи на конкретному підприємстві та збільшити ймовірність того, що співробітник працюватиме на підприємстві довший час [3].

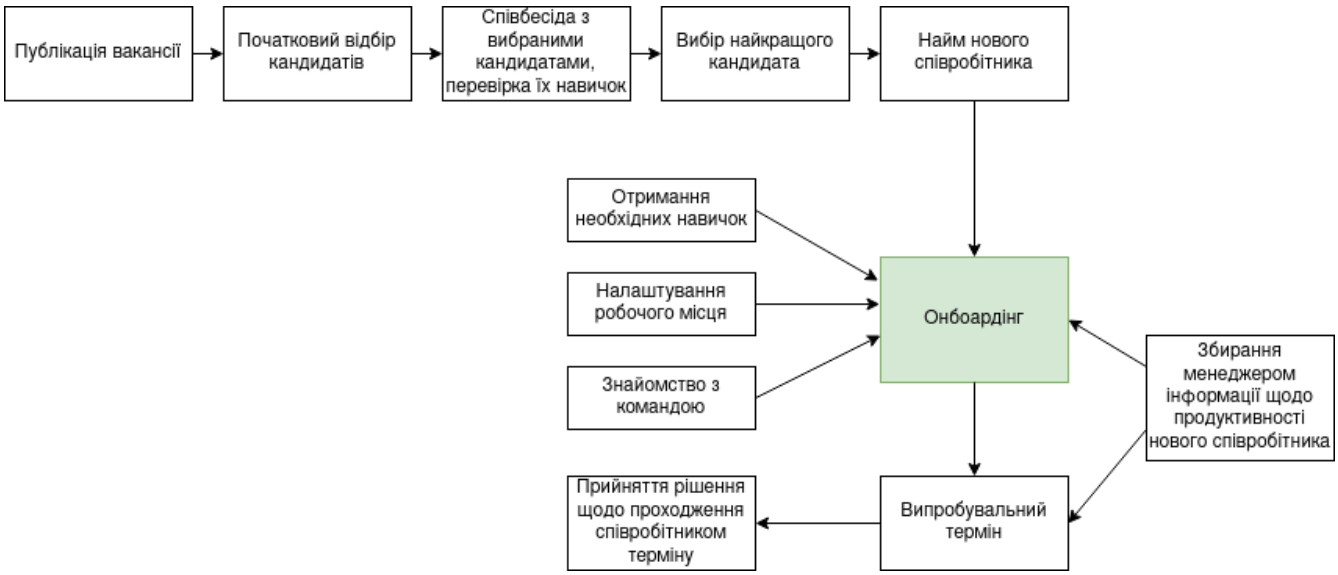


Рис. 1.1. Діаграма загальної процедури прийняття на роботу нового співробітника з виділеним етапом, що розглядається.

Програмне забезпечення для онбоардингу використовується для полегшення описаного вище процесу. Під час користування додатком новому співробітнику надаються навчальні матеріали та початкові робочі завдання. Він також має можливість ставити запитання ментору та комунікувати з іншими співробітниками. Менеджер чи HR може слідкувати за прогресом нового співробітника, збирати необхідну для налагодження робочих процесів додаткову інформацію про нього, призначати ментора, відслідковуючи соціальну інтеграцію нового співробітника в команду. В менеджера повинна бути можливість формувати звітність та інформувати співробітника про важливі оновлення через засоби електронної комунікації: електронною поштою, пуш-сповіщеннями чи іншими засобами для робочої комунікації між співробітниками (Slack тощо), інтегрованими з додатком.

1.2. Огляд наявних програмних рішень

Нижче наведено розгорнутий огляд деяких сервісів з відповідним функціоналом: навчанням нових співробітників на підприємстві. Варто зауважити, що для всіх оглянутих додатків дана функція не є основною —

переважно вони зосереджені на управлінні робочим персоналом загалом: тобто розподілом робочих задач між співробітниками, вимірюванням їх ефективності чи управління наймом. Всі сервіси є комерційними, доступ до яких можна отримати через платну підписку на певний термін.

Згадані нижче сервіси для навчання нових співробітників на підприємстві є SaaS-застосунками (“програмне забезпечення як послуга” з англ. “Software as a Service”) — тобто комерційне програмне забезпечення, яке розробник постачає через мережу Інтернет (зазвичай через хмарний сервіс), а користувачі отримують доступ до нього через інтерфейс вебзастосунку [4]. Також усі з них, окрім “хмарної” версії підтримують версії для встановлення на власний сервер з можливістю налаштування.

1.2.1. Process Street

Process Street — це комерційний американський SaaS-застосунок, що розробляється та підтримується з 2013 року, для управління бізнес-процесами підприємства з використанням контрольних списків — шаблонів, в яких знаходиться необхідна інформація для виконання задачі. До основного функціоналу належать реєстрація підприємства через окрему поштову скриньку користувача, призначення адміністраторів, додавання нових співробітників, створення контрольних списків та їх наборів — сценаріїв для певних робочих задач (наприклад, налаштування програмного забезпечення для роботи, процесу стандартного обслуговування клієнта тощо). Контрольні списки з задачами (рис. 1.2) можна призначати окремим співробітникам та контролювати їх виконання. Сервісу притаманна простота та інтуїтивність у використанні, є вбудована документація, автоматичне формування звітності та інтеграція зі сторонніми застосунками для управління робочими процесами (Slack, Trello, Microsoft Outlook та інші).

Організація роботи за допомогою контрольних списків дозволяє ефективно

розподілити шаблонну задачу багатьом співробітникам, відслідкувати їх прогрес. Використання контрольних списків допомагає отримати співробітнику всю необхідну інформацію для виконання задачі, мінімізувати власні помилки, повідомити менеджеру деталі її виконання та ефективніше управляти своїм часом.

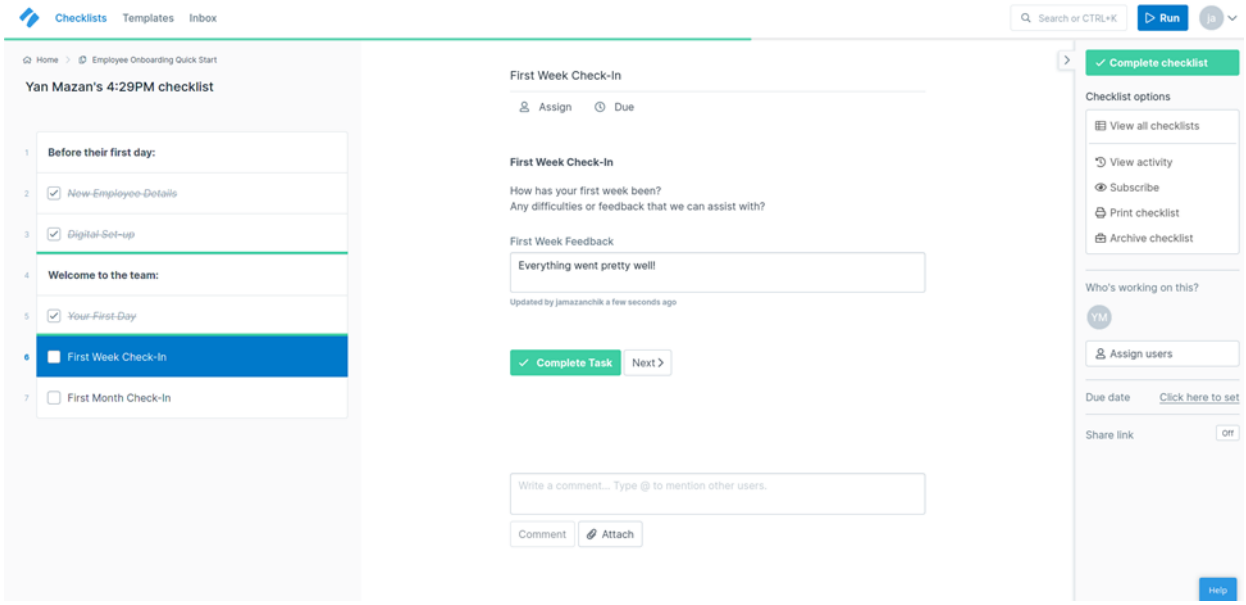


Рис. 1.2. Заповнення контрольного списку в Process Street [5].

Переваги:

1. Сучасний мінімалістичний дизайн;
2. Зручне створення шаблонів для задач, можливість їх перевикористання для навчання великої кількості співробітників;
3. Загальна мінімалістичність застосунку, його орієнтованість на саме навчання співробітників;
4. Можливість назначити ментора новому співробітнику.

Недоліки:

1. Незручність функції перегляду прогресу усіх співробітників адміністратором (HR чи менеджером), необхідність перевірки деталей за кожним користувачем особисто;
2. Відсутність локалізації мовами відмінними від англійської.

1.2.2. ApplicantStack

ApplicantStack — комерційний SaaS-застосунок, орієнтований на споживачів з Канади та США для управління процесом найму та навчання співробітників. Є однією з найпопулярніших платформ для найму співробітників у цих країнах у сферах охорони здоров'я, наукових досліджень, інженерного проєктування та промислового виробництва [6]. Розроблений американською компанією SwipeClock і підтримується з 1999 року. Продукт розділений на дві частини: ApplicantStack Recruit (найм нових співробітників) та ApplicantStack Onboard (їх навчання) — надалі в розділі розглядатиметься лише друга.

Процес навчання нового співробітника, як і в додатку Process Street, полягає в призначенні співробітнику менеджера та набору базових необхідних для навчання задач, які можна шаблонізувати для перевикористання. Процес навчання співробітника можна розділити на декілька етапів, назначивши для кожного відповідний контрольний список.

У порівнянні з застосунком Process Street, ApplicantStack пропонує можливість кращої деталізації інформації щодо нового співробітника (рис. 1.3), специфічної для території США та Канади: електронну верифікацію, зберігання даних про найнятого співробітника у форматі форми I-9 ("Employment Eligibility Verification", яка є обов'язковою для кожного співробітника підприємства на території США), реалізована можливість при наймі надсилати співробітнику користувацьке e-mail-запрошення. Доступ до продукту можна отримати в режимі платної підписки на певний термін. Йому притаманний дещо більш складний та деталізований користувацький інтерфейс, але інструменту опису нових задач не вистачає інтерактивності та повноти, необхідної для нового співробітника — вони описуються шляхом заповнення форми з необов'язковим текстовим полем опису задачі. У ApplicantStack є API, яке можна використовувати для інтеграції сервісу з іншими системами всередині

підприємства [7].

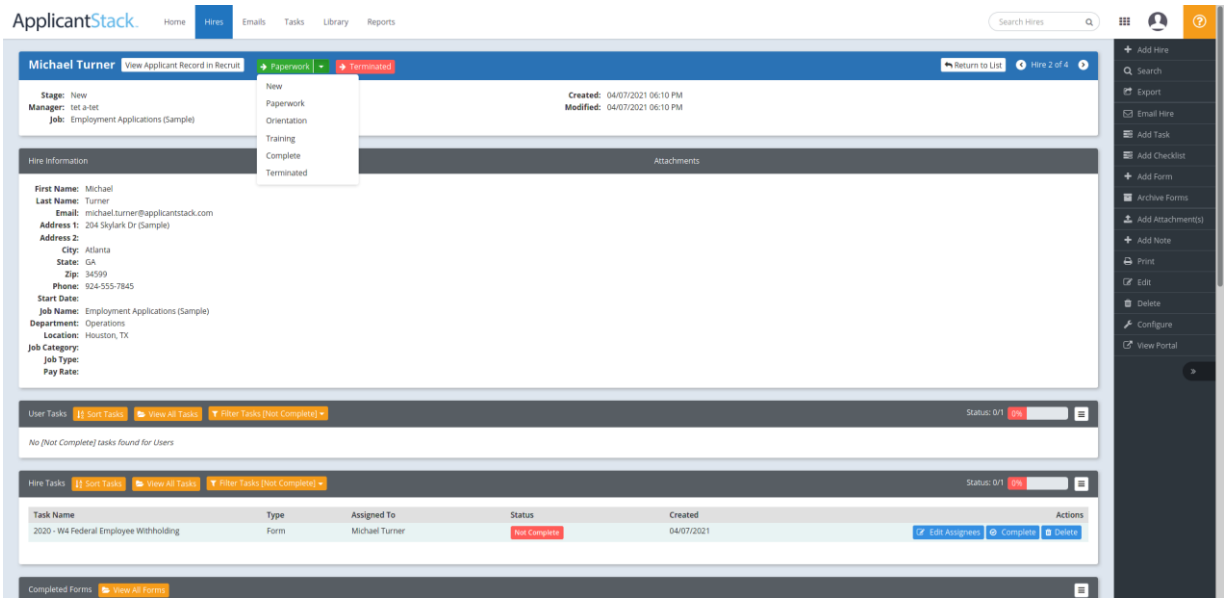


Рис. 1.3. Перегляд інформації про нового співробітника в ApplicantStack [8].

Переваги:

1. Окреме доменне ім’я для підприємства на ресурсі;
2. Можливість управління процесом найму нових співробітників з додатка.

Недоліки:

1. Орієнтованість на виключно американський ринок;
2. Несучасний заплутаний інтерфейс користувача, неоптимальне використання ширини екрана;
3. Гірша у порівнянні з конкурентними продуктами можливість якісного опису для працівника деталей задачі;
4. Відсутність мобільної версії застосунку;
5. Висока вартість даного технологічного рішення;
6. Відсутність локалізації мовами, відмінними від англійської.

1.2.3. Rippling

Це комерційний SaaS-застосунок, що розробляється та підтримується однойменною компанією з 2012 року для управління персоналом на

підприємстві (рис. 1.4). Це означає, що управління навчанням нових співробітників є лише однією незначною функцією сервісу. Загалом, до його функціоналу належать також функції розподілу задач між робітниками, вимірювання їх ефективності, обрахунок рівня заробітної плати з урахуванням роботи у святкові/неробочі дні, хвороби чи відпусток тощо, управління документообігом, управління наймом нових співробітників (розміщення вакансії, відбір кандидатів серед надісланих резюме, організація проведення співбесід з остаточним оцінюванням кандидатів та вибору найкращого), формуванням звітності тощо. Підтримує інтеграцію з іншими застосунками для управління робочими процесами (Slack, Jira, Google Workspace тощо).

Порівнюючи з попередніми оглянутими сервісами, функціонал навчання нових співробітників у Rippling не є одним з основних. Він реалізується шляхом запрошення співробітника на роботу (присутня можливість надіслати йому користувацьке e-mail-запрошення) та наданням йому базових початкових задач. Можливість об’єднати задачі у контрольний список відсутня, а збір інформації, специфічної для навчання нового співробітника, не є для менеджера окремою функцією.

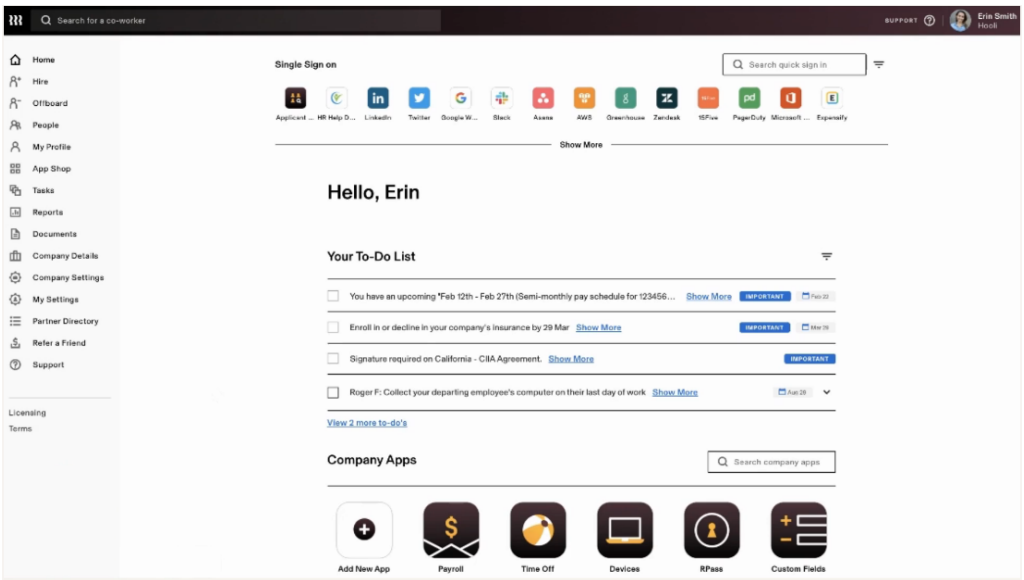


Рис. 1.4. Вигляд головного екрана застосунку Rippling [9].

Переваги:

1. Можливість налаштування SSO (налаштувавши застосунок у локальній мережі, можна використовувати єдиний логін та пароль для використання з усіма іншими установленими програмами);
2. Можливість інтеграції зі сторонніми застосунками для організації робочого процесу на підприємстві (Slack, Jira, Google Workspace, Microsoft Outlook, LinkedIn, застосунки для медичного трекінгу працівників тощо);
3. Зручний інтерфейс користувача;
4. Можливість трекінгу часу, витраченого на задачі, управління наймом співробітників та формування звітності.

Недоліки:

1. Неможливість назначити ментора для нового співробітника, нерозрізнення його як окремої сутності від “досвідчених” робітників; відсутність можливості співробітника ставити запитання старшому та отримувати відповідь;
2. Загальна орієнтованість на управління персоналом, а не на онбоардинг нових співробітників — означає деяку заплутаність застосунку та його можливо надто великий об’єм для невеликого підприємства;
3. Відсутність локалізації мовами, відмінними від англійської.

1.2.4. Greenhouse

Greenhouse — комерційний SaaS-застосунок для управління процесом найму та навчання нових співробітників. Розробляється американською компанією Greenhouse Software з 2012 року. Як і з попередніми оглянутими додатками, робота з ним починається з реєстрації підприємства користувачем, який стає в ньому адміністратором. Клієнтська частина сервісу розділена на робочі панелі (рис. 1.5), в кожній з яких є відповідні функціональні елементи: вакансії, розміщені компанією, список кандидатів, взаємодія з клієнтами, генерування

звітності, інтеграція з іншими застосунками тощо. Також присутня бічна панель з робочими завданнями для кожного користувача. При наймі нового співробітника менеджер має можливість призначити йому шаблон з набором стандартних початкових завдань та список документів, які той повинен підписати. Як і в Rippling та ApplicantStack, у застосунку реалізована можливість надсилати новому співробітнику користувацьке e-mail-запрошення.

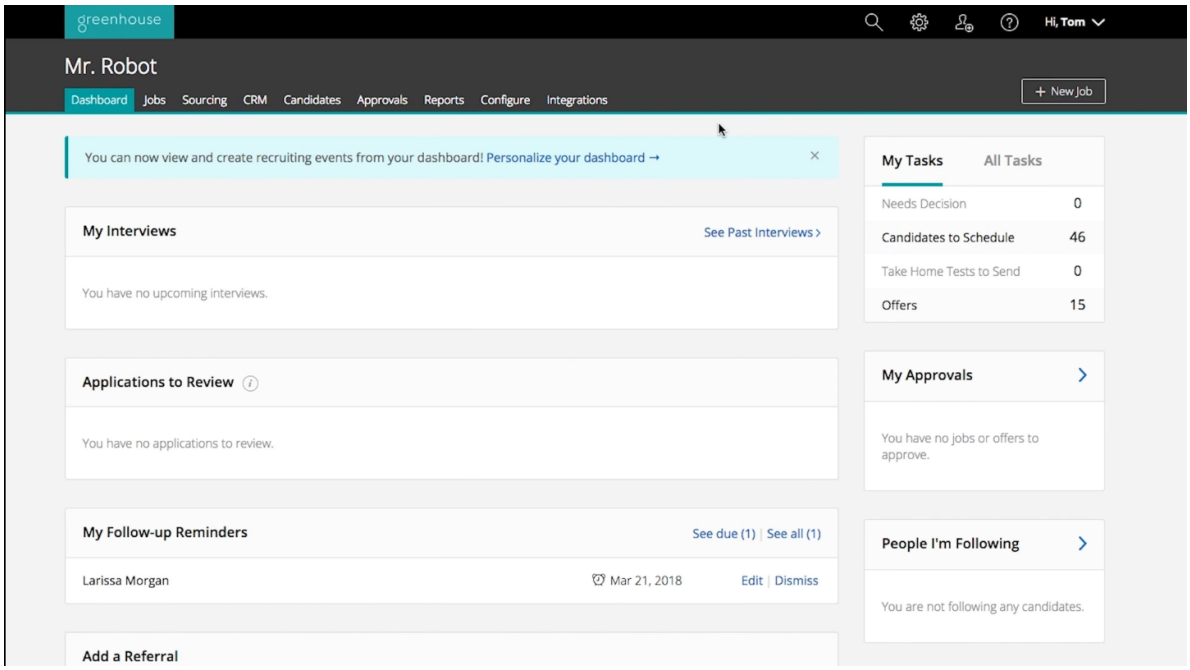


Рис. 1.5. Вигляд головного екрана додатку Greenhouse [10].

Переваги:

1. Можливість налаштування SSO;
2. Можливість управління процесом найму співробітників;
3. Зручність зберігання специфічної інформації (на кшталт особистих деталей) про кожного співробітника;
4. Зручне налаштування плану онбоардингу, трекінг часу, витраченого на задачі;
5. Опція налаштування сповіщень, інтегровність зі Slack.

Недоліки:

1. Загальна орієнтованість на управління персоналом, а не на онбоардинг нових співробітників — означає деяку заплутаність застосунку та його можливо надто масштабний функціонал для невеликого підприємства;
2. Відсутність локалізації мовами, відмінними від англійської.

1.2.5. Бітрікс24

Бітрікс24 (рос. “Битрикс24”) — російський SaaS-застосунок, що використовується для організації робочих процесів в компанії та взамодії з клієнтами (англ. Customer Relationship Management або CRM). Розроблений на мові PHP, може бути встановлений як на окремих серверах з можливістю користувацького налаштування, так і використовуватися через вебверсію. Проєкт було запущено у 2012 році.

Як і в попередніх оглянутих сервісах, користувач має можливість зареєструвати підприємство, запросити до нього співробітників (рис. 1.6). Функція управління новими співробітниками та їх навчання не є основною, натомість робота сервісу зосереджена на налаштуванні процесів взаємодії з клієнтами. Також сервіс критикується за регулярні збої в роботі, застарілий графічний інтерфейс користувача та низьку якість роботи служби технічної підтримки.

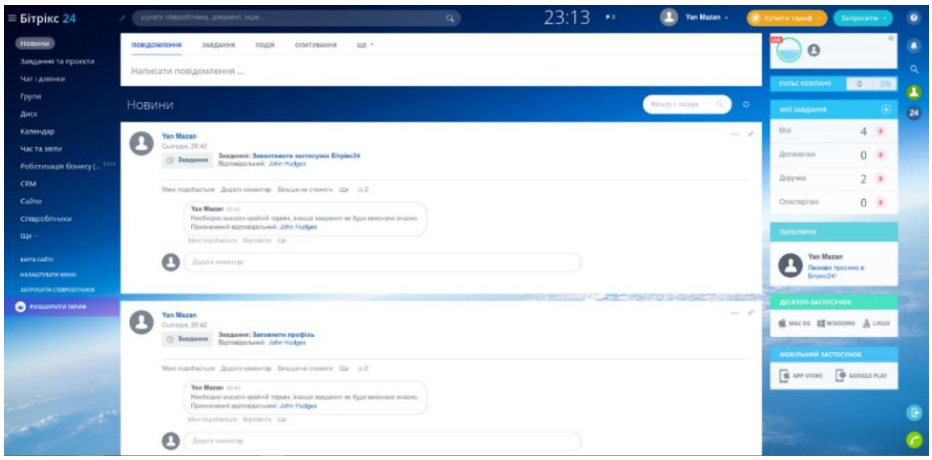


Рис. 1.6. Вигляд головної сторінки підприємства у Бітрікс24.

Переваги:

1. Локалізація іншими мовами, окрім англійської (зокрема, українською).

Недоліки:

1. Несучасний користувацький дизайн;
2. Неякісна робота служби підтримки.

Таблиця 1.1. Порівняння оглянутих сервісів

	Process Street	ApplicantStack	Rippling	Greenhouse	Бітрікс24
Наявність версії для інсталяції на власному сервері, окрім SaaS	Відсутня	Наявна	Наявна	Наявна	Наявна
Якість користувацького інтерфейсу	Сучасний, є адаптивний дизайн	Без адаптивного дизайну, неоптимальне використання ширини екрану	Сучасний, без адаптивного дизайну	Сучасний, без адаптивного дизайну	Несучасний надто кольоровий дизайн, який відволікає від роботи
Організація задач для співробітника у контрольні списки	Присутня	Присутня	Відсутня	Присутня	Відсутня
Наявність можливості назначити спеціальний список задач для нового співробітника	Присутня	Присутня	Присутня	Присутня	Відсутня
Налаштування SSO	Відсутнє	Через розширення	Присутнє	Присутнє	Відсутнє

Продовження таблиці 1.1

	Process Street	ApplicantStack	Rippling	Greenhouse	Бітрікс24
Локалізація іншими мовами, окрім англійської	Відсутня	Відсутня	Відсутня	Відсутня	Присутня
Комунікація зі співробітниками	Тільки сповіщення про оновлення	Через електронну пошту	Через електронну пошту	Через електрон-ну пошту	Можли-вість перепис-ки, відео-дзвінки

1.3. Вимоги до проєктованого продукту

Враховуючи предметну область та аналіз схожих програмних рішень, можна визначити необхідні мінімальні вимоги, яким має відповідати проєктований сервіс. По-перше, в ньому повинна бути присутня сутність підприємства, яку його власник має мати можливість зареєструвати та до якої надавати доступ іншим користувачам (менеджеру та новим співробітникам). Має бути забезпечена ізольованість сутностей підприємства одна від одної та відповідний рівень захисту як особистої інформації співробітників, так і деталей роботи підприємства.

У нового співробітника у сервісі повинен бути доступ до списку базових початкових задач з необхідною інформацією для її виконання. Ефективність показала організація опису задач з використанням контрольних списків. У співробітника повинна бути можливість комунікувати з ментором та отримувати відповіді на запитання, а в менеджера — можливість відслідковувати прогрес співробітника, формувати звітність, створювати та редагувати наявні контрольні списки задач, організовуючи їх у різні шаблони для сценаріїв навчання співробітників на різних посадах. Обов’язковою є підтримка інформування

інших користувачів через електронну пошту, внутрішній інтерфейс для надсилання та отримання повідомлень, бажаною є можливість інтеграції сервісу з іншими застосунками для робочої комунікації (такими як Slack тощо).

Сервіс повинен бути доступним для нових співробітників, зокрема, на мобільних пристроях, наприклад, у випадку працівників ресторанного бізнесу. Виникають вимоги до зручного адаптивного дизайну, доступності з різних пристроїв та налаштування сервісу. Підхід SaaS, який використовується у всіх оглянутих технологічних рішеннях, виявляється доцільним, так як потреба у “розгортанні” та налаштуванні сервера застосунку переноситься на розробника, а доступ відбувається через інтерфейс веб-, десктопного чи мобільного додатка — це вирішує проблему з налаштуванням та доступністю сервісу. Доступ через вебзастосунок полегшує розробку та підтримку програмного продукту, усуває проблеми з переносимістю схожого інтерфейсу.

ВИСНОВКИ ДО РОЗДІЛУ 1

Існує велика кількість рішень цієї задачі (сервісів для навчання нових співробітників на підприємстві) — в першому розділі було проаналізовано деякі найпопулярніші з них і визначено мінімальні необхідні складові, які повинен включати в себе відповідний сервіс.

При огляді рішень було виявлено, що всі з них є складними та дороговартісними для користувача системами, велика частина яких орієнтована на специфічний американський ринок та є доволі комплексною, вирішуючи не тільки проблему навчання нових співробітників, а й загалом управління персоналом. Останній функціонал для підприємства зазвичай не є надлишковим, але може значно ускладнити роботу з сервісом та використовуватись не повною мірою — виникає ситуація, в якій невелике підприємство деколи переплачує за непотрібний йому функціонал. Також застосунок може не вирішувати всіх проблем зі зручним наданням необхідної інформації для нового співробітника, а відсутність локалізації іншими мовами, окрім англійської, у більшості згаданих вище застосунків створює незручності для використання.

Визначені мінімальні вимоги до сервісу дозволяють краще поставити технічне завдання для його проєктування. Запропоновано дотримуватись підходу SaaS, розробляючи єдиний вебзастосунок зі сучасним адаптивним дизайном.

РОЗДІЛ 2
АНАЛІЗ ЗАСОБІВ РОЗРОБКИ

2.1. Архітектура сервісу

2.1.1. Монолітна архітектура

При розробці програмного застосунку можна використати монолітну архітектуру, коли програмні компоненти розгортаються разом, як окремий сервіс. У даному розділі розглянуто монолітну архітектуру з організацією на прикладі багатошарової, а також шаблон проєктування MVC, який часто застосовується при розробці.

Розроблюваний сервіс, як згадувалось у першому розділі, повинен містити клієнтську частину (браузерний додаток), з якою взаємодіє користувач (Presentation Layer), рівень бізнес-логіки (Business Logic Layer), рівень доступу до даних (Data Access Layer) та саме джерело даних (база даних) і, можливо, зовнішні сервіси — такі як сервіс сповіщень по e-mail. Традиційно така структура називається багатошаровою (рис. 2.1) [11].

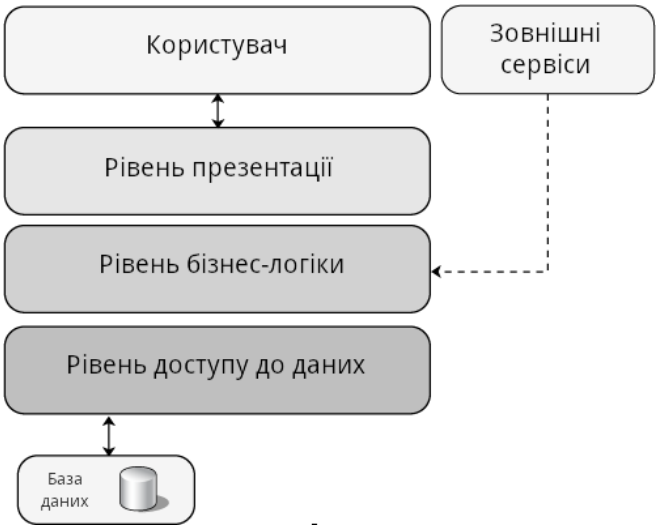


Рис. 2.1. Багатошарова архітектура.

Для реалізації багатошарової архітектури використовується архітектурний шаблон MVC (Model-View-Controller). Це фундаментальний шаблон

проєктування програмного забезпечення, який складається з трьох видів об’єктів: моделі (Model), представлення (View) та контролера (Controller). Уперше був описаний для мови програмування Smalltask-80 у 1988 році, є популярним у веброзробці.

Модель — частина програми, що містить в собі її функціональну бізнес-логіку. Є незалежною він інших елементів застосунку. Може мати вигляд як просто шару доступу до даних, так і мати додатково набір сервісів, які проводять над ними необхідні операції.

Представлення — частина програми, яка відображає дані, отримані від моделі і не може зворотньо змінювати її стан. Зазвичай, це інтерфейс користувача. В програмній частині цього компонента реалізоване тільки відображення даних та, в деяких випадках, мінімальна бізнес-логіка: наприклад, це може бути валідація вхідних даних та надсилання на сервер — передача вводу користувача контролеру. Оновлення моделі стандартно реалізоване шаблоном публікації-підписки: представлення підписується на зміни стану моделі та при зміні отримує новий стан.

Контролер — частина програми, яка приймає на вхід ввід користувача та змінює стан моделі, викликаючи її потрібні методи. Водночас, це не означає, що модель залежить від контролера — використовується тільки її публічний інтерфейс, а модель змінює свій стан самостійно (рис 2.2) [12].

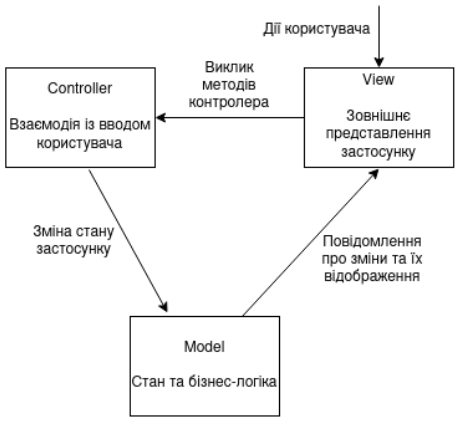


Рис. 2.2. Структура шаблону MVC.

У монолітної архітектури є значна кількість переваг, зокрема більша швидкість розробки (так як програмні компоненти розробляються разом), полегшується тестування (наприклад, можна легко запустити end-to-end — тестування з точки зору користувача — чи інтеграційні тести — перевірка того, як окремі модулі системи взаємодіють між собою), так як додаток є цілісним. Важливою перевагою є легкість розгортання з тієї ж причини, а також легкість масштабування.

Недоліки полягають у складності внесення змін чи заміни компонентів, так як вони тісно пов’язані, великий розмір застосунку значно збільшує час запуску, а помилка в окремій підсистемі одразу негативно впливає на весь застосунок, що загрожує припиненням роботи всієї системі в разі її виникнення.

2.1.2. Мікросервісна архітектура

Ідея мікросервісної архітектури полягає у тому, що застосунок розділений на деяку кількість малих зв’язаних сервісів — мікросервісів. Зазвичай окремий мікросервіс має гексагональну архітектуру, коли програма розбивається на декілька слабозв’язаних компонентів, що поєднуються з ядром — програмним середовищем — за допомогою портів та адаптерів. Мікросервіси можуть бути пов’язані між собою через REST (модель “клієнт-сервер”), RPC (виклик віддалених процедур, коли комп’ютерна програма викликає команду в іншому середовищі — наприклад, на іншому комп’ютері чи віртуальній машині) чи інший інтерфейс передавання команд (рис. 2.3).

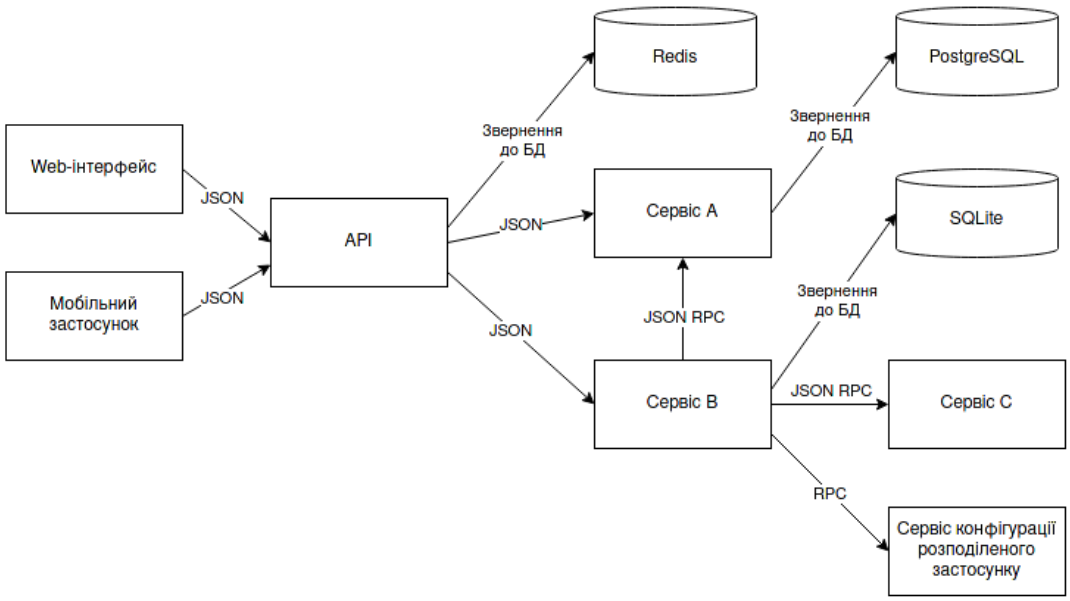


Рис. 2.3. Приклад застосунку з мікросервісною архітектурою.

Переваги мікросервісної архітектури полягають у зменшенні пов’язаності компонентів між собою — це полегшує їх замінність, покращує розуміння структури застосунку та пришвидшує розробку окремих підсистем, так як вона може відбуватись незалежно. За її допомогою можна використовувати різні технології (не має великого значення, на якій мові написаний мікросервіс, якщо він виконує свою функцію та взаємодіє з іншими мікросервісами через зрозумілий інтерфейс). Також така архітектура дозволяє розгортати підсистеми окремо — це спрощує постійну інтеграцію та постійне розгортання (англ. Continious Integration/Continious Deployment або CI/CD) [13].

Серед недоліків можна виділити створення додаткового шару складності — необхідність пов’язувати підсистеми між собою, ускладнює тестування та пошук помилок (для end-to-end чи інтеграційних тестів необхідно вмикати всі підсистеми разом чи окремо).

2.1.3. Вибір архітектури

Після аналізу можливих видів проєктування сервісу було вирішено використовувати монолітну багатошарову архітектуру, компоненти якої

реалізують шаблон MVC. Це пов’язано із достатнім рівнем гнучкості такої архітектури, більшою простотою розробки. Мікросервісна архітектура, на відміну від багатошарової, є більш складною для розгортання, що є критичним при розробці сервісу, потребує більше часу для розробки та складніша у відшуканні помилок.

2.2. Технологія для реалізації серверної частини

2.2.1. Java

Java — це популярна статично типізована об’єктно-орієнтована мова програмування, уперше випущена компанією Sun Microsystems у 1995 р. У своїй реалізації компілюється у байт-код, який вже інтерпретується віртуальною машиною Java (JVM) та виконується — унаслідок цього мова є платформо-незалежною. Зазвичай використовується для веброзробки, зокрема для складних сервісів з тривалим циклом розробки та підтримки, вимогою до яких є надійність, платформо-незалежність та якісна архітектура. Водночас серед недоліків можна вказати деяку архаїчність мови та повільність впровадження в ній змін, використання об’єктно-орієнтованої парадигми програмування, що не дозволяє повною мірою використовувати зручні концепції з, наприклад, мов функціонального програмування, гіршу швидкодію, ніж мов з ручним чи автоматичним управлінням пам’яттю, такі як C, C++ чи Rust. Використання збирання сміття збільшує використання мовою оперативної пам’яті.

Найпопулярнішим для мови фреймворком є Spring, який реалізує принцип інверсії управління (англ. Inversion of Control або IoC) — управління залежностями окремих частин системи передається фреймворку, покращуючи модульність та розширюваність програми. Найменшими елементами є Spring-біни — Java-класи, управління якими передається до Spring Core — модулю для реалізації інверсії контролю. Ключовими модулями є Spring Boot, який відповідає зв’язування окремих модулів між собою, та Spring Security —

управління доступами. Існує велика кількість інших модулів для роботи дотатку, наприклад, Spring MVC, Spring Cloud, JDBC (взаємодія з базою даних), Hibernate (модуль об’єктно-реляційного відображення) тощо. Серед переваг можна виділити популярність, якість підтримки фреймворку, детальність документації. Недоліком є складність фреймворку, унаслідок чого значно зменшується швидкість розробки.

2.2.2. C#

C# — мультипарадигмена мова програмування зі статичною безпечною системою типізації. Розробляється та підтримується корпорацією Microsoft з 2001 р. і виконується в екосистемі .NET Framework — платформа для різних мов (таких як C++, C#, F# та інших) та модулів (таких як Entity Framework, LINQ, WPF тощо), заснована на принципі Common Languare Runtime (CLR) — віртуальна машина, на якій в одному середовищі виконуються різні мови. Мова схожа на Java, зокрема тому, що об’єктно-орієнтована парадигма програмування є основною, збіркою сміття та кросплатформеністю, яка забезпечується середовищем. Використовується для розробки складних вебзастосунків, як і Java.

ASP .NET Core є найпопулярнішим фреймворком для цієї цілі. Йому притаманна неперервна компіляція, отже розробник не повинен додатково використовувати відповідну команду, оптимізованість для запуску у хмарному сховищі, кросплатформеність, використання NuGet пакунків для залежностей, а також пряма інтегрованість із інтегрованими середовищами розробки (IDE) Visual Studio та Visual Studio Code, підтримка асинхронності. Серед недоліків можна відзначити відсутність підтримки сторонніх бібліотек, відсутність підтримки ASP .NET Web Forms і ASP .NET Web Pages (попередніх технологій з ASP .NET).

2.2.3. Python

Python — популярна кросплатформена багатопарадигмова інтерпретована мова програмування з динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом та активно розвивається з того часу. Мова зручна для розробки застосунків різних видів, зокрема і для веброзробки. Їй притаманний чистий синтаксис — для виділення блоків використовуються переноси, зручність для розв'язання математичних проблем, розвинута система зовнішніх пакетів, кросплатформеність, підтримка асинхронності. Серед недоліків можна виділити GIL (глобальне блокування інтерпретатора), через що значно знижується ефективність використання мови при розробці багатопотокових програм, динамічна типізація — хоч мова підтримує анотації типів, вони не є обов'язковими, що в довгостроковій перспективі знижує швидкість розробки, та низька швидкодія, що притаманно для інтерпретованих мов.

Flask — мінімалістичний фреймворк для мови Python для розробки вебзастосунків з використанням підходу REST, представляючи рівень контролера в шаблоні MVC. Наявні розширення для роботи з базою даних, об'єктно-реляційного відображення, перевірки форм, різноманітних видів аутентифікації. Шляхи у REST API визначаються за допомогою анотацій. Є підтримка серверного рендерингу вебсторінок за допомогою HTML-шаблонів.

2.2.4. JavaScript

JavaScript — динамічно типізована інтерпретована об'єктно-орієнтована мова програмування, що використовує прототипне спадкування. Розробляється з 1995 року. Початково використовувалась тільки для надання вебсторінкам інтерактивності, але з появою середовища Node.js використовується також і для побудови масштабованих серверних застосунків, яке швидко набирає популярність в нових проєктах. Середовищу притаманні кросплатформеність, нативна підтримка асинхронності (неблокуючий ввід-вивід), можливість

багатопотоковості, гнучка система пакетів у NPM (Node Package Manager). Код легко конфігурувати під свої стандарти розробки, використовуючи, наприклад, prettier та eslint. Розширенням цієї мови є TypeScript — мова, яка компілюється у JavaScript і яка завдяки таким можливостям, як узагальненому програмуванню чи статичній типізації, полегшує процес розробки. Серед недоліків можна перелічити низьку швидкодію, як і для інших інтерпретовних мов, погану політику перевірки залежностей – вони хоч і можуть спростити вирішення великої кількості задач, але слабко контролюються і можуть бути неефективними чи неякісно написаними.

Node.js як середовищу притаманна мінімалістичність, на його основі побудована значна кількість фреймворків для розробки серверних додатків. Одним з них є Nest.js — MVC-фреймворк для побудови масштабованих, легких у розробці та підтримці серверних застосунків. Як і Spring, він використовує IoC, з реалізацією через впровадження залежностей (англ. Dependency Injection або DI) — процес створення залежностей делегується фреймворку, а компонент чи інший клас отримує їх як аргументи конструктора. Це дозволяє збільшити гнучкість розроблюваного застосунку, бо класи не так жорстко зв’язані між собою, полегшує майбутній рефакторинг (переписування коду), сприяє паралельній та незалежній розробці програмного продукту.

Структура застосунку на Nest.js розділена на модулі (містять в собі певну бізнес-логіку), в яких знаходяться сервіси (сама бізнес-логіка, об’єкт моделі у MVC), контролери (елементи REST API, які передають ввід користувача сервісам та HTTP-відповідь на клієнт), DTO (англ. Data Transfer Object — об’єкти для передачі даних іншим компонентам системи з відсутньою в них бізнес-логікою), а також при необхідності DAO (англ. Data Access Object — об’єкти, з використанням яких можна отримати та маніпулювати даними, наприклад, класи сутностей в ORM). Фреймворк побудований на мові TypeScript.

2.2.5. Rust

Rust — компільована мова зі строгою типізацією, яку з 2010 року розробляє Mozilla Research, частина неприбуткової організації Mozilla Foundation. Вона нагадує мову C++ швидкодією, але відрізняється орієнтованістю на безпеку і ефективність коду. На відміну від інших мов, їй притаманне автоматичне керування пам’яттю — коли змінній при оголошенні виділяється блок пам’яті, а потім після виходу з блоку коду, де вона використовувалась, автоматично викликається процедура його звільнення. Мова є багатопрадигмовою — тобто повноцінно підтримує функціональну парадигму нарівні з об’єктно-орієнтовним, метапрограмуванням і узагальненим програмуванням, заохочуючи розробляти програму шляхом написання легких зрозумілих частин коду. Мова орієнтована на легкість багатопотокового виконання програми, є розрирюваною за допомогою макросів. Так як її сирцевий код компілюється у LLVM (Low-Level Virtual Machine), то вона є кросплатформовою. Використовується як для системного низькорівневого програмування, так і для розробки додатків загального призначення, зокрема і веброзробки. Серед недоліків можна вказати жорсткі правила компіляції, які, хоч і забезпечують надійність програми, але ускладнюють швидку розробку та установлюють високий поріг для вивчення мови новачками, а також новизна мови та, відповідно, відсутність багатьох зручних бібліотек і великої спільноти розробників.

Rocket — мінімалістичний вебфреймворк, який, як і Flask, можна розглядати як контролер у шаблоні MVC. Фреймворк підтримує весь стандартний для фреймворків такого типу функціонал, як створеня REST API, підтримку різних MIME-типів у HTTP-відповіді, користувацькі HTTP-відповіді про помилки тощо.

2.2.6. Обрання мови програмування та MVC-фреймворку

Після огляду можливих технологій для розробки було вирішено

використовувати фреймворк Nest.js для серверної частини у зв’язку зі зручністю розгортання, зручним пакетним менеджером та більшим досвідом роботи із даною технологією. Мови Java, C# та відповідні для них технології були відкинуті через гірший рівень знайомства з ними, Python через менш зручну підтримку асинхронності, аніж у Node.js, а Rust через низьку швидкість розробки.

2.3. Підсистема зберігання даних

2.3.1. Вибір системи керування базою даних

Система керування базою даних (СКБД) — програмне забезпечення, що надає доступ користувачу до бази даних, забезпечуючи управління нею та надаючи інструменти для роботи з даними: операцій CRUD — створення (англ. create), читання (read), оновлення (update) і видалення (delete).

Найбільш популярними є реляційні бази даних, які вперше були запропоновоні Едгаром Ф. Коддом у 1970 р. За його визначенням, реляційна БД — це цифрова база даних, заснована на реляційній моделі даних. Дані зберігаються в таблицях, що вміщують рядки — окремі записи, та стовпці — атрибути різного типу даних, які представляють записаний об’єкт [14]. Щоби уникнути повторення та покращити зв’язність даних, застосовується нормалізація бази даних. Для запитів використовується мова SQL (англ. Structured Query Language). Також популярні об’єктні БД, в яких інформація представляється у формі об’єктів. Вони розроблені, щоби гарно взаємодіяти з об’єктно-орієнтованими мовами програмування.

Перевага реляційних СКБД полягає у легкості виконання складних запитів та маніпуляцій над даними та забезпеченості атомарності, узгодженості, ізольованості і довговічності транзакцій — набору елементарних дій, які формують єдину цілісну операцію. Серед недоліків можна виокремити гіршу розширюваність та масштабованість.

Прикладами найпопулярніших реляційних СКБД є Oracle DB, PostgreSQL, MySQL, MariaDB, Microsoft SQL Server, SQLite. PostgreSQL і Oracle DB є зокрема і об’єктними. Переваги PostgreSQL полягають у вільності використання, на відміну від Oracle DB і Microsoft SQL Server, надійності, масштабованості та безпеці.

З 2000-х рр. набувають популярність non SQL (NoSQL) бази даних. Їх використання зростає через такі причини, як краща масштабованість, більша швидкість розробки, гнучкість для зберігання слабко структурованих даних. Існує базліч видів NoSQL баз даних, найбільш поширеними з яких є [15]:

1. Документні — зберігають дані в документних об’єктах, схожих за форматом з JSON (JavaScript Object Notation). За ключами зберігаються значення різних типів, зокрема об’єкти більш низької ієрархії. Прикладом такої СКБД є MongoDB.
2. Бази даних “ключ-значення” — більш простий тип баз даних, в яких за кожним унікальним ключем зберігається окреме значення. Використовуються для кешування чи для даних, операції над якими не потребують складних запитів. Наприклад, такими СКБД є Redis чи DynamoDB.
3. Графові бази даних — дані зберігаються у вершинах графа, а ребра відображають відношення між ними. Використовуються у машинному навчанні, для відображення графових відношень тощо.
4. Стовпцеві — зберігають дані у таблицях, рядках та динамічних стовпцях. Стовпці є набором атрибутів, які може містити об’єкт, що зберігається, і цим такі БД є більш гнучкими за реляційні. Прикладом є Cassandra чи HBase.

Із порівняння найпопулярніших СКБД можна зробити висновок, що нереляційні СКБД гірше підходять для реалізації сервісу. Причиною є чітка структурованість та зв’язаність даних, що зберігатимуться, а також необхідність

виконання комплексних запитів. У випадку використання підходу сесій для підсистеми аутентифікації є зміст використовувати базу даних “ключ-значення” [16].

З аналізу підходів вирішено використовувати PostgeSQL у зв’язку з тим, що реляційні СКБД краще підходять для реалізації розроблюваного сервісу, вільністю використання, наявності значної кількості функціоналу та більшим досвідом роботи з даною системою.

2.3.2. Вибір системи об’єктно-реляційного відображення

ORM (Object-Relational Mapping) — це технологія програмування, що поєднує базу даних та концепції об’єктно-орієнтованих мов програмування. Таким чином створюється “віртуальна об’єктна база даних”, яка взаємодіє із сутностями, збереженими в базі даних, як із об’єктами, і які розробник може використовувати на рівні бізнес-логіки.

Серед переваг можна виділити те, що в такому випадку схема БД явно описана в термінах мови, яка використовується, і генерування запитів відбувається автоматично — це дозволяє уникнути написання однотипного програмного коду. Серед недоліків можна виділити виникнення додаткового шару абстракції — об’єктів ORM, можливість генерування реалізацією ORM неоптимізованого та несприйнятного людиною SQL-коду та необхідність вивчення об’ємної документації, яка в залежності від реалізації відрізняється [17].

Існує багато реалізацій цієї технології. Для середовища node.js існують такі ORM, як Prisma, Sequelize, Typeorm, Mongoose (виключно для MongoDB) тощо. Усі перелічені, окрім останньої, підтримують найпопулярніші реляційні СКБД, такі як PostgreSQL, MySQL, Microsoft SQL Server тощо. Вони відрізняються програмним інтерфейсом взаємодії з БД, представленням об’єктів та відношень між ними, часом розробки та підтримки, використанням різних шаблонів взаємодії з БД (таких як Active Record чи Data Mapper) тощо.

У якості ORM вирішено використати Sequelize — модуль інтегрований з Nest.js (рис. 2.4), наявна підтримка транзакцій, безвідкладне та ліниве завантаження моделей (eager and lazy loading — негайне завантаження до використання та тільки коли вони будуть потрібні).

Рис. 2.4. Приклад моделі Sequelize в проєкті з Nest.js.

2.3.3. Вибір файлового сховища

Amazon S3 (Amazon Simple Storage Service) — масштабоване об’єктне сховище для зберігання та отримання даних. Надає функціонал зберігання даних через контейнери (англ. “buckets”), що слугують базою даних “ключ-значення” і в які можна поміщувати об’єкти різного типу. Наявна можливість управляти доступом читання та запису та зберігати дані у різних регіонах. Доступ до даних стандартно відбувається через REST API, наявна підтримка застарілого SOAP-інтерфейсу.

Адреса *https://examplebucket.s3.us-west-2.amazonaws.com/photos/parrot.jpg*, наприклад, ідентифікує JPG-файл *parrot.jpg*, який знаходиться за ключем

photos/parrot.jpg у контейнері *examplebucket* в регіоні *us-west-2* (“Захід США”). Ідентифікація типу відбувається через стандартний заголовок *Content-Type* у HTTP-відповіді [18].

Дане сховище підходить для зберігання файлових даних сервісу, таких як фотографії у контрольному списку.

2.4. Технологія для реалізації клієнтської частини

2.4.1. Підхід до розробки клієнтської частини

Односторінковий додаток (англ. Single-Page Application або SPA) — це підхід до розробки вебзастосунків, коли користувачу завантажується лише одна вебсторінка з мініфікованим JavaScript-кодом (“бандлом”), яка потім оновлюється шляхом запитів до сервера через виклики таких JavaScript API, як XMLHttpRequest чи Fetch [19].

Використання підходу дозволяє користувачу використовувати вебсайти, не завантажуючи щоразу нові сторінки з сервера, динамічно оновлюючи лише частину однієї сторінки — це може призвести до підвищення продуктивності (рис. 2.5). Водночас, цьому підходу притаманні певні мінуси, такі як гірша пошукова оптимізація (SEO) вебсайту. Також для зберігання стану застосунку необхідно докладати більше зусиль, імплементувати навігацію додатком та проводити моніторинг продуктивності. Зазвичай він використовується для

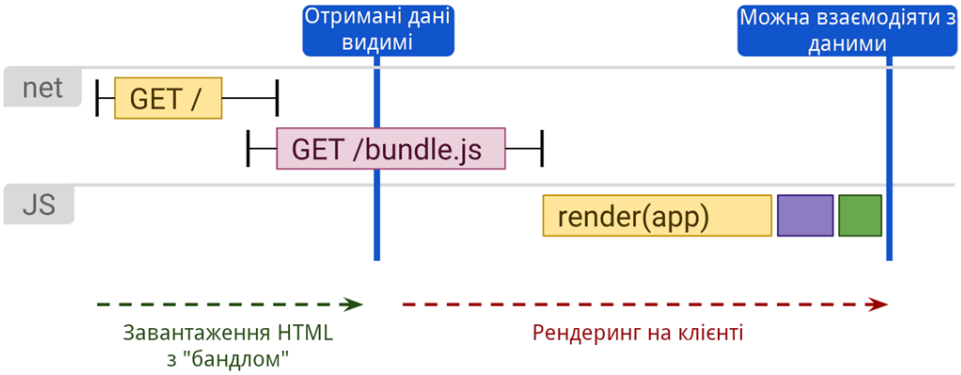


Рис. 2.5. Принцип відображення SPA у браузері.

побудови SaaS-сервісів та інших застосунків, що працюють у браузері. Даний принцип був запатентований в Сполучених Штатах Америки у 2002 році [20].

Багатосторінковий додаток (Multi-page Application або MPA) — більш традиційний підхід розробки, за яким вебзастосунок складається з багатьох сторінок, які завантажуються з сервера. На противагу SPA, даному підходу притаманні переваги кращої пошукової оптимізації та краща масштабованість застосунку, але водночас гірша швидкодія і довший час розробки, через те, що при даному підході вебдодаток більш сильно зв’язаний з бекендом застосунку [21].

Для розробки багатосторінкових додатків застосовується підхід рендерингу на сервері (англ. Server-side Rendering або SSR). За цим підходом на сервері цілком генерується HTML-сторінка, коли користувач переходить за посиланням. Це дозволяє уникнути додаткових запитів даних для представлення їх на стороні клієнта, оскільки вся робота проводиться до того, як браузер отримає відповідь (рис. 2.6).

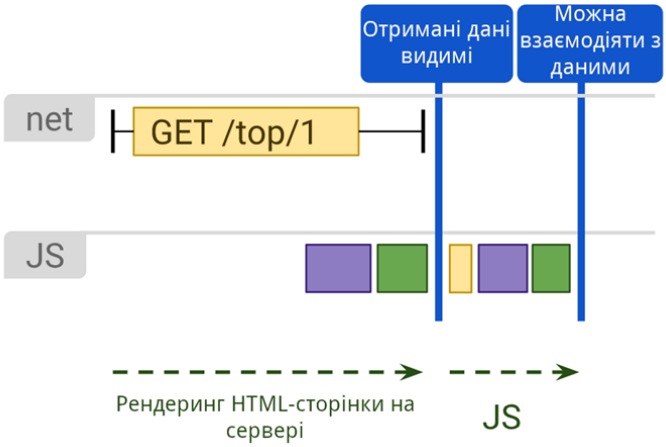


Рис. 2.6. Принцип відображення у браузері сторінок при серверному рендерингу.

Рендеринг на сервері дозволяє зменшити час між початком рендерингу сторінки та часом, коли хоча би якийсь її елемент стане видимим, а також зменшує розмір JavaScript-коду, який надсилається клієнту. Це дозволяє зменшити час між початком завантаження сторінки та часом, коли можна

взаємодіяти з будь-яким її елементом [22].

При розробці сервісу для навчання нових співробітників на підприємстві доцільно використати підхід SPA, тому що для клієнтської частини сервісу пошукова оптимізація не має значення через захищений доступ до майже всіх шляхів навігації тільки тим користувачам, що увійшли в систему. Також робота з клієнтською частиною сервісу передбачає часте оновлення частин сторінки. Те, що розроблюваний сервіс є SaaS-застосунком, також є аргументом на користь використання саме цього підходу.

Таблиця 2.1. Порівняння підходів розробки клієнтських застосунків.

	SPA	SSR
SEO	Гірша оптимізованість, що не грає великої ролі для розроблюваного сервісу	Краща оптимізованість
Генерування вебсторінок	Повністю на клієнті	Повністю на сервері
Роль сервера	Відправляє на клієнт одну сторінку з JavaScript-бандлом, завантажує дані за запитом клієнта	Відправляє на клієнт повністю згенеровану HTML-сторінку
Інтерактивність	Підходить для інтерактивних вебсторінок	Підходить гірше
Використання	SaaS-застосунки, прогресивні вебзастосунки, вебзастосунки, для яких SEO має менше значення	Вебсайти з невисокою інтерактивністю, багатосторінкові додатки

2.4.2. Вибір технології для розробки односторінкового застосунку.

Станом на 2021 рік для розробки односторінкових застосунків наявна велика кількість рішень, які полегшують їх розробку. В даному розділі оглянуто три найпопулярніші засоби, такі як бібліотеку React, фреймворки Angular, Vue.js та мову Dart.

React — бібліотека для побудови користувацьких інтерфейсів з відкритим вихідним кодом. Розробляється компанією Facebook з 2013 року та була створена з метою бути швидкою, масштабованою і простою у використанні. Водночас, бібліотека використовується не тільки для веброзробки, а в поєднанні з іншими бібліотеками, представляючи тільки компонент “представлення” в шаблоні проєктування MVC. Як приклад, React Native — фреймворк для мобільної розробки на JavaScript, в Electron — для десктопних застосунків, чи React 360 — для розробки застосунків віртуальної реальності. Зазвичай використовується в поєднанні з ReactDOM — JavaScript-бібліотекою для рендерингу інтерфейсу, побудованого за допомогою React, і маніпуляцій об’єктною моделлю HTML-документа (англ. Document Object Model або DOM) [23].

Особливості бібліотеки полягають у її заснованості на компонентах — функціональних елементах з певною бізнес-логікою, з яких будується користувацький інтерфейс. Для них зазвичай використовується JSX — HTML-подібне представлення компонентів, яке не є обов’язковим і є синтаксичною надбудовою для JavaScript і перетворюється в нього на етапі компіляції [24]. Для розробки SPA-застосунків на React зазвичай використовується модуль create-react-app для платформи Node.js. З його допомогою можна ініціалізувати клієнтський застосунок з налаштованим компілятором Babel, гарячою заміною (зміною в інтерфейсі одразу під час зміни файлу сирцевого коду), з налаштованою структурою файлів, папок та репозиторію git, якщо ця система контролю версій установлена на пристрої розробника [23]. Часто використовується з модулем Redux для управління станом застосунку. Також присутня можливість згенерувати додаток з використанням TypeScript [25].

Angular — фреймворк для побудови односторінкових застосунків, який розробляється компанією Google з жовтня 2010 р. Спершу називався AngularJS, а з вересня 2016 року був переосмислений та переписаний як Angular тією ж командою розробників і активно підтримується станом на 2021 рік [26]. Це

MVVM-фреймворк для розробки великих, складних та масштабованих клієнтських застосунків. Побудований на мові TypeScript.

На відміну від React, Angular є повноцінним фреймворком, надаючи вбудовані модулі для клієнтської навігації, обробки форм, взаємодії з сервером, реактивного програмування тощо. Використовує компонентний підхід – в кожному з компонентів міститься потрібна бізнес-логіка, HTML-шаблон для представлення та CSS (англ. Cascading Style Sheets — каскадні таблиці стилів). У Angular активно використовується шаблон впровадження залежностей (DI). Також наявний інтерфейс командного рядка (Angular CLI), який полегшує створення початкового програмного проєкту, його компілювання, виконання та запуск тестів [27].

Vue.js — вебфреймворк для побудови інтерфейсів користувача. Розроблений Еваном Ю, колишнім співробітником Google, коли виникла потреба в компактному інструменті для уникнення написання повторюваної HTML-розмітки. На той час AngularJS був занадто громіздким, а React тільки починався як проєкт. Перший реліз відбувся в грудні 2014 року, і з тих пір фреймворк активно розробляється та підтримується. Як і у двох вищезгаданих інструментів, для фреймворка наявний інтерфейс командного рядка для збирання, компіляції та запуску проєкту [28]. В ньому також використовується компонентний підхід, і, на відміну від традиційного підходу в React, стилі CSS замкнені на окремому компоненті (стосуються тільки його). Як і в React, використовується віртуальна DOM — техніки та бібліотеки для покращення продуктивності під час маніпуляції з DOM, працюючи з полегшеним JavaScript-об’єктом, що імітує дерево документа. Дані прив’язані до об’єктної моделі документа і програма керується потоком даних (є реактивною), що є схожістю з Angular. Йому так само притаманні директиви, які використовуються для маніпуляції з DOM. Використання TypeScript не є обов’язковим, фреймворк можна легше оптимізувати та він швидше у вивченні [29].

Dart — розроблювана Google з 2011 року об’єктно-орієнтована мова програмування для клієнтської та серверної розробки. Має синтаксис, схожий з мовою Java та є статично типізованою мовою, в якій типи автоматично виводяться з присвоюваного змінній значення. Може компілюватися як у машинний код, так і в JavaScript для клієнтської розробки. Для мови розроблено Flutter framework, який використовується для побудови як кросплатформених десктопних і мобільних, так і вебзастосунків [30].

Таблиця 2.2. Порівняння технологій для односторінкових застосунків.

	React	Angular	Vue.js	Dart
Інструменти розробника (CLI)	Присутні	Присутні	Присутні	Присутні
Єдина структура всіх проєктів	Нема єдиної структури	Так	Нема єдиної структури	Нема єдиної структури
Віртуальна DOM	Так	Ні	Так	Ні
Використання TypeScript	Можлива	Обов’язкова	Можлива	—
Ізольовані стилі CSS	Ні, можна використовувати модулі styled-components чи css-modules	Так	Так	Ні
Управління станом програми	Параметри компонентів (пропси) чи за допомогою Redux	Параметри компонентів, RxJS (модуль для реактивного програмування) чи Mobx, Redux.	Параметри компонентів чи бібліотека Vuex	Використання методів реактивного програмування у Flutter

Продовження таблиці 2.2

	React	Angular	Vue.js	Dart
Локалізація іншими мовами	Модуль react-localization чи react-i18n	Вбудована	Модуль vue-i18n	Пакет Flutter-localizations

Після порівняння технологій було вирішено використати React для розробки клієнтської частини сервісу, так як йому притаманна краща налаштовуваність для використання супутніх технологій, більша швидкість та простота розробки, а також через більший досвід використання цієї технології.

2.4.3. Вибір технології для дизайну

У сучасній веброзробці зручний дизайн інтерфейсу клієнтських додатків є важливим фактором для досягнення успішності програмного продукту й активно розвивається. Для зовнішнього дизайну вебсторінок існує велика кількість бібліотек, які полегшують процес розробки, забезпечують адаптивність — такий дизайн вебсторінок, який надає оптимальне представлення на пристроях з різною роздільною здатністю. Вони надають стандартні шаблони для стилізації різних компонентів користувацького інтерфейсу, забезпечуючи кращий досвід користування системою (англ. User Experience або UX). Зазвичай бібліотеки пропонують розроблені класи стилів CSS, які застосовуються на HTML-теги, надаючи їм потрібний дизайн та спрощуючи роботу розробника, якому потрібно менше працювати над забезпеченням адаптивністю інтерфейсу чи стилізацією тривіальних об’єктів.

Для дизайну вирішено використати бібліотеку Bootstrap. Це клієнтський фреймворк для побудови адаптивних вебінтерфейсів з mobile-first підходом — тобто програмний продукт початково розробляється для мобільних пристроїв [31]. Це відповідає вимозі адаптивності до розроблюваного сервісу. Для

інтеграції з React існує модуль react-bootstrap, який надає відповідні компоненти для стандартних класів стилів [32].

В проєктуванні React-застосунків набуває популярність використання atomic-дизайну. Даний підхід був уперше запропонований у статті Брода Фроста у червні 2013 року та полягає у декомпозиції елементів дизайну інтерфейсу на елементарні блоки — “атоми”, наприклад, кнопка чи поле вводу (рис. 2.7). Далі вони структуруються у більш складні компоненти: “молекули” (об’єднання “атомів”, такі як форма для пошуку), “організми” (комплексні елементи інтерфейсу, як бічне меню) та “шаблони” — групи “організмів”. Кінцевим етапом є створення сторінок — екземплярів “шаблонів”, які й бачить користувач під час користування інтерфейсом. Вони використовуються для тестування дрібніших компонентів.

Рис. 2.7. Принцип побудови atomic-дизайну.

Atomic-дизайн надає чітку послідовність розробки клієнтського застосунку, покращує його гнучність та розширюваність [33]. Цей підхід вирішено використати для розробки клієнтської частини сервісу.

2.5. Технології для розгортання застосунку

Хмарні обчислення — це надання обчислювальних потужностей, сховища баз даних, додатків та інших ресурсів на вимогу з використанням інтернет-платформи хмарного сервісу, оплата за користування яким відбувається за фактом споживання. Платформа хмарних сервісів забезпечує швидкий доступ до

програмних ресурсів для будь-яких завдань: наприклад, додатки, що дозволяють поділитися фото з користувачами на мобільних пристроях, чи програмне забезпечення для обслуговування критично важливих робочих процесів підприємства. Хмарні обчислення зменшують залежність користувачів від апаратного ресурсу та час на налаштування та підтримку програмного продукту [34].

Ідея надавати програмне забезпечення в доступ через Інтернет-мережу не є новою та походить ще з операційних систем розподіленого часу для мейнфреймів у 1960-х і 1970-х рр. та хостингу додатків з 1980-х і 1990-х. Підхід Software as a Service (SaaS), як і більш загальні інфраструктурні технології, на основі яких функціонує дане програмне рішення, давно стали провідними для використання в комерційних програмних продуктах, складаючи конкуренцію більш традиційним десктопним чи мобільним додаткам, які працюють безпосередньо на пристрої користувача [35].

2.5.1. Docker

Docker — це програмне забезпечення для швидкої розробки, тестування і розгортання додатків, на рівні віртуалізації операційної системи, запаковуючи програмне забезпечення у пакети, які називаються контейнерами. Кожен контейнер містить все необхідне для роботи програми: середовище виконання, бібліотеки і сирцевий код. Docker дозволяє швидко розгортати і масштабувати застосунки в будь-якому середовищі, забезпечуючи виконання програми на різних пристроях та платформах. Розробляється та підтримується компанією Docker, Inc. з березня 2013 року [36].

2.5.2. AWS

AWS (Amazon Web Services) — це дочірня компанія Amazon, яка в оренду приватним особам надає платформу хмарних обчислень, компаніям та урядам на

основі платної підписки. Була заснована у березні 2006 року та з тих пір стала найбільш популярним рішенням для хмарних обчислень, займаючи станом на перший квартал 2021 року третину ринку хмарних обчислень, у порівнянні з Microsoft Azure та Google Cloud [37]. Даний інструмент використовуватиметься для розміщення сервісу “у хмарі”.

Amazon Elastic Container Service (Amazon ECS) — масштабований високопродуктивний сервіс для управління контейнерами Docker. Сервіс дозволяє розробнику запускати та масштабувати контейнеризовані додатки на AWS, використовуючи інтерфейс взаємодії через прості запити до API [38].

Amazon Elastic Compute Cloud (Amazon EC2) — це вебсервіс, що надає захищені та масштабовані обчислювальні ресурси у хмарі, спрощуючи хмарні обчислення для розробників. Він надає програмісту повний контроль над обчислювальними ресурсами і дозволяє запускати застосунки в обчислювальному середовищі Amazon. Користувачу надаються у доступ віртуальні машини у хмарі, до яких він має доступ на рівні користувача операційної системи, маючи змогу запускати потрібні обчислення та програмне забезпечення, зокрема розгортати SaaS-застосунки [39]. Підключення користувача відбувається через SSH (англ. Secure Shell — “безпечна оболонка”, мережевий протокол для безпечного віддаленого управління комп’ютером).

Обидва сервіси — Amazon EC2 та Amazon ECS є взаємозамінними для використання, але Amazon EC2 гнучкіший у налаштуванні та надає програмісту більшу свободу дій, так як надає рівень управління на рівні користувача ОС, тому вирішено використати його.

ВИСНОВКИ ДО РОЗДІЛУ 2

Після аналізу поширених шаблонів проєктування SaaS-застосунків та огляду наявних технологій для розробки вебдодатків було вирішено використовувати монолітну багатошарову архітектуру, реалізуючи шаблон проєктування MVC. На рівні представлення використовується React з використанням atomic-дизайну при розробці та Bootstrap у якості CSS-бібліотеки для дизайну. На рівні бізнес-логіки використовується MVC-фреймворк Nest.js для середовища Node.js. Прийнято рішення використовувати об’єктно-реляційне відображення на рівні доступу до даних, а саме модуль Sequelize, PostgreSQL у якості СКБД, а також Amazon S3 для зберігання файлових даних. Для розгортання застосунку використовується Docker та Amazon EC2.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВІСУ

3.1. Основні сценарії взаємодії користувача із сервісом

Для основних сценаріїв взаємодії з системою накреслено діаграми послідовності (англ. Sequence Diagram) — різновид UML-діаграм для відображення задіяних у певному процесі елементів та сценарію їх роботи.

При реєстрації (рис. 3.1) користувач вводить своє ім’я, e-mail, нікнейм (спосіб ідентифікувати користувача при згадуванні в повідомленнях чи при пересиланні даних на сервер, щоб уникнути зайвого використання e-mail) та пароль. Відбувається валідація даних на стороні клієнта: наприклад, e-mail повинен бути записаний у відповідному форматі та є обмеження на мінімальну довжину паролю.

Після передавання даних про користувача в базі даних створюється неверифікований користувач, для підтвердження йому надсилається e-mail із запрошенням перейти за посиланням верифікації. При переході за посиланням користувач перенаправляється на сторінку з підтвердженням та отримує запрошення зареєструвати підприємство чи дослідити функції сервісу.

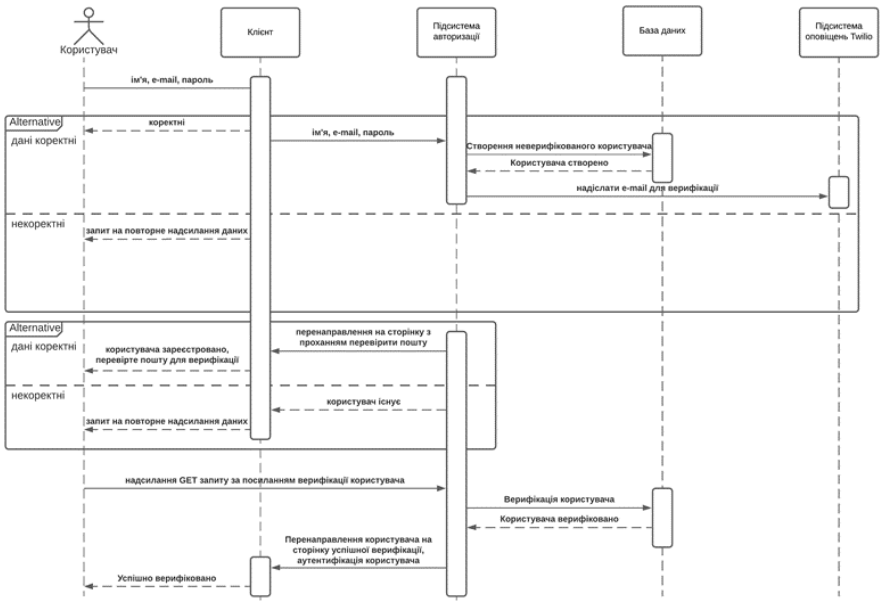


Рис. 3.1. Послідовність процесу реєстрації користувача.

При вході в систему (рис. 3.2) користувач надсилає свій e-mail та пароль. У серверній частині підсистема взаємодії з БД знаходить користувача за e-mail. Отриманий з клієнта пароль зіставляється зі збереженим у базі. У випадку невідповідності користувачу надсилається повідомлення про помилку. Інакше у підсистемі авторизації для користувача створюється підписаний токен JWT та надсилається у заголовку Authorization з HTTP-відповіддю на клієнт.

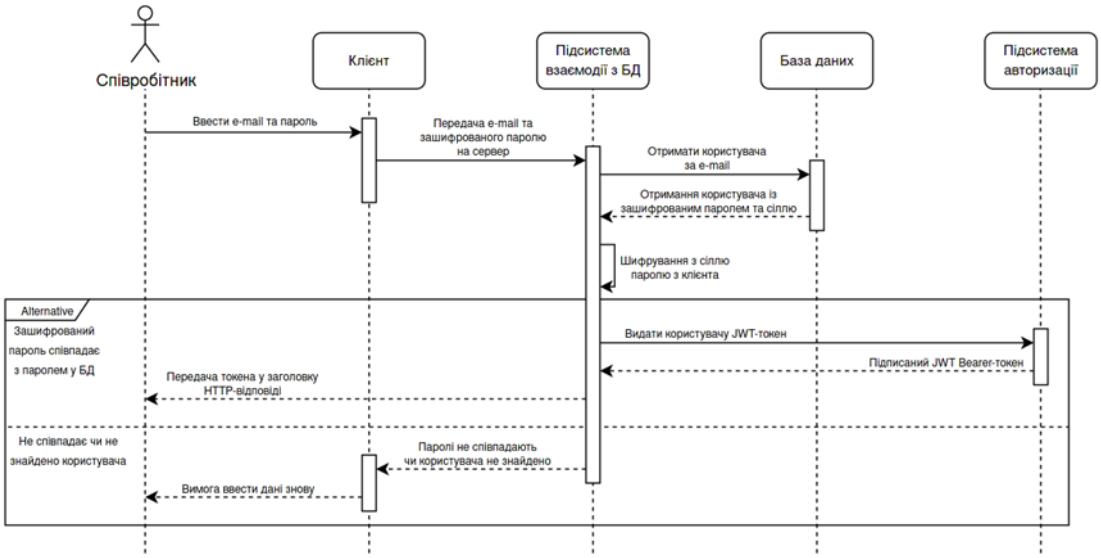


Рис. 3.2. Послідовність процесу входу користувача в систему.

Користувач реєструє підприємство за назвою, вказуючи також країну, яке зберігається в базі. Цей користувач отримує в системі роль адміністратора підприємства — тобто він має можливість змінювати ролі інших користувачів. Користувачу надсилається e-mail-підтвердження про реєстрацію підприємства.

Додавання нового співробітника на підприємство (рис. 3.3) є складнішим процесом, бо новий співробітник як міг використовувати сервіс раніше, так і може мати необхідність зареєструватись. Менеджер чи адміністратор заповнює початкові дані про співробітника (ім’я та e-mail, права та, за потреби, нікнейм ментора). В базі даних зберігається запрошення з токеном його підтвердження. Також за e-mail перевіряється, чи був користувач зареєстрований раніше.

Якщо користувач не був зареєстрований раніше, то йому за e-mail надсилається запрошення реєстрації з токеном в рядку запиту (англ. Query

String). Коли користувач переходить за посиланням, він заповнює решту даних про себе, а за токеном потім запрошення підтверджується. Якщо користувач був зареєстрований, то просто відбувається підтвердження запрошення — сутність користувача зв’язується з підприємством, користувач отримує права та ментора, вказані в запрошенні.

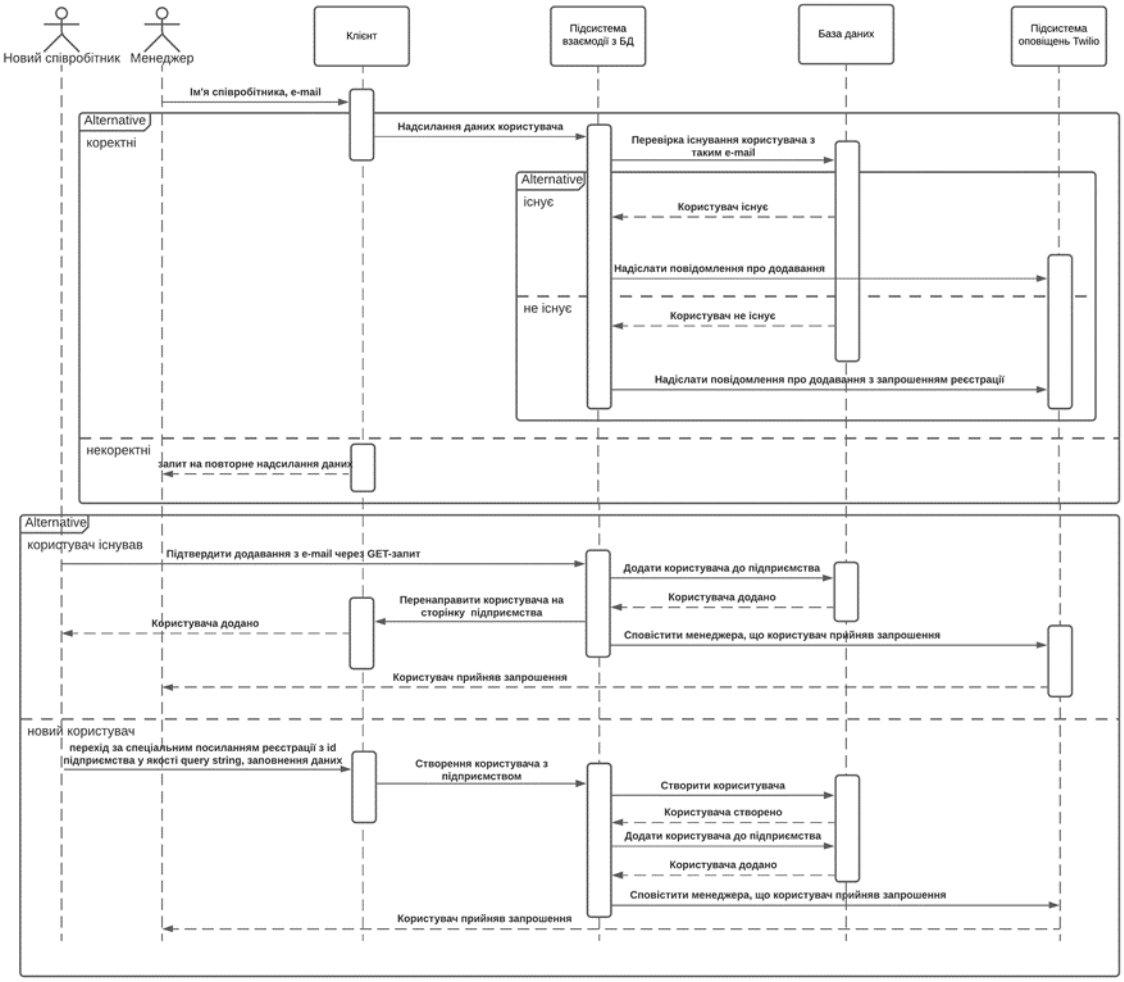


Рис. 3.3. Послідовність процесу додавання нового співробітника

Для управління робочими задачами для нових співробітників використовується метод контрольних списків: описів необхідних дій співробітника при виконанні певної задачі. Для створення їх шаблонів (рис. 3.4), які потім можна призначити співробітникам, менеджер або адміністратор створює його на стороні клієнта, за необхідності додаючи мультимедійні дані: посилання на відео, завантажує зображення чи документ. Якщо серед доданих

даних є файли, то використовується multipart-запит, тобто з файловими даними. Ці дані зберігаються на Amazon S3, а їх адреси зберігаються в базі даних.

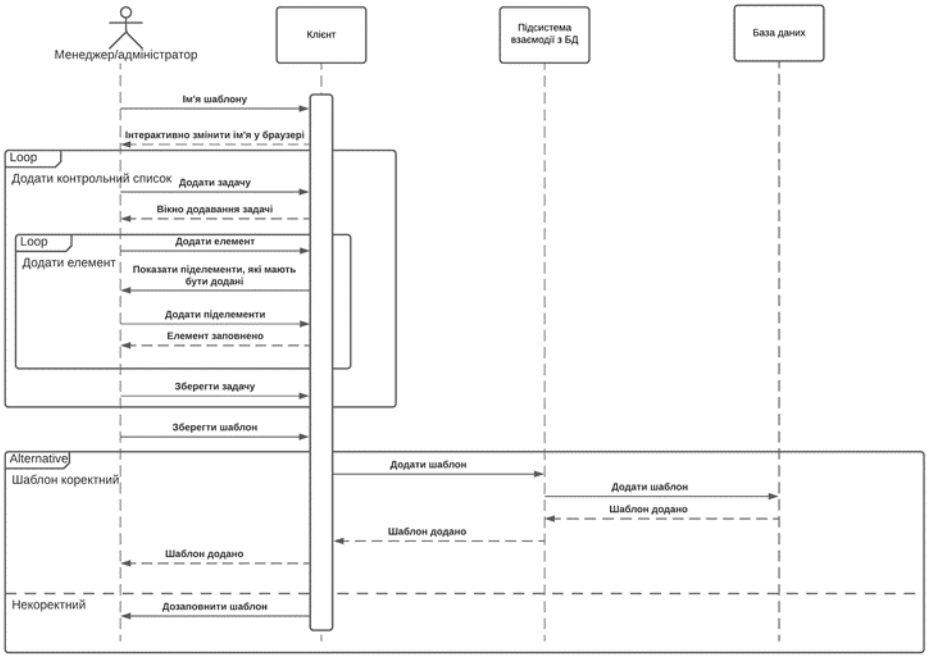


Рис. 3.4. Послідовність процесу створення шаблону задач.

Після додавання нового співробітника йому можна назначити задачі зі списку шаблонів (рис. 3.5). Менеджер чи адміністратор переходить до списку шаблонів і вибирає потрібний. У формі призначення він вказує нікнейм співробітника. Підсистема взаємодії з БД визначає співробітника за нікнеймом та створює для нього незаповнений контрольний список.

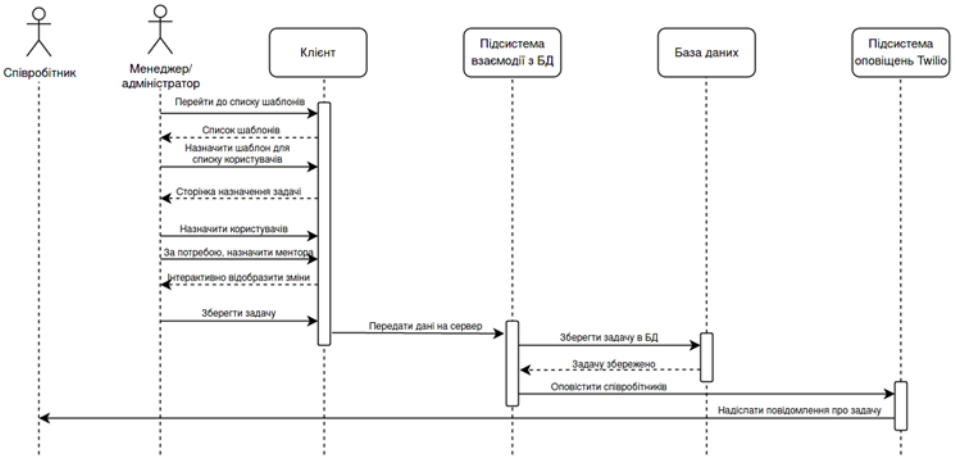


Рис. 3.5. Послідовність процесу призначення співробітнику нової задачі.

Новий співробітник при виконанні задачі проглядає контрольний список зі списком задач, вивчаючи необхідну інформацію та за потреби заповнюючи його (рис. 3.6). Користувач при перегляді контрольного списку має можливість оновити записану інформацію та позначити пункт задачі чи її цілком виконаною. Зміни записуються в базу даних, при заповненні всього контрольного списку менеджеру надходить сповіщення.

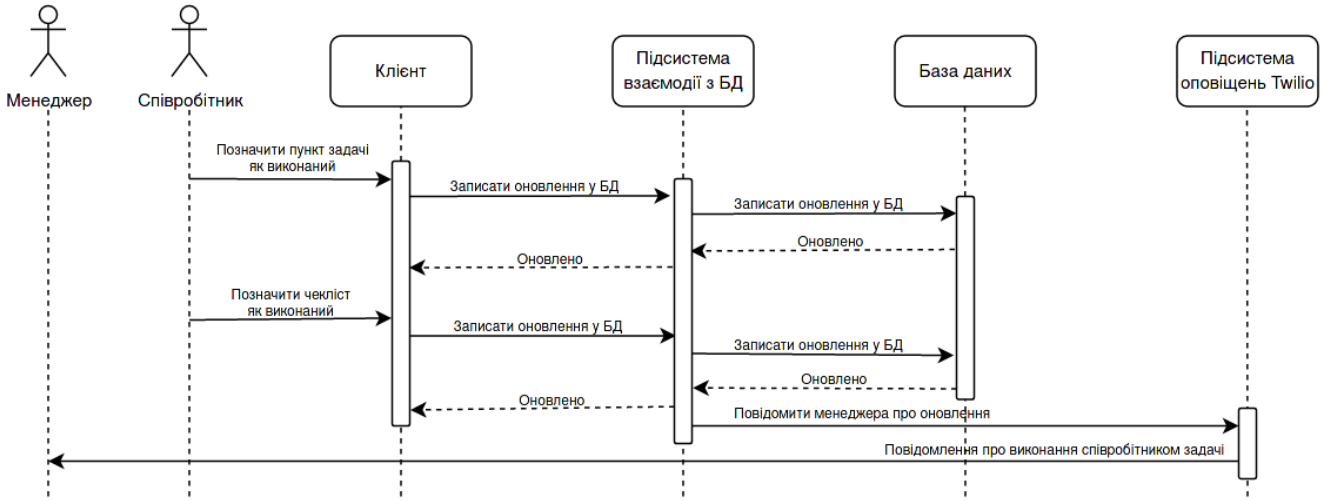


Рис. 3.6. Послідовність процесу заповнення контрольного списку.

Ментор для нового співробітника — це та людина, яку він може запитати в будь-який час та отримати відповідь. В клієнтській частині сервісу є вебінтерфейс для простого чату між ними. При надсиланні повідомлення воно записується в базу даних, а ментор отримує сповіщення про це (рис. 3.7).

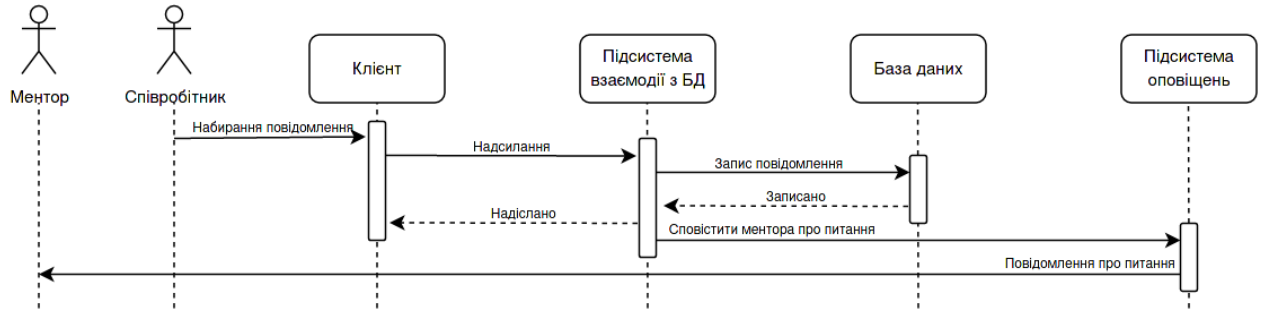


Рис. 3.7. Послідовність процесу спілкування з ментором.

Важливою частиною розроблюваного сервісу є можливість перегляду та формування звітності. Менеджер чи адміністратор має можливість вибрати параметри для звітування (такі як вибрати конкретних співробітників, за конкретним шаблоном контрольних списків чи час) і відправити запит на сервер. Генерується електронна таблиця з даними і відправляється менеджеру електронною поштою (рис. 3.8).

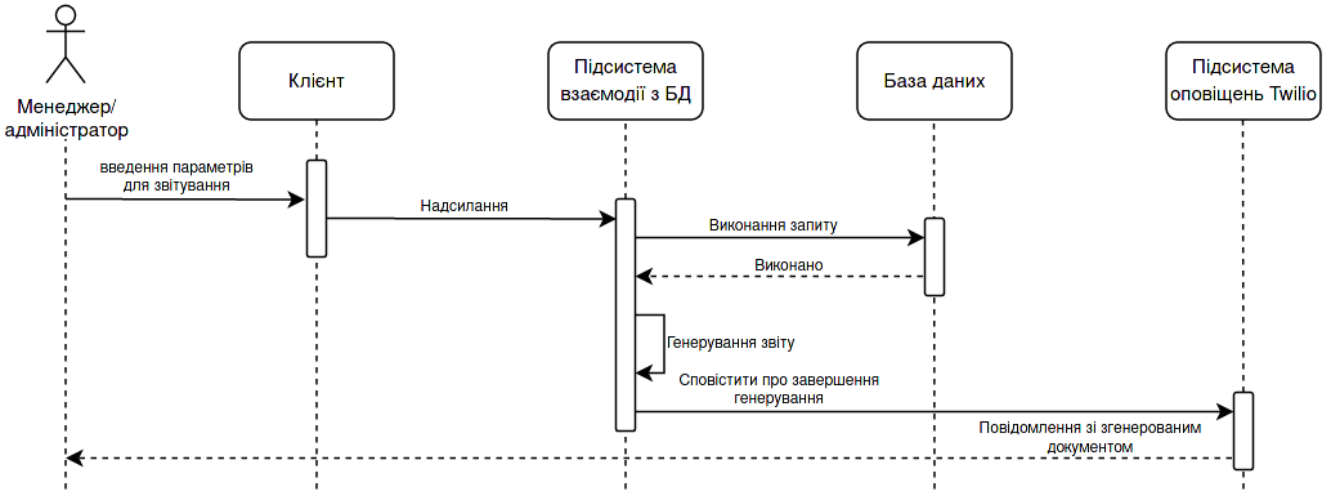


Рис. 3.8. Послідовність процесу генерування звітності.

3.2. Опис бази даних

У PostgreSQL було створено таблиці з використанням механізму міграції баз даних — способу оновити структуру бази даних, синхронізуючи її стан зі станом рівня доступу до даних застосунку: наприклад, додати нову таблицю чи змінити тип у стовпці з конвертування існуючих даних. Для цього було використано sequelize-cli — модуль, який дає змогу створювати міграції та, під’єднавшись до бази даних, виконувати міграції чи відкидати невдалі дії (рис. 3.9).

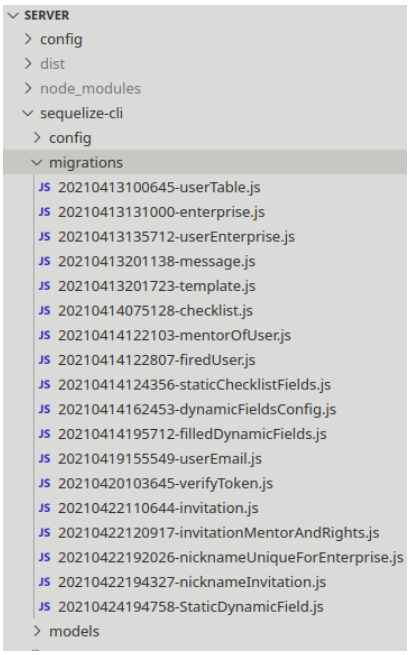


Рис. 3.9. Структура міграцій бази даних у розробленому проєкті.

В базі даних були створені таблиці, в яких зберігаються необхідні дані. В кожній таблиці, окрім перерахованих, є стовпці *created_at* та *deleted_at* — вони використовуються для більшої безпеки зберігання даних: при “видаленні” рядка в *deleted_at* записується час видалення — після цього даний рядок не вибирається та може як бути видаленим по-справжньому, так і відновленим.

Таблиця 3.1. Опис таблиць бази даних

Назва таблиці та опис	Стовпець	Опис стовпця
user – таблиця з даними для входу користувача.	id	Унікальне ID користувача. Натуральне значення з автоінкрементуванням.
	first_name	Ім’я.
	last_name	Прізвище.
	email	Адреса електронної пошти.
	password_hash	Зашифрований пароль.
	password_salt	“Сіль” для паролю — використовується при його розшифруванні.
	verified	Чи є користувач підтвердженим.

Продовження таблиці 3.1

Назва таблиці та опис	Стовпець	Опис стовпця
enterprise – загальні дані підприємства.	id	Унікальне ID підприємства.
	name	Назва підприємства.
	country	Країна. Є перерахуванням, тобто може приймати певне рядкове значення з набору “Ukraine”, “Poland”, “USA” тощо.
	creator_id	id користувача, який створив підприємство. Зовнішній ключ до таблиці user.
user_enterprise – дані співробітника підприємства.	id	Унікальне ID співробітника підприємства.
	user_id	id користувача. Зовнішній ключ до таблиці user.
	enterprise_id	id підприємства. Зовнішній ключ до таблиці enterprise.
	nickname	Нікнейм користувача. Для стовпців enterprise_id та nickname існує обмеження (англ. constraint) — разом вони унікальні.
	mentor_id	id ментора. Необов’язковий стовпець. Зовнішній ключ до власного стовпця id.
	rights	Права користувача. Перерахування, має одне зі значень “fired” (колишній співробітник), “worker” (звичайні права), “manager” (права назначати задачі, створювати шаблони контрольних списків та запрошувати нових співробітників) або “admin” (ті ж права з можливістю управляти правами інших користувачів).

Продовження таблиці 3.1

Назва таблиці та опис	Стовпець	Опис стовпця
invitation – запрошення.	token	Токен запрошення, первинний ключ таблиці. Випадкова рядкова послідовність.
	invited_user_email	E-mail запрошеного користувача. За ним ідентифікується користувач, якщо був зареєстрований раніше.
	invited_user_name	Ім’я запрошеного користувача.
	enterprise_id	ID підприємства, до якого відноситься запрошення.
	rights	Права користувача. Після прийняття запрошення копіюються в таблицю user_enterprise.
	mentor_id	ID ментора. Необов’язковий стовпець. Також копіюється в user_enterprise.
	accepted	Булеве значення, чи було запрошення прийнято.
message – повідомлення.	id	Унікальне ID повідомлення.
	from_user_id	ID відправника. Зовнішній ключ до user_enterprise.
	to_user_id	ID одержувача. Зовнішній ключ до user_enterprise.
	text	Текст повідомлення.
	is_read	Булеве значення, чи було повідомлення прочитано.
template –шаблон контрольного списку.	id	Унікальне ID шаблону.
	creator_id	ID автора шаблону. Зовнішній ключ до таблиці user_enterprise.
	name	Назва шаблону.

Продовження таблиці 3.1

Назва таблиці та опис	Стовпець	Опис стовпця
template_task – робоча задача, з яких складається шаблон.	id	Унікальне ID задачі шаблону.
	template_id	ID шаблону, до якого відноситься задача.
	order	Порядок задачі в шаблоні.
	name	Назва задачі.
template_task_item – елемент робочої задачі, який її описує (поле вводу, посилання на файл тощо). Може бути або статичним полем, або динамічним (із вводом користувача).	id	Унікальне ID елемента.
	template_task_id	ID задачі, до якої він відноситься.
	order	Порядок елемента.
	is_required	Чи обов’язкове поле для заповнення.
static_field – опис статичного поля, без заповнення співробітником.	id	Унікальне ID задачі шаблону.
	template_task_item_id	ID елемента, до якого відноситься.
	type	Тип поля. Може приймати значення “text_field”, “image”, “video” чи “file”.
	config	Опис властивостей поля (заголовок та вміст текстового поля, посилання на зображення тощо).

Присутня таблиця **dynamic_field**, в якій відповідним чином описуються динамічні поля, де співробітник може вводити дані, але стовпець type може приймати значення “subtasks_list”, “text_input”, “numeric_input”, “choices_list” та “file_input”. При назначенні співробітнику робочих задач створюються записи в таблицях **checklist**, **checklist_task** та **checklist_task_item**. При заповненні будь-якого динамічного поля створюється запис у таблиці **filled_dynamic_field**. Всі таблиці бази даних та відношення між ними продемонстровані на рис. 3.10.

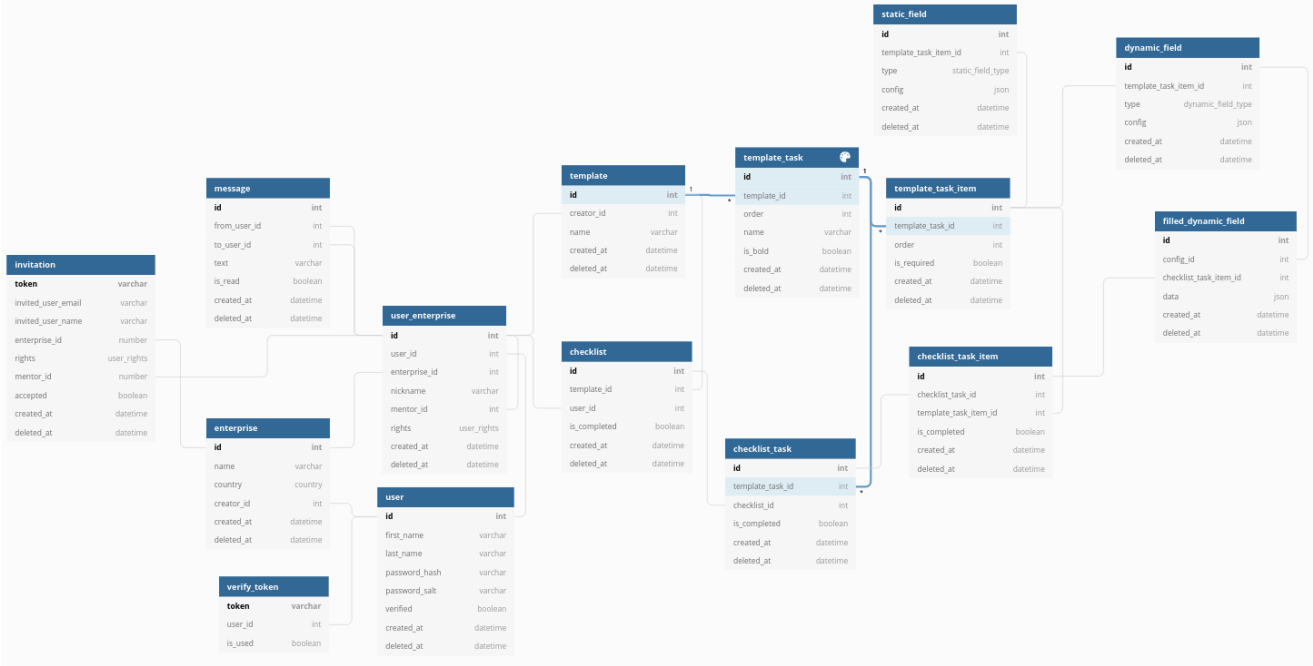


Рис. 3.10. Діаграма бази даних.

3.3. Рівні бізнес-логіки та доступу до даних

На рівні бізнес-логіки використовується NodeJS-фреймворк Nest.js, а на рівні доступу до даних — Sequelize, з ним інтегрований. Для опису проєкту використовуються файли package.json (така інформація як скрипти для стандартних дій над програмою і використані залежності) та package-lock.json (конфігурація залежностей). Файл tsconfig.json використовується для конфігурації мови TypeScript, .eslintrc.js та .prettierc — для модулів eslint та prettier, які полегшують розробку в середовищі NodeJS, задаючи правила до стилю написаного коду. Dockerfile описує послідовність дій при розгортанні застосунку з використанням Docker. Також в проєкті існують певні конфігураційні дані, які використовуються в роботі застосунку і які небезпечно включати в програмний код (такі як IP-адреса підключеної бази даних, логін та пароль доступу до неї чи секретний ключ для підпису JWT-токенів) — вони зберігаються у файлі .env, а потім з використанням модуля dotenv імпортуються в залежності окремих модулів застосунку.

В теці src знаходиться сирцевий код застосунку (рис. 3.11), розділений на

окремі частини: controllers (контролери, приймають ввід користувача і передають зміни до моделі), models (сутності бази даних), services (сторонні сервіси, такі як налаштований nodemailer для електронної пошти). В теці exceptions знаходяться користувацькі HTTP-виключення, в types типи даних для роботи з БД (такі як перерахування можливих прав користувача).

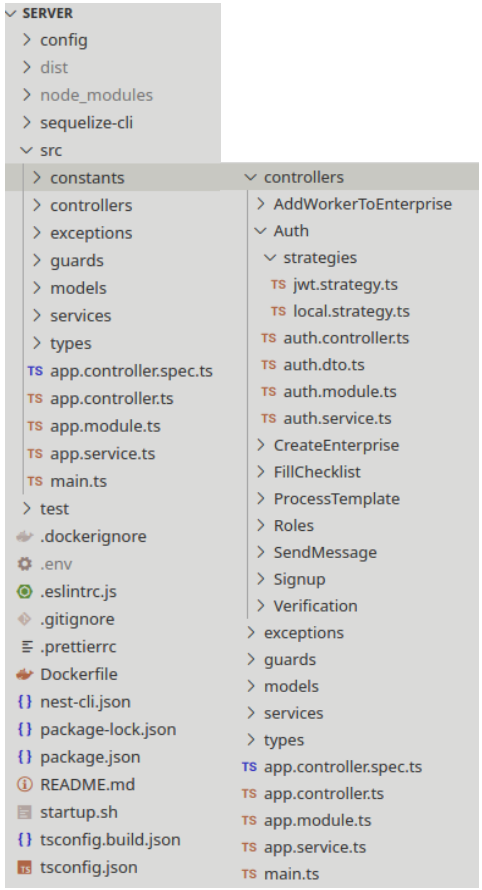


Рис. 3.11. Структура серверної частини проєкту.

На прикладі модуля авторизації розглянемо структуру проєкту. Кожна підсистема складається з окремих модулів, для опису яких використовується окремий файл модуля: в ньому описуються імпортовані залежності, елементи бізнес-логіки модуля та перелік тих із них, що надає в доступ іншим модулям, щоби ті через впровадження залежностей їх отримали. Модуль авторизації (рис. 3.12) використовує модуль роботи з користувачем, модуль роботи з JWT (JSON Web Token) — видача JWT-токена та його валідації, та модуль Passport.js (для обробки HTTP-запиту і перевірки, чи є користувач авторизованим).

```
@Module({
  imports: [
    UserModule,
    PassportModule,
    JwtModule.register({
      secret: server.JWT_SECRET,
      signOptions: { expiresIn: '12h' },
    }),
  ],
  controllers: [AuthController],
  providers: [AuthService, LocalStrategy, JwtStrategy],
  exports: [AuthService],
})
export class AuthModule {}
```

Рис. 3.12. Структура модуля авторизації auth.module.ts.

В контролерах (рис. 3.13) обробляється HTTP-запит та передається до сервісу аутентифікації. Як і відповідає шаблону MVC, бізнес-логіка в контролері мінімальна. Для захисту шляхів REST API використовується захисний middleware (Passport Guards) – тобто додаткова перевірка HTTP-запиту перед обробкою на контролері. В ньому перевіряється, чи є користувач авторизованим (JWTGuard), або навпаки — обов’язково неавторизованим (LocalGuard).

```
import { Controller, Post, UseGuards, Request, Res } from '@nestjs/common';
import { Response } from 'express';
import { AuthService } from './auth.service';
import { LocalGuard } from '../../guards/local.guard';
import { JwtGuard } from '../../guards/jwt.guard';

@Controller('/auth')
export class AuthController {
  constructor(private authService: AuthService) {}

  @UseGuards(LocalGuard)
  @Post('/login')
  login(@Request() req: any) {
    return this.authService.login(req.user);
  }

  @UseGuards(JwtGuard)
  @Post('/logout')
  logout(@Request() req: any, @Res() res: Response) {
    req.logout();
    res.redirect('/');
  }

  @UseGuards(JwtGuard)
  @Post('/refresh')
  async refresh(@Request() req) {
    return this.authService.login(req.user);
  }
}
```

Рис. 3.13. Структура контролера авторизації auth.controller.ts.

У сервісах зосереджена основна бізнес-логіка програми. Наприклад, у сервісі аутентифікації (рис. 3.14) реалізоване шифрування паролю: спершу генерується криптографічна сіль (випадкова рядкова послідовність для підвищення безпеки та унеможливлення зламу паролю з використанням “райдужних таблиць”) та використовується KDF (англ. Key Derivation Function — функція генерування

ключа, використовується разом з іншими секретними ключами, зокрема, криптографічною сіллю, щоби значно збільшити час генерування зашифрованого паролю і, відповідно, його підбір). Також реалізоване створення JWT-токену та аутентифікація користувача.

```
@Injectable()
export class AuthService {
  constructor(
    private userService: UserService,
    private jwtService: JwtService,
  ) {}

  private static cypherPassword(password: string, salt: string): string {
    return pbkdf2Sync(password, salt, 10000, 64, 'sha256').toString('hex');
  }

  /**
   * Hash password using KDF with sha256 and random salt
   * @param password password to hash
   * @return array pair of hashed password and salt
   */
  hashPassword(password: string): [string, string] {
    const salt = randomBytes(16).toString('hex');
    return [AuthService.cypherPassword(password, salt), salt];
  }

  /**
   * Authenticate user by its' e-mail and password
   * @param email email of user
   * @param password password of user
   * @return number id of authenticated user
   */
  async authenticateUser({ email, password }: AuthDto): Promise<User> {
    const user = await this.userService.findOneByEmail(email);
    if (!user) {
      throw new UserNotFoundException();
    }
    const hashedPassword = AuthService.cypherPassword(
      password,
      user.password_salt,
    );

    if (hashedPassword !== user.password_hash) {
      throw new InvalidPasswordException();
    }
    return user;
  }

  login(user: User) {
    const payload = { id: user.id };
    return {
      accessToken: this.jwtService.sign(payload),
    };
  }
}
```

Рис. 3.14. Структура сервісу авторизації auth.service.ts.

3.4. Розробка API

У таблиці 3.2 описано API (англ. Application Program Interface) — інтерфейс для взаємодії клієнтської та серверної частини сервісу на рівні бізнес-логіки.

Таблиця 3.2. Опис прикладного програмного інтерфейсу

Шлях	Захищений від неавторизованих користувачів	Короткий опис
POST /auth/login	Ні	Зайти в систему
POST /auth/logout	Захищений	Вийти з системи
POST /auth/refresh	Захищений	Видати користувачу новий JWT-токен після закінчення дії старого
POST /signup	Ні	Зареєструватись
POST/signup?token=:token	Ні	Зареєструватись при запрошенні
GET /verify/:token	Ні	Верифікація нового користувача за токеном
POST /enterprise	Захищений	Додати підприємство
POST /user_enterprise	Захищений	Додати співробітника на підприємство
GET /user_enterprise/invitation/accept/:token	Ні	Перейти за посиланням верифікації додавання користувача на підприємство
POST /template	Захищений	Додати шаблон (з усіма задачами)
GET /template	Захищений	Отримати шаблон (з усіма задачами)
POST /checklist	Захищений	Назначити контрольний список користувачу
GET /checklist?id=:id	Захищений	Отримати контрольний список з усіма даними

Продовження таблиці 3.2

Шлях	Захищений від неавторизованих користувачів	Короткий опис
GET /checklist/all?nickn=&page=&enterprise=	Захищений	Отримати список усіх задач співробітника
PUT /checklist/:id	Захищений	Оновити стан задачі (позначити пункт як виконаний чи заповнити поля)
GET /checklist?id=	Захищений	Отримати контрольний список
POST /message	Захищений	Надіслати повідомлення
POST /report	Захищений	Згенерувати звіт за параметрами в тілі запиту
POST /roles/add/:role	Захищений	Адміністратор назначає роль користувачу
POST /roles/remove	Захищений	Адміністратор змінює роль користувача до звичайної

3.5. Розробка клієнтської частини застосунку

Клієнтська частина сервісу складається з односторінкового застосунку на React. В теці public розміщені статичні файли, які доступні публічно і використовуються React для відображення застосунку: такі як іконка сайту (англ. Favicon) чи базова HTML-сторінка, в яку відображаються React-компоненти. Вище у файловій структурі рівня сирцевого коду, як і в серверній частині, знаходиться файл зі змінними оточення, package.json і package-lock.json, Dockerfile та налаштування для esint і prettier (рис 3.15).

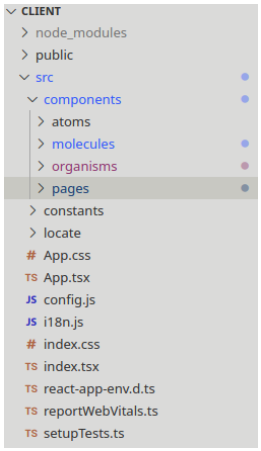


Рис. 3.15. Структура сирцевого коду клієнтської частини.

Для локалізації використовується модуль react-i18next. Конфігурація модуля розміщується у файлі i18n.js, де вказуються мови локалізації та мова за умовчанням. Сервіс розроблено з локалізацією українською та англійською мовами, стандартною є українська. Файли з локалізацією розміщуються в теці locate — вони є JSON-файлами з ключами-назвами об’єктів локалізації та значеннями перекладу елемента інтерфейсу різними мовами. Для використання локалізації (рис. 3.16) та зміни мови використовуються React-хуки для управління станом React-застосунку.

```
export function TopAppBar() {
  const { t, i18n } = useTranslation();

  return (
    <Navbar expand="lg" id="app-bar">
      <Navbar.Brand href="/">TrainTo</Navbar.Brand>

      <Navbar.Toggle aria-controls="basic-navbar-nav" />
      <Navbar.Collapse id="basic-navbar-nav">
        <Nav className="ml-auto">
          <Nav.Link href="/dashboard">{t('dashboard')}</Nav.Link>
          <Nav.Link href="#link">{t('myEnterprise')}</Nav.Link>
          <Nav.Link href="#link">
            <Notification />
          </Nav.Link>
          <Dropdown />
          <Nav.Item
            onClick={() => {
              switch (i18n.language) {
                case 'uk':
                  i18n.changeLanguage('en');
                  break;
                case 'en':
                  i18n.changeLanguage('uk');
                  break;
                default:
                  i18n.changeLanguage('uk');
              }
            }}
          >
            {t('switchLang')}
          </Nav.Item>
        </Nav>
      </Navbar.Collapse>
    </Navbar>
  );
}
```

Рис. 3.16. Локалізація та зміна мови в компоненті верхнього меню.

При розробці використано бібліотеку Bootstrap та підхід atomic-дизайну. Як видно зі структури проєкту, файли сирцевого коду розділені на теки з компонентами в них різного рівня ієрархії, як наприклад в “атомах” міститься тільки елементарна бізнес-логіка (див. розділ 2.4.3).

Для взаємодії з сервером використовується axios — модуль асинхронного HTTP-клієнта для середовища NodeJS чи браузера. Він підтримує multipart-запити (з мультимедійними даними), конфігуровний та легкий у налаштуванні. Для навігації використовується модуль react-router-dom, який дозволяє оголошувати шляхи навігації, прив’язувати їх до відображення окремих компонентів, обробляти параметри в URL чи рядки запиту (англ. Query String).

У таблиці 3.3 наведені шляхи для навігації сторінками у клієнтській частині застосунку. В захищених від неавторизованих користувачів шляхах використовується рядок запиту *?enterpriseId=*, за значенням якого ідентифікується підприємство, коли необхідно надіслати запит до сервера.

Таблиця 3.3. Шляхи навігації у клієнтській частині застосунку

Шлях	Опис
/	Головна сторінка
/login	Увійти
/signup	Зареєструватись
/verify	Сторінка, на яку після реєстрації перенаправляється користувач з проханням перевірити електронну пошту для верифікації акаунту
/verified	Сторінка, на яку перенаправляється користувач, коли верифікація закінчена
/enterprise	Сторінка підприємства
/enterprise/new	Створити підприємство
/dashboard	Робоча панель співробітника із задачами та сповіщеннями
/tasks	Всі задачі співробітника (у формі контрольних списків)

Продовження таблиці 3.3

Шлях	Опис
/task?id=	Окрема задача
/notifications	Всі сповіщення
/message/new	Надіслати повідомлення з питанням
/checklist?id=	Сторінка задачі (у формі контрольного списку)
/add_worker	Запросити співробітника на підприємство
/template/new	Створити шаблон для контрольних списків
/report/new	Сторінка генерування звітності

3.6. Розгортання застосунку

Для розгортання застосунку був використаний сервіс Amazon EC2. При його використанні надається доступ до віртуальної машини з можливістю її налаштування (рис. 3.17). Для розгортання застосунку було взято віртуальну машину з установленою операційною системою Ubuntu 20.04 та налаштованим доступом підключення по SSH та доступ HTTP через глобальну IP-адресу чи доменне ім’я DNS. Через підключення по SSH (з використанням приватного pem-ключа, який генерується на стороні сервісу AWS) відбувається налаштування застосунку для розгортання. Для клієнтської та серверної частин застосунку були створені Dockerfile — з їх виконанням створюються контейнери, які можна запустити не тільки на пристрої розробника. Для поєднання зображень Docker між собою було використано docker-compose — засіб налаштування контейнерів для спільної роботи, в ньому було і налаштовано контейнер для бази даних. Команди Docker для запуску контейнерів з docker-compose виконуються за допомогою bash-скрипта. Приватний git-репозиторій з розробленим застосунком склоновано на віртуальну машину та запущено з використанням контейнерів. У Dockerfile для серверної частини застосунку та бази даних вказано доступ тільки в локальній мережі, так як API не є публічним.

Доступ до клієнтської частини застосунку захищений TLS (англ. Transport Level Security) — це надає змогу користувачу передавати свої дані на сервер захищено.

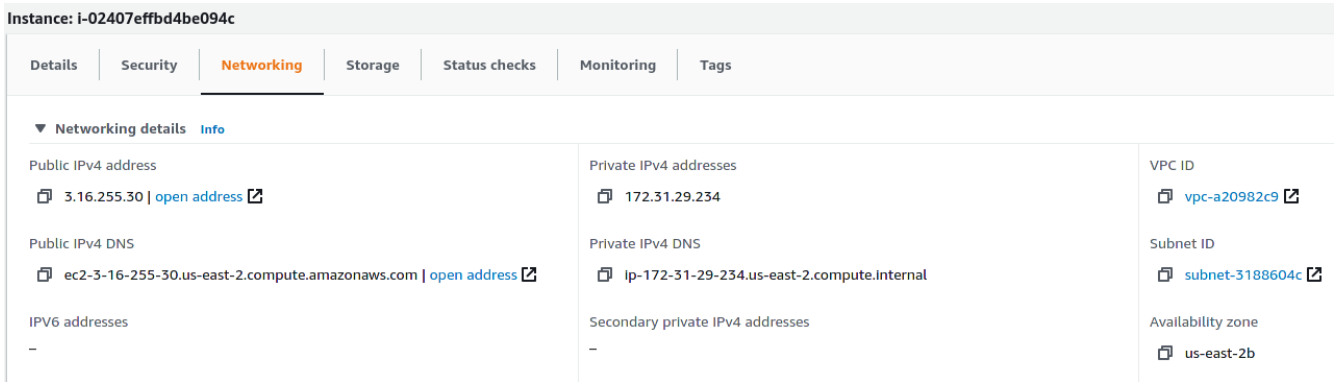


Рис. 3.17. Запущена віртуальна машина Amazon EC2.

ВИСНОВКИ ДО РОЗДІЛУ 3

В розділі була описана розробка компонентів сервісу, а саме бази даних, серверної та клієнтської частини сервісу. Вибрані технології, описані в другому розділі дипломної роботи, були описані детальніше та показано, як їх використано при розробці сервісу.

В клієнтській частині сервісу реалізований SPA-застосунок з адаптивним вебдизайном, локалізацією англійською та українською мовами і використанням підходу atomic-дизайну при розробці. Був розроблений захищений API на основі MVC-фреймворку Nest.js та спроєктована база даних для роботи сервісу. Використання міграцій бази даних забезпечують її переносимість та допомагають уникнути небажаних змін її структури. З використанням Docker сервіс було розгорнуто на Amazon EC2, що відповідає підходу SaaS — підприємство має можливість використовувати застосунок “на вимогу”, отримуючи до нього захищений доступ.

РОЗДІЛ 4
ТЕСТУВАННЯ РОЗРОБЛЕНОГО СЕРВІСУ

В розділі описаний функціонал сервісу з точки зору користувача, оглянуто його можливості та базовий функціонал. Для розгортання застосунку необхідно склонувати git-репозиторій та створити .env-файл (рис. 4.1) — він використовується для зберігання змінних середовища, які зберігають інформацію, що не може включатись у сирцевий код. Для запуску контейнерів необхідно запустити bash-скрипт — буде автоматично виконане установлення залежностей, запуск бази даних та виконані її міграції зі створенням у ній необхідних таблиць і відношень.

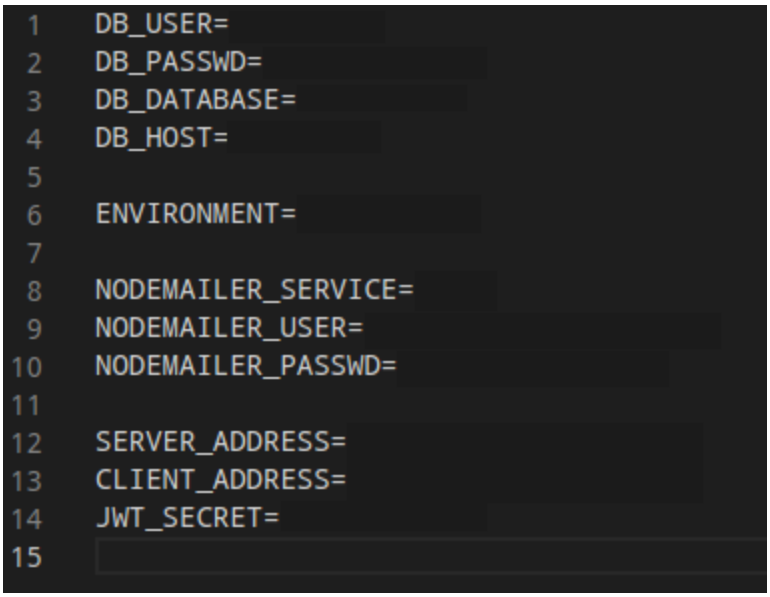


Рис. 4.1. Файл .env із прихованою особистою інформацією.

Для використання сервісу необхідно перейти за посиланням *https://my-application-diploma.ec2-website.eu-north-1.amazonaws.com* — користувацьке доменне ім’я не використовувалось унаслідок високої ціни.

При переході користувачу показується головна сторінка з можливістю увійти чи зареєструватись. На рис. 4.2. показано локалізовану головну сторінку з реалізованим адаптивним дизайном. Сторінка не відрізняється для користувачів, що увійшли чи не увійшли в систему, і використовується для загального опису

застосунку — основний функціонал розташований на сторінках користувача та “Моє підприємство”.

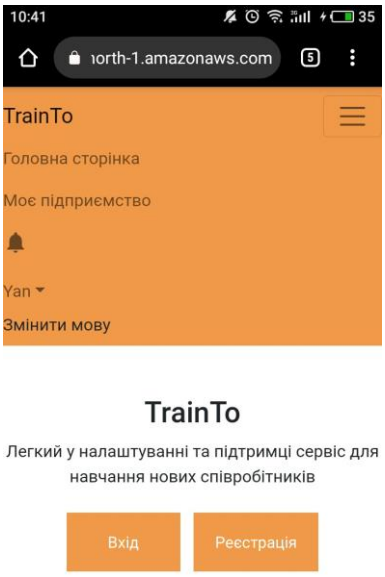


Рис. 4.2. Головна сторінка на мобільному пристрої.

Користувач може увійти за адресою електронної пошти та паролем або зареєструватись. Ці сторінки мало відрізняються за своїм оформленням, в потрібних полях вводу є валідація вхідних даних, як на рис. 4.3.

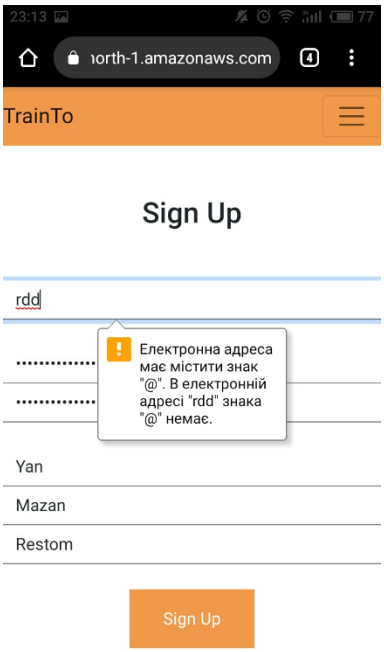


Рис. 4.3. Валідація вхідних даних у формі реєстрації.

На сторінці підприємства (рис. 4.4) користувач бачить свої робочі завдання, сповіщення та повідомлення. Менеджеру чи адміністратору додатково доступна функція перегляду шаблонів робочих задач, оновлення щодо виконання задач іншими співробітниками та генерування звітності.

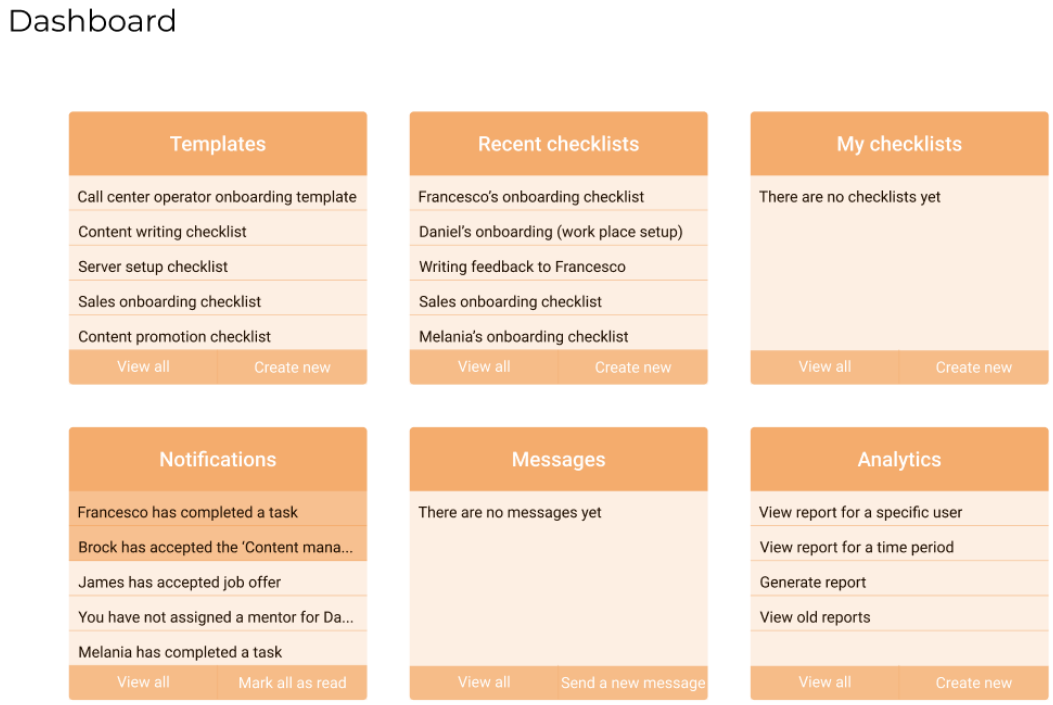


Рис. 4.4. Головна робоча панель менеджера та адміністратора.

Менеджер чи адміністратор має можливість створити шаблон (рис. 4.5): дати йому назву, додати задачі та описати їх. Перетягнувши поле зі списку справа, він може налаштувати його, наприклад, додати заголовок і вміст у текстове поле, і зберегти. Відповідно, цей шаблон можна призначати співробітникам як контрольний список. Його вигляд майже не відрізняється від шаблону, окрім відсутності меню редагування справа та кнопок з позначенням задачі виконаною, переходу до наступної і кнопки зв'язку з ментором для запитань (рис. 4.6).

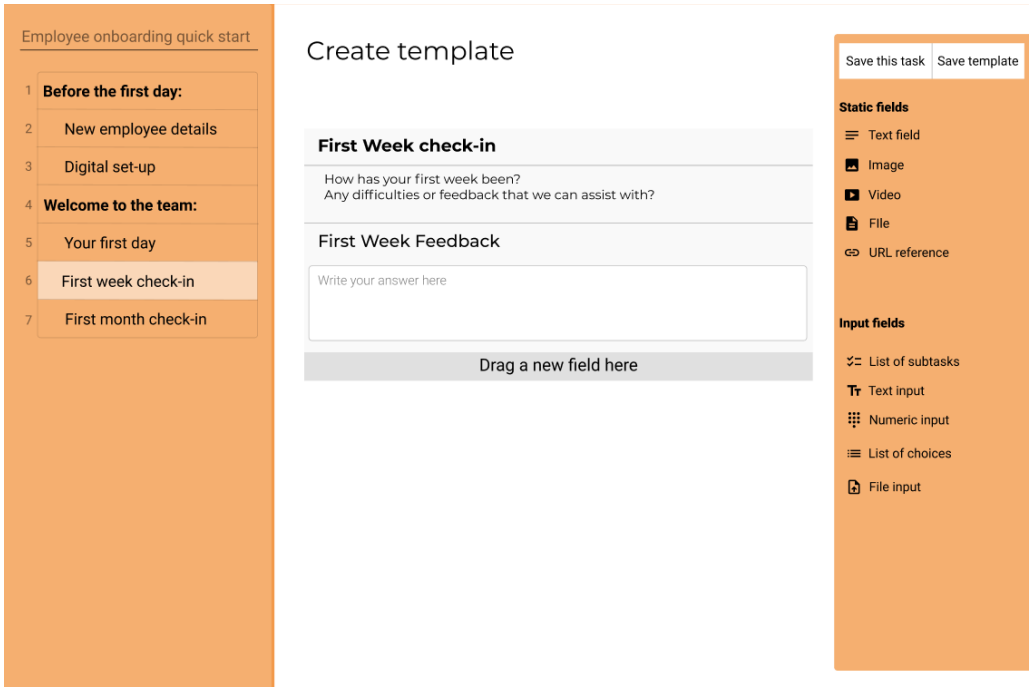


Рис. 4.5. Сторінка створення шаблону.

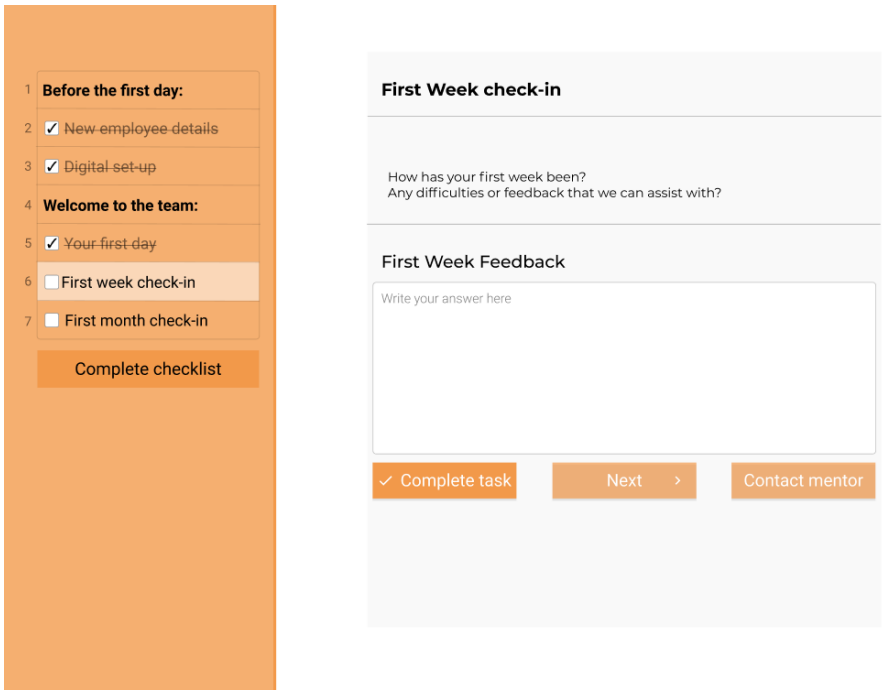


Рис. 4.6. Заповнення контрольного списку.

Також є можливість запросити нового співробітника (рис. 4.7). Менеджер чи адміністратор створює запрошення, після цього новому користувачу воно надсилається електронною поштою (рис. 4.8).

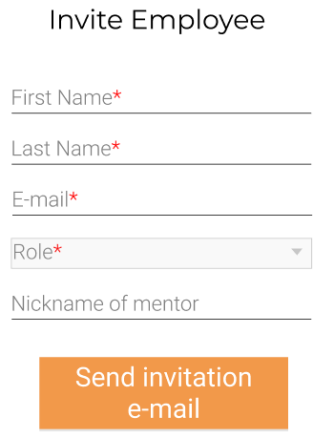


Рис. 4.7. Запрошення нового співробітника.

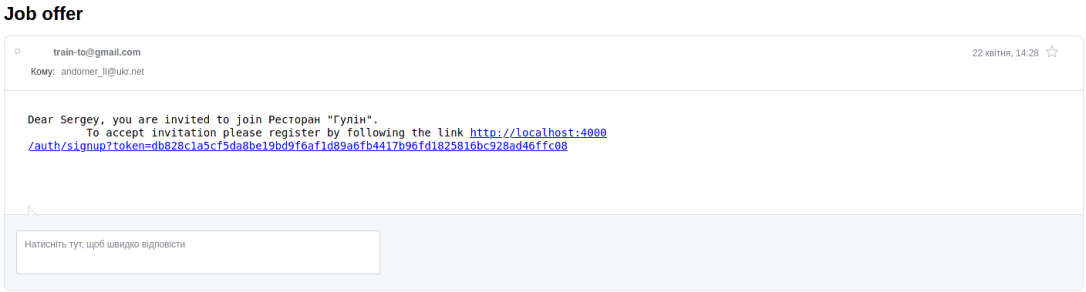


Рис. 4.8. Запрошення незареєстрованому співробітнику. Реєстрація відбувається за токеном, за яким ідентифікується запрошення.

Співробітник може переглядати свої навчальні задачі на спеціальній сторінці (рис. 4.9). На ній показані задачі в процесі та виконані раніше, кількість виконаних пунктів контрольного списку і можливість перейти до деталей.

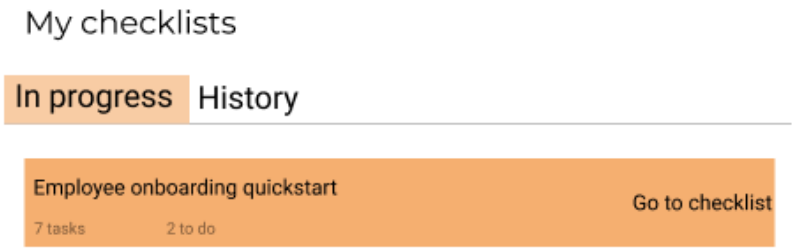


Рис. 4.9. Сторінка з контрольними списками користувача.

Також у нього є можливість надіслати повідомлення ментору, що показано на

рис. 4.10. Ментору може як прийти сповіщення електронною поштою, так і звичайне повідомлення в чаті. Даний функціонал не є основним та доповнює інші елементи розробленого сервісу.

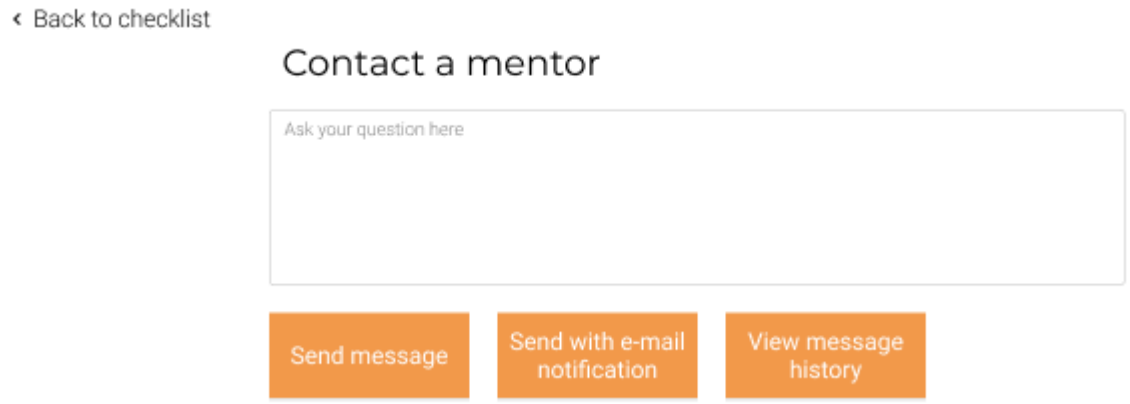


Рис. 4.10. Надсилання повідомлення ментору.

ВИСНОВКИ ДО РОЗДІЛУ 4

В розділі було протестовано роботу розробленого SaaS-застосунку. Продемонстровано роботу базового функціоналу, а саме, реєстрації користувача, його верифікацію електронною поштою, запрошення нових співробітників, створення шаблонів контрольних списків і можливості їх заповнення та інформування менеджера про це. Сервіс розгорнутий на Amazon EC2, показана послідовність дій, як його можна розгорнути на іншому пристрої та забезпечити приватність даних, необхідних для запуску програми.

ВИСНОВКИ

В дипломній роботі була оглянута тема адаптації нового співробітника на підприємстві та розглянуті наявні програмні рішення для полегшення цього процесу. Виявлено, що вони можуть бути орієнтованими на ринок виключно конкретної країни, мати застарілий інтерфейс користувача з відсутністю локалізації чи містити надто деталізований функціонал, через що може виникнути ситуація, в якій підприємство передплачує за те, що йому не потрібно.

Запропоноване рішення є SaaS-застосунком, що повністю відповідає сучасним тенденціям розробки складних сервісів. Тобто розроблений сервіс є застосунком “на вимогу” — користувач через інтерфейс вебзастосунка використовує програмне забезпечення, яке розміщується у хмарному сервісі.

Окреслений мінімальний функціонал може використовуватись для навчання нових співробітників на підприємстві, а використані для розробки технології дозволяють підтримувати і доповнювати застосунок у довгостроковій перспективі. Так як застосунок розгорнуто у хмарному сервісі AWS, то він може бути масштабований для підтримки більшої кількості користувачів та конкурувати з існуючими рішеннями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. T. Bauer, B. Erdogan Organizational socialization: The effective onboarding of new employees // APA Handbook of I/O Psychology, Volume II. — 2011. — P. 51-64.
2. Forbes. How To Get Employee Onboarding Right. — 2015. — [Електронний ресурс] Режим доступу https://www.forbes.com/sites/theyec/2015/05/29/how-to-get-employee-onboarding-right/?sh=9d2cff7407bd
3. SHRM. Don’t Underestimate the Importance of Good Onboarding. — 2017. —[Електронний ресурс] Режим доступу https://www.shrm.org/resourcesandtools/hr-topics/talent-acquisition/pages/dont-underestimate-the-importance-of-effective-onboarding.aspx
4. Microsoft Azure. What is SaaS? Software as a Service [Електронний ресурс] — Режим доступу https://azure.microsoft.com/en-us/overview/what-is-saas/
5. Process Street. [Електронний ресурс] — Режим доступу https://app.process.st
6. Capterra. Applicant Stack. Reviews and Pricing. [Електронний ресурс] — Режим доступу https://www.capterra.com/p/99349/ApplicantStack/
7. ApplicantStack. API Integration Guide [Електронний ресурс] — Режим доступу https://help.applicantstack.com/hc/en-us/articles/115000786273-API-Integration-Guide
8. Applicant Tracking System. HR Software. Офіційний сайт компанії ApplicantStack [Електронний ресурс] — Режим доступу https://applicantstack.com/
9. Rippling Demo. Офіційний сайт компанії Rippling [Електронний ресурс] — Режим доступу https://www.rippling.com/tour-video
10. Greenhouse Onboarding Software. Офіційний сайт [Електронний ресурс]

— Режим доступу https://www.greenhouse.io/onboarding

11. Microsoft Application Architecture Guide, 2nd Edition. Chapter 5: Layered Application Guidelines. — 2010. [Електронний ресурс] Режим доступу https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658109(v=pandp.10)
12. Ian Davis. What Are The Benefits of MVC? // Internet Alchemy, the blog of Ian Davis. — 2008. — [Електронний ресурс] Режим доступу https://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc/
13. Kharenko A. Monolithic vs. Microservices Architecture // Microservices Practitioner Articles. — 2015. — [Електронний ресурс] Режим доступу https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59
14. Codd. E. A relational model of data for large shared data banks. // Communications of the ACM. — 1970.
15. What is NoSQL? NoSQL Datadases Explained. Документація MongoDB. — [Електронний ресурс] Режим доступу https://www.mongodb.com/nosql-explained
16. Chamith M. Session Management in Nodejs Using Redis as Session Store. // Medium. — 2020. — [Електронний ресурс] Режим доступу https://medium.com/swlh/session-management-in-nodejs-using-redis-as-session-store-64186112aa9
17. ORM (Object-Relational Mapping). // Национальная библиотека им. Н. Э. Баумана. — 2019. — [Електронний ресурс] Режим доступу https://ru.bmstu.wiki/ORM_(Object-Relational_Mapping)
18. What is Amazon S3? Amazon Simple Storage Service. Документація AWS. — 2021. — [Електронний ресурс] Режим доступу https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
19. SPA (Single-page application). MDN Web Docs Glossary. — 2021. —

[Електронний ресурс] Режим доступу https://developer.mozilla.org/en-US/docs/Glossary/SPA

20. Birdeau L. Delivery of data and formatting information to allow client-side manipulation. / L. Birdeau, C. Yeh, M. Peachey, K. Hakman // US8136109B1. — 2002.
21. B. Omelchenko. SPA vs MPA: Which One is Better For You? Офіційний сайт компанії Yojji. — 2020. — [Електронний ресурс] Режим доступу https://yojji.io/blog/spa-vs-mpa
22. J. Miller. Rendering on the Web / J. Miller, A. Osmani // Google Developers Blog. — 2019. — [Електронний ресурс] Режим доступу https://developers.google.com/web/updates/2019/02/rendering-on-the-web?hl=en
23. Начало работы с React — Изучение веб-разработки. MDN Web Docs Glossary. — 2021. — [Електронний ресурс] Режим доступу https://developer.mozilla.org/ru/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started
24. React — JavaScript-бібліотека для створення користувацьких інтерфейсів. Офіційна документація React. — 2021. — [Електронний ресурс] Режим доступу https://uk.reactjs.org/
25. React+TypeScript Cheatsheets. — 2019. — [Електронний ресурс] Режим доступу https://github.com/typescript-cheatsheets/react
26. AngularJS: Branding Guidelines for Angular and AngularJS. — 2017. — [Електронний ресурс] Режим доступу http://blog.angularjs.org/2017/01/branding-guidelines-for-angular-and.html
27. Angular — What is Angular? Офіційна документація Angular. [Електронний ресурс] Режим доступу https://angular.io/guide/what-is-angular
28. Introduction. Vue.js. [Електронний ресурс] Режим доступу

https://v3.vuejs.org/guide/introduction.html#getting-started
29. Comparison with other Frameworks. Vue.js. [Електронний ресурс] Режим доступу https://vuejs.org/v2/guide/comparison.html
30. Language Tour. Документація Dart. [Електронний ресурс] Режим доступу https://dart.dev/guides/language/language-tour
31. Introduction. Bootstrap v5.0. [Електронний ресурс] Режим доступу https://getbootstrap.com/docs/5.0/getting-started/introduction/
32. react-bootstrap. NPM. [Електронний ресурс] Режим доступу https://www.npmjs.com/package/react-bootstrap
33. B. Frost. Atomic Design. // Brad Frost Personal Blog. — 2013. — [Електронний ресурс] Режим доступу https://bradfrost.com/blog/post/atomic-web-design/
34. Мэтью С. Обзор Amazon Web Services. — 2017. — [Електронний ресурс] Режим доступу https://d1.awsstatic.com/whitepapers/ru_RU/aws-overview.pdf
35. Cusumano M. Cloud computing and SaaS as new computing platforms // Communications of the ACM. – 2010. – Т. 53. – №. 4. – С. 27-29.
36. Что такое Docker? Документация AWS. [Електронний ресурс] Режим доступу https://aws.amazon.com/ru/docker/
37. Bicheno. S. AWS accounted for a third of $42 billion cloud market in Q1 2021. / S. Bicheno // Telecoms.com. — 2021. — [Електронний ресурс] Режим доступу https://telecoms.com/509588/aws-accounted-for-a-third-of-42-billion-cloud-market-in-q1-2021/
38. Mathew S. Overview of Amazon Web Services. / S. Mathew // AWS Whitepaper. — 2021. — [Електронний ресурс] Режим доступу https://d1.awsstatic.com/whitepapers/aws-overview.pdf
39. Amazon EC2. Документація AWS. [Електронний ресурс] Режим доступу https://aws.amazon.com/ec

ДОДАТОК 1

Сервіс для навчання нових співробітників на підприємстві

Принципова схема – алгоритм роботи програми
ІАЛЦ.467200.004 Д1

Аркушів 1

Київ – 2021

					ІАЛЦ.467200.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Мазан Я. В.			*Принципова схема. Алгоритм роботи програми*	Лит. / Арк. 1 / Аркушів 1
Перев.		Сергієнко А. А.				
Реценз.						НТУУ «КПІ», ФІОТ, ІВ-71
Н. Контр.		Сімоненко В. П.				
Затверд.		Стіренко С. Г.				

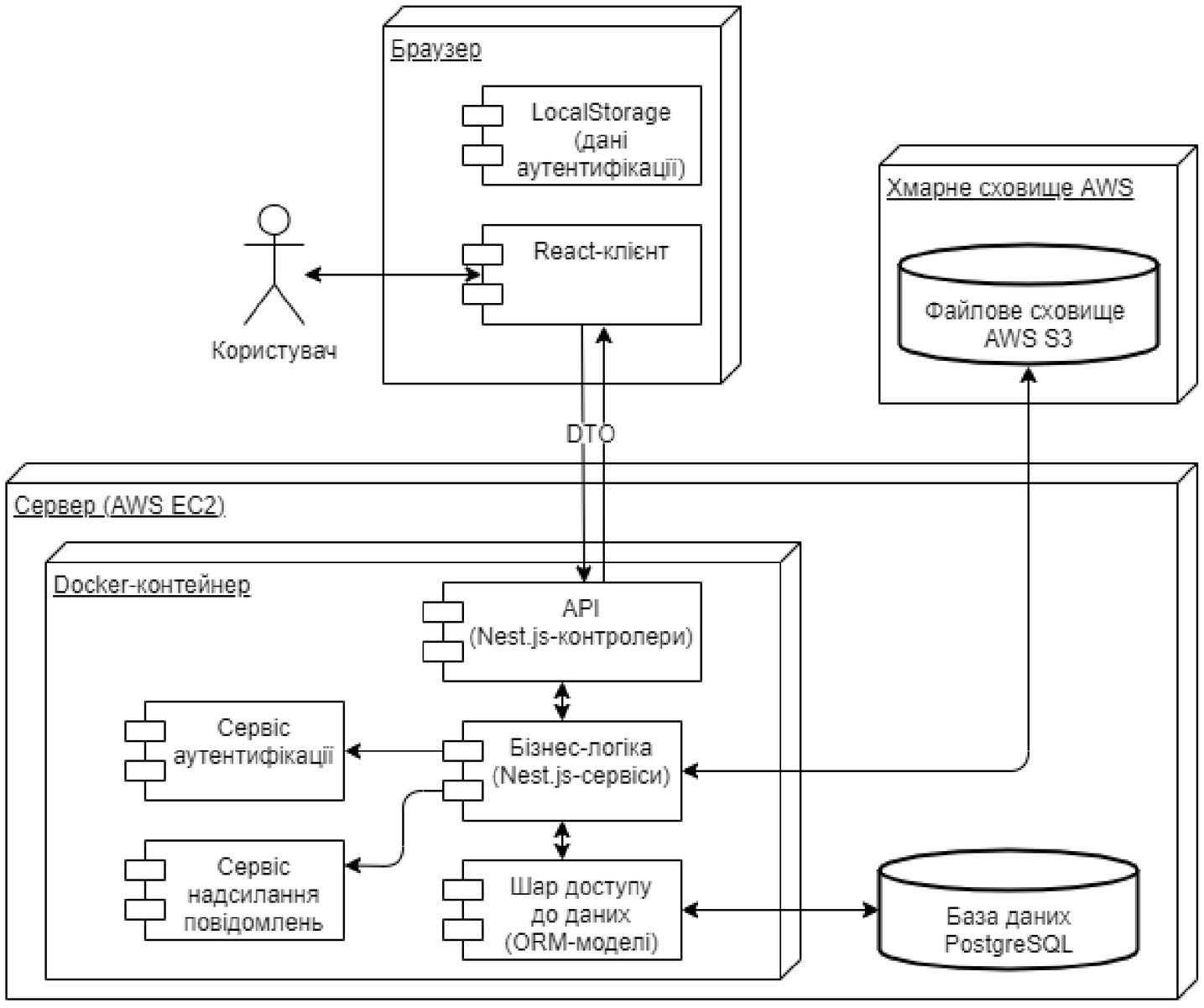
ДОДАТОК 2

Сервіс для навчання нових співробітників на підприємстві

Структурна схема – діаграма розгортання застосунку
ІАЛЦ.467200.005 Д2

Аркушів 1

Київ – 2021



					ІАЛЦ.467200.005 Д2	
Змн.	Арк.	№ докум.	Підпис	Дат		
Розроб.		Мазан Я. В.			Структурна схема. Діаграма розгортання застосунку	Літ. / Маса / Масш.
Перевір.		Сергієнко А. А.				
Т. Контр.						Аркуш / Аркушів
Н. Контр.		Сімоненко В. П.			Дипломна робота	НТУУ «КПІ», ФІОТ, ІВ-71
Затверд.		Стіренко С. Г.				

ДОДАТОК З

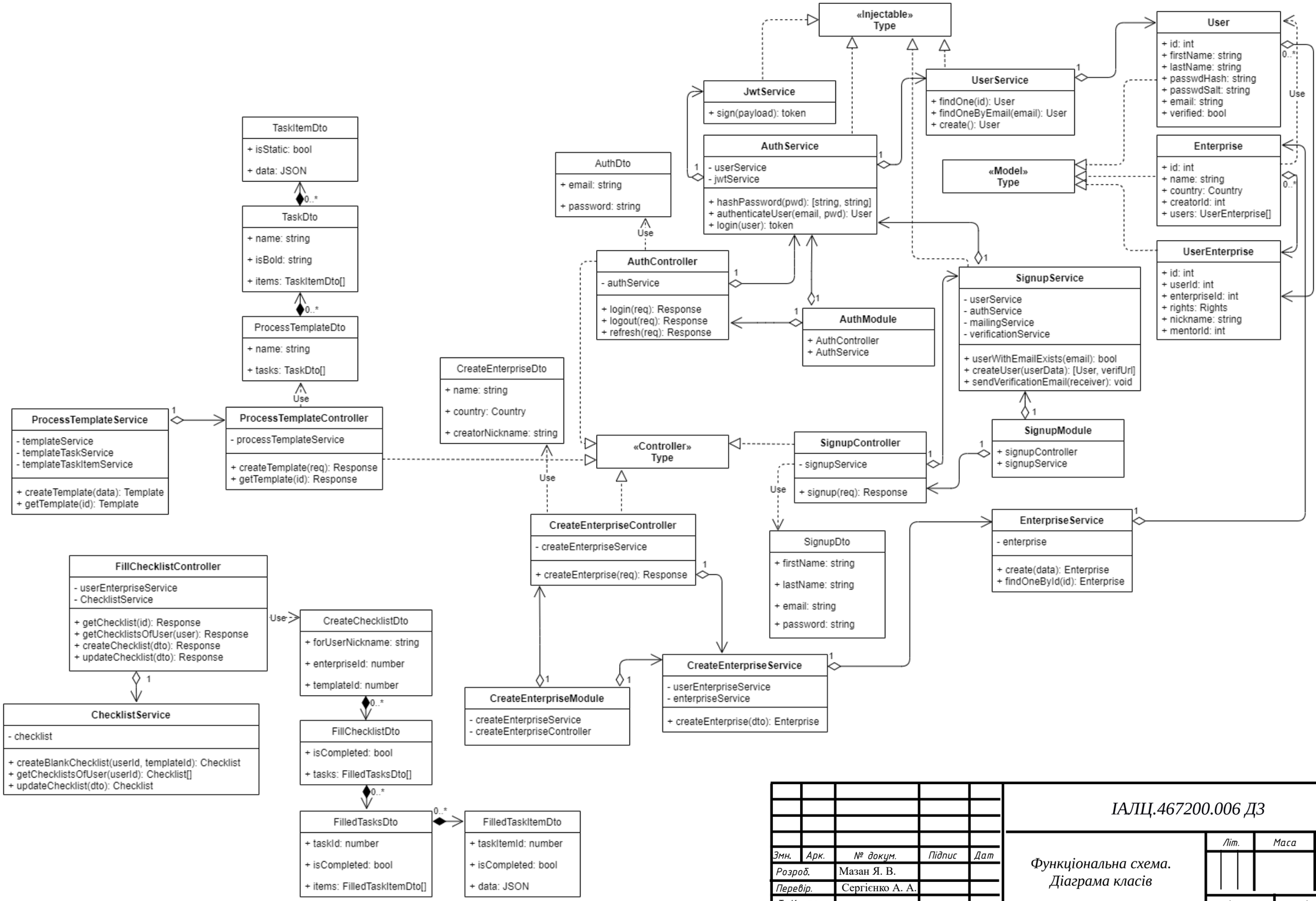
Сервіс для навчання нових співробітників на підприємстві

Функціональна схема – діаграма класів

ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ – 2021



					ІАЛЦ.467200.006 Д3	
Змн.	Арк.	№ докум.	Підпис	Дат		
Розроб.		Мазан Я. В.			Функціональна схема. Діаграма класів	Літ. / Маса / Масш.
Перевір.		Сергієнко А. А.				
Т. Контр.						Аркуш / Аркушів
Н. Контр.		Сімоненко В. П.			Дипломна робота	НТУУ «КПІ», ФІОТ, ІВ-71
Затверд.		Стіренко С. Г.				

ДОДАТОК 4

Сервіс для навчання нових співробітників на підприємстві

Лістинг програми
ІАЛЦ.467200.007 Д4

Аркушів 10

Київ – 2021

auth.service.ts

```
@Injectable()
export class AuthService {
  constructor(
    private userService: UserService,
    private jwtService: JwtService,
  ) {}

  private static cypherPassword(password: string, salt: string): string {
    return pbkdf2Sync(password, salt, 10000, 64, 'sha256').toString('hex');
  }

  /**
   * Hash password using KDF with sha256 and random salt
   * @param password password to hash
   * @return array pair of hashed password and salt
   */
  hashPassword(password: string): [string, string] {
    const salt = randomBytes(16).toString('hex');
    return [AuthService.cypherPassword(password, salt), salt];
  }

  /**
   * Authenticate user by its' e-mail and password
   * @param email email of user
   * @param password password of user
   * @return number id of authenticated user
   */
  async authenticateUser({ email, password }: AuthDto): Promise<User> {
    const user = await this.userService.findOneByEmail(email);
    if (!user) {
      throw new UserNotFoundException();
    }
    const hashedPassword = AuthService.cypherPassword(
      password,
      user.password_salt,
    );

    if (hashedPassword !== user.password_hash) {
      throw new InvalidPasswordException();
    }
    return user;
  }

  login(user: User) {
```

```
    const payload = { id: user.id };
    return {
      accessToken: this.jwtService.sign(payload),
    };
  }
}
```

processTemplate.controller.ts

```
@Controller('/template')
export class ProcessTemplateController {
  constructor(
    private processTemplateService: ProcessTemplateService,
    private userEnterpriseService: UserEnterpriseService,
    private checkAccessService: CheckAccessService,
  ) {}

  @UseGuards(JwtGuard)
  @Post()
  async createTemplate(@Body() body: ProcessTemplateDto, @Request() req) {
    const {
      id: userEnterpriseId,
    } = await this.checkAccessService.mustBeAManagerOrAdmin(
      req.user.id,
      body.enterpriseId,
    );
    return this.processTemplateService.createTemplate(body, userEnterpriseId);
  }

  @UseGuards(JwtGuard)
  @Get()
  async getTemplate(@Query('id') templateId: string) {
    return this.processTemplateService.getTemplateById(templateId);
  }
}
```

processTemplate.service.ts

```
@Injectable()
export class ProcessTemplateService {
  constructor(
    private templateService: TemplateService,
    private templateTaskService: TemplateTaskService,
    private templateTaskItemService: TemplateTaskItemService,
  ) {}

  async createTemplate(
```

```
    data: ProcessTemplateDto,
    userId: number,
  ): Promise<Template> {
    const template = await this.templateService.create(userId, data.name);
    for (const [i, taskDto] of data.tasks.entries()) {
      const task = await this.templateTaskService.create(
        template.id,
        i,
        taskDto.name,
        taskDto.isBold,
      );
      for (const [j, taskItemDto] of taskDto.items.entries()) {
        await this.templateTaskItemService.createWithField(
          task.id,
          j,
          true,
          taskItemDto.isStatic,
          taskItemDto.data,
        );
      }
    }

    return this.templateService.findOneById(template.id);
  }

  async getTemplateById(queryParamTemplateId: string): Promise<Template> {
    return this.templateService.findOneById(parseInt(queryParamTemplateId));
  }
}
```

mailing.service.ts

```
@Injectable()
export class MailingService {
  private transporter;

  constructor() {
    this.transporter = createTransport({
      service: nodemailer.NODEMAILER_SERVICE,
      auth: {
        user: nodemailer.NODEMAILER_USER,
        pass: nodemailer.NODEMAILER_PASSWD,
      },
    });
  }
```

```
  async sendMail(to: string, subject: string, text: string, html?: string) {
    const mailOptions = {
      from: nodemailer.NODEMAILER_USER,
      to,
      subject,
      text,
      html,
    };
    await this.transporter.sendMail(mailOptions, (error, info) => {
      if (error) {
        throw new EmailNotSentException('Could not send email');
      }
    });
  }
}
```

checklist.service.ts

```
@Injectable()
export class ChecklistService {
  constructor(
    @InjectModel(Checklist)
    private checklist: typeof Checklist,
    private checklistTaskService: ChecklistTaskService,
    private checklistTaskItemService: ChecklistTaskItemService,
    private templateService: TemplateService,
  ) {}

  create(templateId: number, userId: number): Promise<Checklist> {
    // @ts-ignore
    return this.checklist.create({
      template_id: templateId,
      user_id: userId,
    });
  }

  private readonly includedChecklistModels: Includeable[] = [
    {
      model: Template,
      attributes: ['id', 'name'],
    },
    {
      model: ChecklistTask,
      attributes: [
        'id',
        ['is_completed', 'isCompleted'],
```

```
      ['created_at', 'createdAt'],
    ],
    order: [['order', 'ASC']],
    include: [
     {
      model: ChecklistTaskItem,
      attributes: [
       'id',
       ['is_completed', 'isCompleted'],
       ['created_at', 'createdAt'],
      ],
      include: [
       {
        model: TemplateTaskItem,
        include: [
         {
          model: StaticField,
         },
         {
          model: DynamicField,
         },
        ],
       },
       {
        model: FilledDynamicField,
       },
      ],
     },
     {
      model: TemplateTask,
      attributes: ['id', 'name', ['is_bold', 'isBold'], 'order'],
     },
    ],
   },
  ];

  findOneById(id: number) {
   return this.checklist.findOne({
    where: {
     id,
     deleted_at: { [Op.is]: null },
    },
    include: this.includedChecklistModels,
   });
  }
```

```
findAllChecklistsOfUser(
  userId: number,
  pageNumber: number,
): Promise<Checklist[]> {
  return this.checklist.findAll({
    where: { user_id: userId },
    offset: (pageNumber - 1) * CHECKLIST_PAGE_SIZE,
    limit: CHECKLIST_PAGE_SIZE,
    include: this.includedChecklistModels,
  });
}

async createWithTasksAndTaskItems(
  templateId: number,
  userId: number,
): Promise<Checklist> {
  const templateWithTasks = await this.templateService.findOneById(
    templateId,
  );
  const createdChecklist = await this.create(templateId, userId);
  for (const templateTask of templateWithTasks.tasks) {
    const createdChecklistTask = await this.checklistTaskService.create(
      createdChecklist.id,
      templateTask.id,
    );
    for (const templateTaskItem of templateTask.taskItems) {
      await this.checklistTaskItemService.create(
        createdChecklistTask.id,
        templateTaskItem.id,
      );
    }
  }
  return this.findOneById(createdChecklist.id);
}

async getRelatedEnterprise(checklistId: number): Promise<Enterprise> {
  const checklist = await this.checklist.findOne({
    where: { id: checklistId, deleted_at: { [Op.is]: null } },
    include: [
      {
        model: UserEnterprise,
        include: [
          {
            model: Enterprise,
```

```
          },
        ],
      },
    ],
  });
  return checklist.assignedUser.enterprise;
 }

 async updateAllTasks(
  data: FillChecklistDto,
  checklistId: number,
 ): Promise<Checklist> {
  await this.checklist.update(
   { is_completed: data.isCompleted },
   { where: { id: checklistId } },
  );

  for (const task of data.tasks) {
   await this.checklistTaskService.updateById(task.taskId, {
    is_completed: task.isCompleted,
   });
   for (const taskItem of task.items) {
    await this.checklistTaskItemService.updateByIdWithField(
     taskItem.taskItemId,
     taskItem.isCompleted,
     (taskItem.field as unknown) as Record<string, unknown>,
    );
   }
  }
  // return updated checklist
  return this.findOneById(checklistId);
 }
}
ROM node:14
WORKDIR /usr/src/app-server

COPY package*.json ./

RUN npm ci --only=production

# Bundle app source
COPY . .

EXPOSE 3000
CMD [ "npm", “run”, "start:prod" ]
```

startup.sh

```
#!/usr/bin/env sh

docker build --no-cache -t yan/nest-server .
docker run -p 3000:3000 -d yan/nest-server
```

Index.tsx:

```
export function Index() {
  const { t } = useTranslation();

  return (
    <>
      <PageMain id="main">
        <Title>{t('serviceName')}</Title>
        <Paragraph className="text-center">{t('basicDescription')}</Paragraph>
        <div className="text-center">
          <LinkButton title="Log In" clickLink="/login" />
          <LinkButton title="Sign Up" clickLink="/signup" />
        </div>
      </PageMain>
      <dl className="description-list" id="descr">
        {t('serviceDescription', { returnObjects: true, joinArrays: '\n' })
          .split('\n')
          .map((str, i) => (
            <dt key={i} className="description-elem">
              {str}
            </dt>
          ))}
      </dl>
    </>
  );
}
```

TopAppBar.tsx

```
import './topAppBar.style.css';

export function TopAppBar() {
  const { t, i18n } = useTranslation();

  return (
    <Navbar expand="lg" id="app-bar">
      <Navbar.Brand href="/">TrainTo</Navbar.Brand>

      <Navbar.Toggle aria-controls="basic-navbar-nav" />
```

```
      <Navbar.Collapse id="basic-navbar-nav">
        <Nav className="ml-auto">
          <Nav.Link href="/dashboard">{t('dashboard')}</Nav.Link>
          <Nav.Link href="#link">{t('myEnterprise')}</Nav.Link>
          <Nav.Link href="#link">
            <Notification />
          </Nav.Link>
          <Dropdown />
          <Nav.Item
            onClick={() => {
              switch (i18n.language) {
                case 'uk':
                  i18n.changeLanguage('en');
                  break;
                case 'en':
                  i18n.changeLanguage('uk');
                  break;
                default:
                  i18n.changeLanguage('uk');
              }
            }}
          >
            {t('switchLang')}
          </Nav.Item>
        </Nav>
      </Navbar.Collapse>
    </Navbar>
  );
}
```