МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: ТЕХНОЛОГІЇ РОЗРОБКИ ВЕБДОДАТКІВ: СУЧАСНІ ПІДХОДИ ТА МЕТОДОЛОГІЇ

Здобувача вищої освіти другого (магістерського) рівня,
ІІ курсу, групи КМІ-24-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»

_____ Підька Андрія Сергійовича
(підпис) (прізвище, ім'я, по батькові)

Науковий керівник _____ Гладченко Оксана Вікторівна, канд. пед. наук, доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми _____ Омельчук Антон Анатолійович, канд. техн. наук
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри протокол від «02» січня 2026 № 10

Завідувач кафедри _____ Філоненко Михайло Миколайович, канд. фіз.-мат. наук, доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2026

Реєстрація № 16 21.01.2026 _____
(дата) (підпис відповідальної особи)

Результати перевірки до захисту 02.01.2026 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 4 (добре) 09.02.2026 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ О.С. Дюбанов
(підпис)

Члени комісії:

_____ А.Ю. Горбовий
(підпис)

_____ М.М. Філоненко
(підпис)

_____ І.Д. Погорєловська
(підпис)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст кваліфікаційної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	**від 0 до 30**	26		09.02.2026 Дюбанов О.С. Горбовий А.Ю. Філоненко М.М. Погорєловська І.Д. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	**від 0 до 10**	8		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	**від 0 до 5**	4		
4. Рецензент	**від 0 до 5**	4		
РАЗОМ БАЛІВ	**від 0 до 50**	42		

Оцінювання кваліфікаційної роботи науковим керівником – 44 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалоюЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
86	B	4	09.02.2026	Дюбанов О.С. Горбовий А.Ю. Філоненко М.М. Погорєловська І.Д. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології в економіці»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«11» грудня 2024

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Підьку Андрію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Технології розробки вебдодатків: сучасні підходи та методології
Затверджена наказом ректора від «10» грудня 2024 № 1852
2. Строк здачі закінченої роботи «2» січня 2026
3. Мета роботи полягає в аналізі сучасних підходів до створення вебдодатків і реалізації інтерактивного конструктора резюме із застосуванням новітніх вебтехнологій.
Об'єкт дослідження процес розробки вебдодатків у сучасному технологічному середовищі.
Предмет дослідження методологія реалізації односторінкового вебдодатка з інструментами персоналізації.
Орієнтовна база дослідження Виконавчий комітет Ірпінської міської ради.
4. Вихідні дані до роботи вимоги до функціоналу вебдодатку-конструктора резюме, наукові та навчальні джерела з веброзробки, матеріали з технологій React, Node.js і MongoDB, а також результати аналізу аналогових вебсистем.
5. Перелік графічного матеріалу Архітектурна схема вебсистеми, діаграма компонентів системи, структура бази даних, порівняльна таблиця аналогових систем, скріншоти інтерфейсу вебдодатка, схема взаємодії клієнт–сервер.
6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)
ВСТУП. РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ВЕБДОДАТКІВ. 1.1. Сучасні технології розробки вебсистем. 1.2. Аналіз аналогових систем конструкторів резюме. 1.3. Постановка задачі. РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСИСТЕМИ КОНСТРУКТОРА РЕЗЮМЕ. 2.1. Формування вимог до системи. 2.2. Розробка структури колекцій (база даних). 2.3. Архітектура системи та її ключові компоненти. РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ КОНСТРУКТОРА РЕЗЮМЕ. 3.1. Вибір інструментів реалізації. 3.2. Опис ключового функціоналу системи. 3.3. Тестування та демонстрація роботи системи. ВИСНОВКИ. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ. ДОДАТКИ.

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційноїроботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.12.2024	До 15.12.2024
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	28.05.2025	28.05.2025
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	08.09.2025	08.09.2025
4	**ІІІ розділ** (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	17.11.2025	17.11.2025
5	**Висновки**	08.12.2025	08.12.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	22.12.2025	22.12.2025
7	Отримання відгуку керівника	16.01.2026	16.01.2026
8	Отримання зовнішньої рецензії	19.01.2026	19.01.2026

8. Дата видачі завдання «11» грудня 2024

9. Керівник кваліфікаційної роботи _____ Гладченко О.І
(підпис) (прізвище, ініціали)

10. Керівник освітньо-професійної програми _____ Омельчук А.А
(підпис) (прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти _____ Підько А.С.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Технології розробки вебдодатків: сучасні підходи та методології

Мета роботи полягає в аналізі сучасних підходів до створення вебдодатків і реалізації інтерактивного конструктора резюме із застосуванням новітніх вебтехнологій.

Завданнями роботи є дослідження фреймворку React, хмарних баз даних, розробка архітектури застосунку та забезпечення ефективної взаємодії з користувачем.

Предметом дослідження є методологія реалізації односторінкового вебдодатка з інструментами персоналізації.

Об'єктом дослідження створення вебдодатків із використанням сучасних фреймворків та хмарних сервісів.

Базою дослідження виступає Виконавчий комітет Ірпінської міської ради.

За результатами дослідження сформульовані рекомендації щодо побудови зручного та масштабованого вебінтерфейсу.

Одержані результати можуть бути використані для створення схожих вебзастосунків у сфері HR та онлайн-сервісів.

Кваліфікаційна робота містить 53 сторінки, 3 таблиці, 17 рисунків, список використаних джерел 40 найменувань, додаток 1.

Ключові слова: вебдодаток, React, конструктор резюме, хмарна база даних, інтерфейс користувача.

Рік виконання кваліфікаційної роботи: 2025.

Рік захисту роботи: 2026.

ABSTRACT

Web Application Development Technologies: Modern Approaches and Methodologies.

The aim of the work is to analyze modern approaches to web application development and implement an interactive resume builder using advanced web technologies.

The tasks include exploring the React framework, cloud databases, designing the application architecture, and ensuring effective user interaction.

The subject of the study is the methodology of implementing a single-page application with personalization tools.

The object of the study is the development of web applications using modern frameworks and cloud services.

The base of the research is the Executive Committee of the Irpin City Council.

The study results include recommendations for building a user-friendly and scalable web interface.

The obtained results can be used for developing similar web services in the HR and online service fields.

Work structure. The qualification work contains 53 pages, 3 tables, 17 figures, and a reference list of 40 items, with appendices 1.

Keywords: web application, React, resume builder, cloud database, user interface.

Year of completion: 2025.

Year of defense: 2026.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – інтерфейс програмної взаємодії між компонентами програмного забезпечення.

Backend – серверна частина вебдодатку, що забезпечує обробку логіки та роботу з даними.

CRUD – базові операції роботи з даними: створення, читання, оновлення та видалення.

Frontend – клієнтська частина вебдодатку, що виконується у веббраузері.

HTTP – протокол передавання гіпертекстових даних.

HTTPS – захищена версія протоколу HTTP з використанням шифрування.

JWT – технологія автентифікації користувачів на основі токенів JSON Web Token.

MongoDB – документоорієнтована нереляційна база даних класу NoSQL.

Node.js – серверне середовище виконання мови програмування JavaScript.

NoSQL – клас нереляційних баз даних, орієнтованих на масштабованість і гнучкість.

React – бібліотека JavaScript для побудови інтерфейсів користувача.

REST – архітектурний стиль побудови вебсервісів на основі HTTP-запитів.

SPA – односторінковий вебзастосунок, у якому навігація здійснюється без повного перезавантаження сторінки.

UI – інтерфейс користувача, що забезпечує взаємодію людини з програмним продуктом.

UX – користувацький досвід, що визначає зручність та ефективність роботи із системою.

ЗМІСТ

ВСТУП ... 3
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ВЕБДОДАТКІВ ... 6
1.1. Сучасні технології розробки вебсистем ... 6
1.2. Аналіз аналогових систем конструкторів резюме ... 7
1.3. Постановка задачі ... 11
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСИСТЕМИ КОНСТРУКТОРА РЕЗЮМЕ ... 13
2.1. Формування вимог до системи ... 13
2.2. Розробка структури колекцій (база даних) ... 15
2.3. Архітектура системи та її ключові компоненти ... 18
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ КОНСТРУКТОРА РЕЗЮМЕ ... 21
3.1. Вибір інструментів реалізації ... 21
3.2. Опис реалізації ключового функціоналу системи ... 22
3.3. Тестування та демонстрація роботи системи ... 40
ВИСНОВКИ ... 49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ... 50
ДОДАТКИ ... 54

ВСТУП

Актуальність теми зумовлена потребою у спрощенні процесу створення якісного резюме для широкого кола користувачів, з урахуванням сучасних вимог до дизайну, функціональності та персоналізації. Широке використання JavaScript-фреймворків, зокрема React, а також хмарних баз даних у таких застосунках демонструє сучасний рівень технологічного розвитку у вебсередовищі. Сучасний етап розвитку інформаційних технологій характеризується стрімким поширенням вебзастосунків, які забезпечують доступність сервісів у будь-який час і з будь-якого пристрою. У цьому контексті актуальним постає впровадження інноваційних підходів до розробки вебдодатків, що поєднують зручність, гнучкість, масштабованість та високу швидкодію. Однією з найбільш потрібних сфер є створення онлайн-інструментів для формування професійної документації, зокрема резюме. Це зумовлює доцільність дослідження теми, пов'язаної з технологіями розробки вебдодатків на прикладі конструктора резюме.

Стан і ступінь розробки проблеми в спеціальній літературі. Проблематика автоматизації створення резюме та використання вебтехнологій для формування професійної документації широко описується в сучасних наукових, прикладних дослідженнях вітчизняних і закордонних авторів. Аналіз спеціальної літератури засвідчує поступовий перехід від шаблонних рішень до вебзастосунків із розширеними можливостями персоналізації та адаптації до вимог ринку праці. У праці В.А. Ковальського та О. В. Романюк обґрунтовано доцільність використання методів обробки природної мови для аналізу якості резюме та визначення його відповідності актуальним вакансіям. Автори наголошують на зростальній ролі автоматизованих систем відбору кандидатів і доводять ефективність семантичного аналізу текстового наповнення резюме [29]. В дослідженні Sudarsanam M., Ajith J., Annamalai J. та Darrel P. розглянуто автоматизований вебконструктор резюме на основі HTML і JavaScript, орієнтований на спрощення процесу створення резюме шляхом заповнення

структурованих форм. Основний акцент зроблено на зручності використання та зменшенні часових витрат користувачів [35]. Комплексний підхід представлено у роботі R. Gopika, A. Kumar та співавторів, де запропоновано інтелектуальний Smart Resume Builder із застосуванням штучного інтелекту та NLP. Автори доводять, що оптимізація резюме під вимоги ATS, автоматичний добір ключових слів і надання рекомендацій у реальному часі підвищують ефективність подання кандидатів [13].

Аналіз наукових джерел підтверджує актуальність досліджуваної проблеми та доцільність подальшої розробки вебдодатків для створення резюме, що поєднують сучасні фреймворки, зручний інтерфейс і можливості персоналізації.

Мета дослідження полягає в аналізі сучасних підходів до створення вебдодатків і реалізації інтерактивного конструктора резюме із застосуванням новітніх вебтехнологій.

Завдання дослідження:

1) дослідити фреймворк React як інструмент побудови інтерфейсу користувача;

2) дослідити хмарні бази даних для зберігання та обробки даних застосунку;

3) доробити архітектуру застосунку для генерації резюме;

4) забезпечити ефективну взаємодію з користувачем у межах розробленого рішення.

Об'єкт дослідження створення вебдодатків із використанням сучасних фреймворків та хмарних сервісів.

Предмет дослідження є методологія реалізації односторінкового вебдодатка з інструментами персоналізації.

Методи дослідження аналіз наукових і технічних джерел, моделювання архітектури вебзастосунка, прототипування, тестування, порівняльна характеристика технологій.

База дослідження Виконавчий комітет Ірпінської міської ради. В цьому органі місцевої ради здійснювалося проходження практики. Організація надала можливість аналізувати вимоги до прикладного вебзастосунку, орієнтованого на взаємодію з користувачем та обробку персоналізованих даних, а також реалізувати практичні аспекти розробки вебінтерфейсу з використанням сучасних технологій.

Науково-практична значущість одержаних результатів полягає у можливості їх практичного використання під час розробки сучасних вебзастосунків із персоналізованим інтерфейсом. Результати дослідження можуть бути застосовані для проєктування односторінкових вебдодатків з використанням бібліотеки React і хмарних сервісів зберігання даних, а також у навчальному процесі під час вивчення вебпрограмування. Розроблений конструктор резюме має достатній рівень готовності до впровадження та може бути адаптований і масштабований відповідно до потреб конкретної організації.

Особистий внесок здобувача полягає в самостійному проєктуванні архітектури конструктора резюме, реалізації клієнтської частини застосунку з використанням бібліотеки React, налаштуванні інтерактивної взаємодії з хмарною базою даних, а також у проведенні тестування функціоналу і внесенні вдосконалень на основі отриманих результатів.

Апробація результатів роботи: Основні результати дослідження було апробовано на І Міжнародній науково-практичній конференції здобувачів вищої освіти та молодих вчених «Економіка і бізнес: нові реалії, технологічні тренди, цифрові інновації», де вони отримали позитивне обговорення та були враховані у подальшій науковій роботі [37].

Кваліфікаційна робота містить 53 сторінки, 3 таблиці, 17 рисунків, список використаних джерел 40 найменувань, додаток 1.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ВЕБДОДАТКІВ

1.1. Сучасні технології розробки вебсистем

Впродовж останніх років веброзробка зазнала суттєвих змін, що зумовлено стрімким розвитком технологій, зростанням вимог до користувацького досвіду та необхідністю забезпечення високої продуктивності й безпеки вебзастосунків. Сучасні вебсистеми стали невід'ємною частиною повсякденного життя, охоплюючи різноманітні сфери від електронної комерції до освітніх платформ, що потребує застосування новітніх підходів і технологій у процесі їх створення.

Однією з ключових особливостей сучасної веброзробки є перехід від традиційних багатосторінкових застосунків до односторінкових (SPA), які забезпечують більш динамічну та інтерактивну взаємодію з користувачем. Це стало можливим завдяки використанню JavaScript-фреймворків, таких як React, Angular та Vue.js, які дозволяють створювати складні інтерфейси з високою швидкістю відгуку та адаптивністю до різних пристроїв [40]. Крім того, важливою тенденцією є впровадження прогресивних вебдодатків (PWA), що поєднують переваги вебсайтів і мобільних застосунків. PWA забезпечують офлайн-доступ, швидке завантаження та можливість встановлення на пристрої користувача, що підвищує зручність використання та залученість аудиторії.

У сфері бекенд-розробки спостерігається зростання популярності мікросервісної архітектури, яка передбачає розподіл функціональності застосунку на незалежні сервіси. Такий підхід сприяє гнучкості, масштабованості та спрощенню процесу оновлення окремих компонентів системи без впливу на її загальну роботу. Значну роль у сучасній веброзробці відіграють хмарні технології, які забезпечують зберігання даних, обчислювальні ресурси та інші сервіси через Інтернет. Використання хмарних платформ, таких як AWS, Google Cloud або Microsoft Azure, дозволяє розробникам швидко розгортати та масштабувати свої застосунки відповідно до потреб бізнесу [34, 36].

Безпека вебсистем також залишається пріоритетним напрямом. Застосування протоколів HTTPS, механізмів автентифікації та авторизації, а

також регулярне тестування на вразливості є необхідними заходами для захисту даних користувачів і забезпечення надійної роботи застосунків [35]. У контексті користувацького досвіду важливим аспектом є адаптивний дизайн, який забезпечує коректне відображення вебсайтів на різних пристроях і розмірах екранів. Використання CSS-фреймворків, таких як Bootstrap або Tailwind CSS, спрощує процес створення адаптивних інтерфейсів та забезпечує їх узгодженість [32]. Сучасні технології розробки вебсистем спрямовані на створення швидких, безпечних, масштабованих та зручних у використанні застосунків, постійно змінюваним вимогам користувачів та бізнесу.

1.2. Аналіз аналогових систем конструкторів резюме

В процесі розробки вебзастосунку-конструктора резюме важливо провести аналіз наявних аналогів, щоб виявити їхні сильні та слабкі сторони. Такий аналіз дозволяє визначити ключові функціональні можливості, які користувачі очікують від подібних систем, а також уникнути повторення недоліків, притаманних вже наявним рішенням. Це сприяє створенню конкурентоспроможного продукту, що відповідає сучасним вимогам ринку та забезпечує високий рівень користувацького досвіду.

Kickresume – це онлайн-платформа для створення професійних резюме та супровідних листів. Вона пропонує користувачам широкий вибір шаблонів, які можна легко налаштувати відповідно до індивідуальних потреб (рис. 1.1) [19]. Серед переваг Kickresume варто відзначити інтуїтивно зрозумілий інтерфейс, наявність AI-помічника для написання текстів, а також можливість створення супровідних листів, що відповідають стилю резюме. Однак, деякі користувачі відзначають обмежену кількість безкоштовних шаблонів та необхідність оформлення підписки для доступу до розширених функцій [18].

Resume Genius – це ще одна популярна платформа для створення резюме, яка пропонує користувачам простий процес заповнення інформації за допомогою покрокового майстра (рис. 1.2).

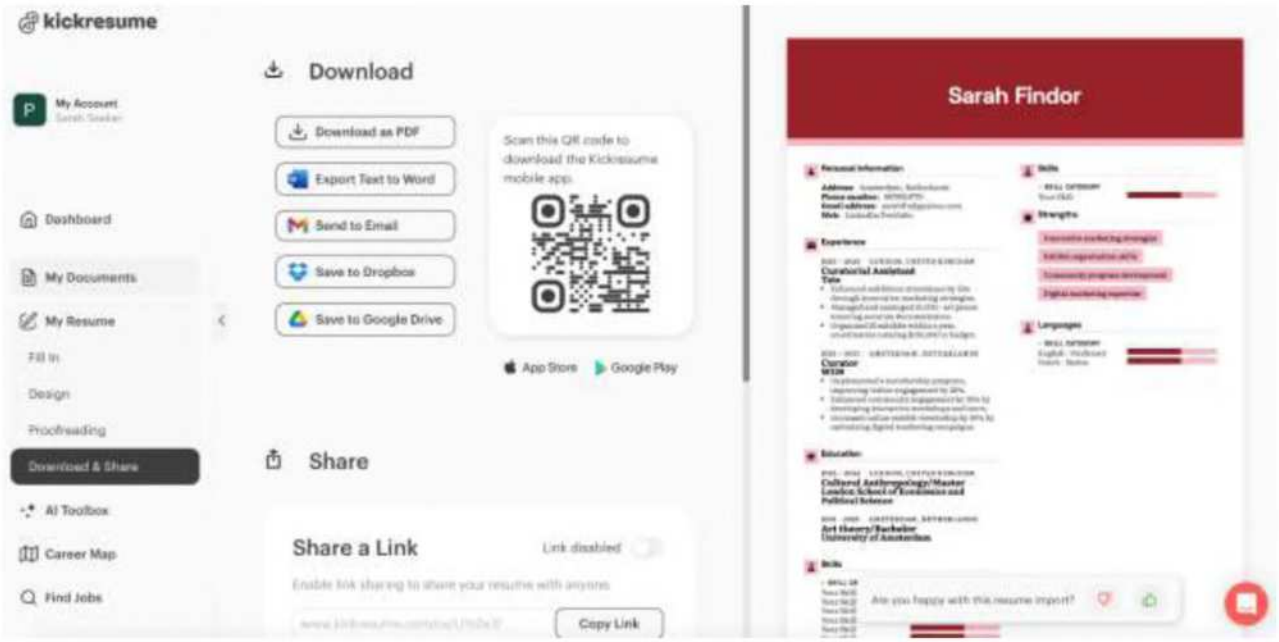


Рис. 1.1 Інтерфейс системи Kickresume

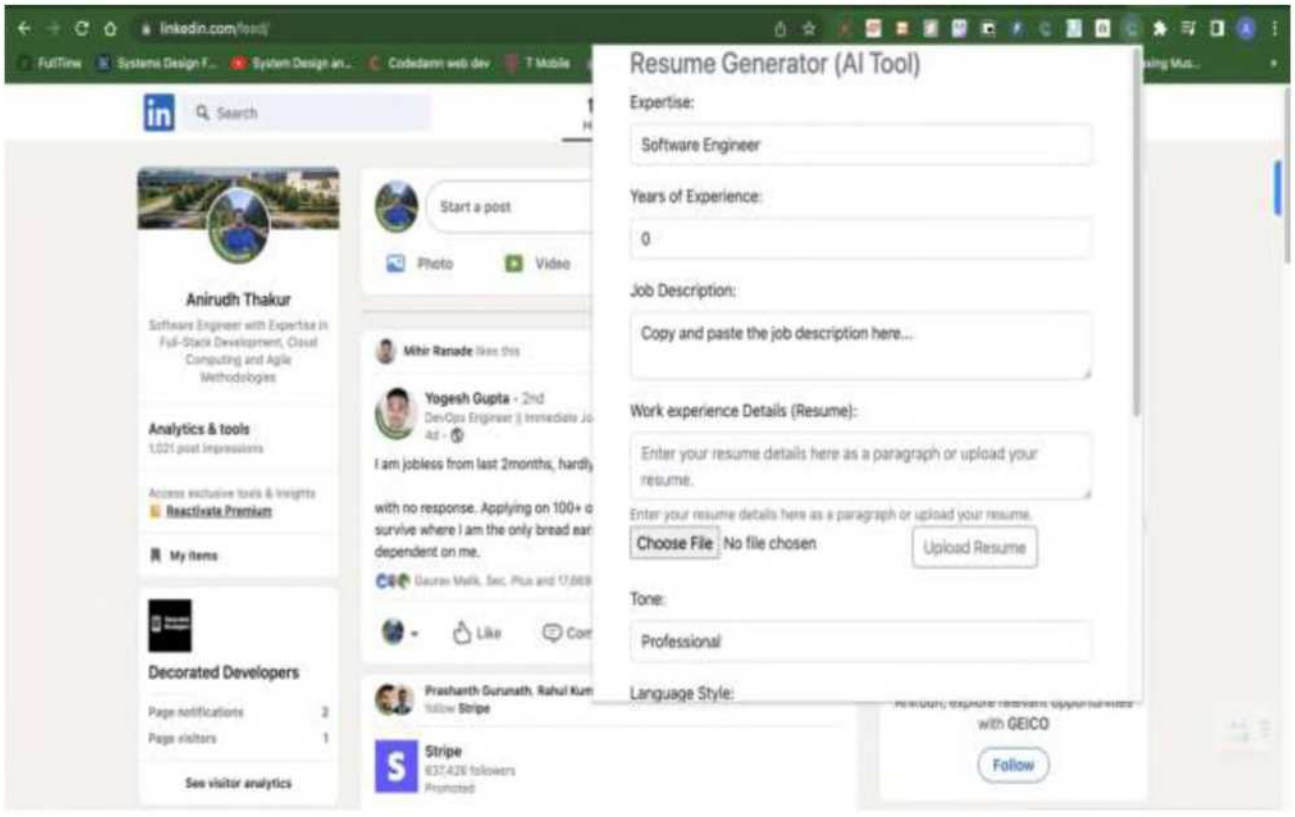


Рис. 1.2 Інтерфейс системи Resume Genius

Система надає готові фрази для опису досвіду та навичок, що значно спрощує процес створення резюме для користувачів без досвіду в написанні подібних документів. До переваг Resume Genius можна віднести зручність

використання та наявність великої бібліотеки шаблонів. Проте, деякі користувачі скаржаться на обмежені можливості налаштування дизайну та необхідність оплати для завантаження готового резюме [31].

Resume.io – це платформа, яка дозволяє створювати резюме та супровідні листи за допомогою сучасних шаблонів, що відповідають вимогам систем автоматичного відбору кандидатів (ATS).

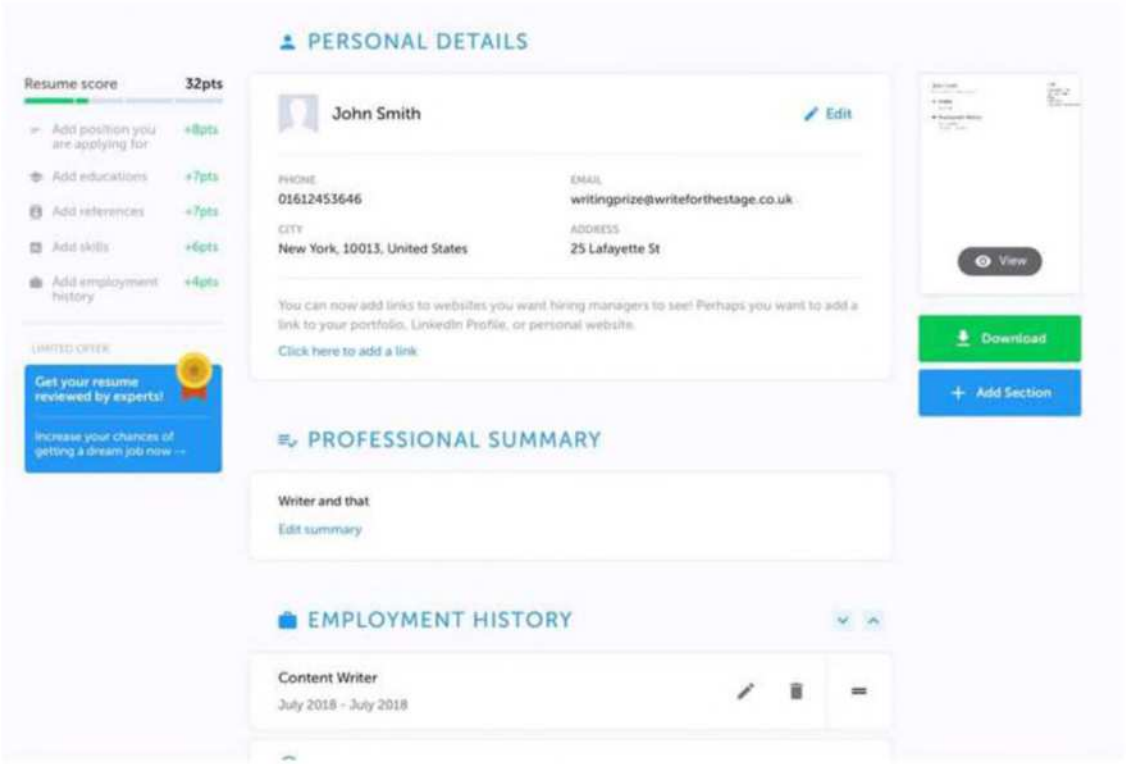


Рис. 1.3 Інтерфейс системи Resume.io

Серед переваг Resume.io варто відзначити простоту використання, можливість збереження та редагування кількох версій резюме, а також наявність інструментів для аналізу ефективності резюме. Однак, деякі користувачі відзначають обмежену кількість безкоштовних функцій та необхідність оформлення підписки для доступу до повного функціоналу [4].

Для ефективного проєктування власного конструктора резюме доцільним є проведення порівняльного аналізу наявних аналогів за певними ключовими критеріями (див. табл. 1.1).

Таблиця 1.1

Порівняльна таблиця аналогових систем

Критерій / Платформа	Kickresume	Resume Genius	Resume.io
Простота інтерфейсу	5	4	5
Кількість шаблонів	4	4	5
Якість дизайну шаблонів	5	3	5
Наявність AI-помічника	5	3	3
Можливість кастомізації	4	3	4
Безкоштовний функціонал	2	2	2
Створення супровідних листів	5	4	4
Підтримка ATS-сумісності	3	4	5
Додаткові інструменти (аналітика, збереження версій)	3	3	5
Загальна зручність	5	4	5

Джерело: [розроблено автором на основі аналізу аналогових вебсистем]

Зазначимо, що кожна система оцінюється за шкалою від 1 (найгірше) до 5 (відмінно).

На основі наведеного порівняння можна зробити висновок, що всі три системи мають достатньо високий рівень зручності використання та якості шаблонів. Resume.io демонструє найкращі показники в контексті професійного використання, тоді як Kickresume вигідно вирізняється інтелектуальною підтримкою користувача через AI-помічник і добре продуманий інтерфейс. Resume Genius, хоча й поступається в аспектах дизайну та гнучкості, залишається доступним варіантом з базовим функціоналом.

Результати аналізу дозволяють визначити напрямки, які слід враховувати при розробці власного конструктора: надання максимально повного функціоналу без потреби підписки та гнучке налаштування шаблонів.

1.3. Постановка задачі

Проведений аналіз аналогових систем-конструкторів резюме дав змогу не лише оцінити поточний стан ринку, але й виокремити найважливіші функціональні компоненти, які забезпечують зручність, ефективність та гнучкість роботи з подібними вебзастосунками. Ці спостереження лягли в основу формулювання вимог до функціоналу системи, яка розробляється в межах даного проєкту. Основним завданням є створення інтерактивного, інтуїтивно зрозумілого конструктора резюме з можливістю збереження, редагування та персоналізації результатів. Користувач повинен мати змогу швидко створити документ що професійно виглядає без потреби у знаннях дизайну чи програмування. З огляду на потреби кінцевого користувача та тенденції сучасних вебрішень, визначено перелік функціональних можливостей, які мають бути реалізовані в межах розробки:

– Автентифікація користувача, забезпечення безпечної реєстрації, входу в систему та керування сеансами користувача за допомогою сучасного механізму JSON Web Token (JWT) [15]. Це дозволить гарантувати конфіденційність та персоналізований доступ до даних;

– Панель керування резюме, реалізація інтерфейсу для зручного перегляду всіх створених резюме користувача, з можливістю їх редагування або видалення. Така панель стане центральною точкою керування інформацією;

– Редактор резюме в реальному часі, інтерактивне середовище, яке дозволяє миттєво бачити зміни у резюме в міру введення даних. Це значно покращує досвід користувача, дозволяючи експериментувати зі змістом та оформленням;

– Перемикання шаблонів, підтримка кількох шаблонів оформлення резюме, які користувач може змінювати без втрати вмісту. Такий підхід дає змогу легко адаптувати документ до різних вимог чи стилістичних вподобань;

– Вибір колірної палітри, функція зміни кольорового оформлення шаблону дозволяє персоналізувати вигляд резюме відповідно до індивідуального стилю користувача;

– Підтримка завантаження у форматі PDF, реалізація можливості експорту створеного резюме у формат PDF, що є стандартом для подання документів при працевлаштуванні;

– Збереження та редагування резюме, можливість безпечно зберігати кілька версій резюме в обліковому записі користувача, з подальшим доступом до редагування в будь-який момент;

– Завантаження та попередній перегляд зображень, підтримка додавання фото профілю з можливістю його попереднього перегляду в режимі реального часу, що дозволяє оцінити візуальну цілісність документа перед його фіналізацією.

З метою реалізації вказаного функціоналу необхідно також спроєктувати архітектуру застосунку, яка включатиме клієнтську частину (інтерфейс користувача), серверну частину (керування автентифікацією та базою даних). Окрему увагу буде приділено моделюванню структури бази даних. Вона повинна забезпечувати ефективне зберігання користувацьких облікових даних, метаданих резюме, шаблонів, зображень. Використання хмарного середовища для зберігання даних дозволить забезпечити надійність та доступність системи незалежно від фізичного розташування користувача.

На завершальному етапі буде проведено повноцінне тестування розробленої системи, з метою перевірки коректності виконання всіх функцій, відповідності інтерфейсу очікуванням користувача та відсутності критичних помилок. Тестування охоплюватиме як функціональні сценарії взаємодії, так і перевірку безпеки та стабільності системи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСИСТЕМИ КОНСТРУКТОРА РЕЗЮМЕ

2.1. Формування вимог до системи

Етап формулювання вимог до програмної системи є одним із ключових у процесі її розробки, оскільки саме на цьому етапі визначається, яку конкретну проблему вирішує система, які функції має реалізовувати, в яких умовах експлуатуватись та які обмеження і залежності слід врахувати. Для проєктування конструктора резюме вебзастосунку, орієнтованого на широке коло користувачів без спеціальних знань у сфері IT чи дизайну, особливо важливо сформувати чіткі, структуровані та обґрунтовані вимоги.

Умовно всі вимоги до системи поділяються на дві основні категорії [39]:

– функціональні вимоги, описують що саме повинна робити система: які сервіси вона має надавати, як вона повинна реагувати на дії користувача, які сценарії взаємодії реалізовуються;

– нефункціональні вимоги, які визначають, як саме система повинна функціонувати: продуктивність, надійність, масштабованість, зручність використання, безпека, адаптивність тощо.

У контексті розробки конструктора резюме функціональні вимоги мають першочергове значення, оскільки саме вони визначають базову логіку взаємодії користувача з системою, від входу в обліковий запис до генерації фінального документу у форматі PDF. Нижче наведено основні функціональні вимоги до системи (див. табл. 2.1).

Функціональні вимоги чітко окреслюють обсяг задач, які має реалізовувати вебзастосунок. Вони враховують основні очікування кінцевих користувачів і спрямовані на створення сервісу, який дозволяє зручно, швидко та ефективно формувати якісне резюме з мінімальними технічними зусиллями.

Нефункціональні вимоги мають важливе значення для забезпечення якості взаємодії користувача з системою. Їхнє дотримання дозволяє досягти високої стабільності, надійності та зручності розробленого вебзастосунку в умовах

реального використання. Нижче наведено основні нефункціональні вимоги до системи (див. табл. 2.2).

Таблиця 2.1

Функціональні вимоги до системи

№	Функціональна вимога
1	Система повинна забезпечувати реєстрацію, вхід та автентифікацію користувача із захистом сесій через JWT.
2	Користувач має мати змогу створювати кілька окремих резюме, що
	зберігаються у його профілі.
3	Резюме повинно бути доступне для редагування в реальному часі з відображенням змін.
4	Передбачена можливість перемикання шаблонів резюме без втрати введеної інформації.
5	Користувач може змінювати колірну палітру резюме для персоналізації оформлення.
6	Система має підтримувати завантаження резюме у форматі PDF.
7	Реалізовано функцію завантаження та попереднього перегляду фотографії профілю.
8	Користувач повинен мати змогу переглядати, редагувати або видаляти наявні резюме.

Таблиця 2.2

Нефункціональні вимоги до системи

№	Нефункціональна вимога
1	Система повинна забезпечувати стабільну роботу при великій кількості одночасних користувачів.
2	Інтерфейс має бути адаптивним та коректно відображатись на різних пристроях (десктоп, планшет, смартфон).

3	Всі користувацькі дані мають зберігатися у безпечному хмарному середовищі з автентифікацією доступу.
4	Система повинна підтримувати розширення функціоналу без порушення наявної структури (масштабованість).
5	Вебзастосунок має демонструвати високу швидкодію при рендерингу змін у реальному часі.
6	Інтерфейс повинен бути зрозумілим і зручним для користувачів без спеціальної підготовки.
7	Завантаження PDF-файлів повинно здійснюватися без втрат у форматуванні.
8	Всі дані користувача мають бути синхронізовані з базою даних у режимі реального часу.

В результаті формування вимог визначено базові та розширені можливості конструктора резюме, які ляжуть в основу його архітектури, структури бази даних і наступного етапу програмної реалізації.

2.2. Розробка структури колекцій (база даних)

Для реалізації системи зберігання та обробки даних у рамках конструктора резюме було обрано документоорієнтовану базу даних MongoDB, яка належить до класу NoSQL-рішень. Такий вибір зумовлений низкою переваг, що мають особливу цінність у динамічному середовищі веброзробки: MongoDB забезпечує високу гнучкість, дозволяючи змінювати структуру даних без необхідності вносити зміни до схеми, підтримує горизонтальне масштабування, що важливо при зростанні кількості користувачів, та характеризується високою продуктивністю при роботі з великим обсягом записів [22].

У MongoDB інформація організована у вигляді колекцій, логічних груп документів, що є аналогами таблиць у реляційних СУБД. Проте, на відміну від класичних таблиць, документи в колекції можуть мати варіативну структуру, що значно полегшує моделювання гнучких і вкладених сутностей. Кожен документ

містить набір пар «ключ–значення» та має унікальний ідентифікатор _id, який генерується автоматично [12]. Такий підхід ідеально підходить для зберігання різнорівневої інформації про користувачів, резюме, шаблони та пов'язані елементи.

У межах даного проєкту сформовано дві основні колекції: users і resumes, кожна з яких подає ключову частину логіки застосунку.

Колекція users відповідає за зберігання облікових даних користувачів та забезпечує можливість автентифікації і персоналізації досвіду роботи в системі. Документ користувача містить такі поля:

- _id – унікальний ідентифікатор (тип ObjectId);
- name – повне ім'я користувача;
- email – електронна пошта, що використовується як логін;
- password – захешований пароль із використанням алгоритму bcrypt;
- profileImageUrl – посилання на зображення профілю;
- createdAt, updatedAt – часові мітки створення та останнього оновлення документа;

Така структура дозволяє не лише виконувати базові операції входу/реєстрації, а й організовувати зберігання додаткових атрибутів профілю з можливістю їх розширення в майбутньому.

Колекція resumes зберігає всі створені користувачами резюме, включно з метаданими, візуальним оформленням та змістовними блоками. Її структура є складнішою, оскільки кожне резюме об'єднує в собі цілу сукупність вкладених підрозділів, зокрема:

- userId – зв'язок із автором резюме (ObjectId);
- title – назва резюме, яка задається користувачем;
- template – об'єкт, що зберігає тип шаблону (theme);
- colorPalette – масив кольорів, обраних для оформлення;
- profileInfo – загальні відомості про кандидата: ім'я, посада, короткий опис, фотографія;

– contactInfo – контактні дані (електронна пошта, телефон, адреса, соціальні мережі);

– workExperience, education, skills, projects, certifications, languages, interests – масиви вкладених об’єктів, що подають відповідні розділи резюме.

Кожен з розділів резюме зберігається як масив об’єктів, що включає поля з основною інформацією (наприклад, company, role, startDate, description у випадку досвіду роботи або degree, institution для освіти). Це дозволяє легко додавати, редагувати та сортувати дані без створення додаткових зв’язків.

Крім того, документ резюме містить:

– createdAt, updatedAt – часові мітки створення та оновлення;

– thumbnailLink – посилання на попередній перегляд згенерованого PDF-файлу.

Попри те, що база даних MongoDB належить до документоорієнтованих і не підтримує жорстко визначених зв’язків між колекціями на рівні схеми, як це притаманно реляційним СУБД, логічні взаємозв’язки між даними все ж мають місце. Зокрема, поле userId у резюме вказує на конкретного користувача, а аналогічно в інших випадках окремі поля можуть зберігати ідентифікатори документів з інших колекцій. З метою кращого розуміння логіки побудови даних та загальної структури системи, доцільним є умовне подання документів MongoDB у табличній формі, що дозволяє чіткіше відстежити їхню взаємодію (рис. 2.1).



Рис. 2.1 Колекції user та resumes подані в табличному форматі

Підсумовуючи, можна зробити висновки, що структурування даних дає змогу ефективно керувати усіма елементами даних в документах, без потреби звертання до зовнішніх колекцій, що прискорює час відгуку системи та спрощує запити до бази даних. Завдяки цьому MongoDB виступає не лише як сховище, а й як інструмент подання логічно завершених одиниць інформації, які зручно обробляються на клієнтському рівні.

2.3. Архітектура системи та її ключові компоненти

Для побудови складної вебсистеми, що включає взаємодію між користувачем, інтерфейсом, сервером і базою даних, доцільним є подання загальної структури рішення у вигляді архітектурної діаграми. Одним із найбільш наочних способів візуалізації подібних систем є діаграма компонентів, яка демонструє розподіл системи на окремі логічні частини (компоненти) та зв'язки між ними. Така діаграма дозволяє швидко охопити логіку побудови проєкту, визначити основні точки взаємодії та прослідкувати залежності між технологічними модулями. На рисунку 2.2 зображено компонентну архітектуру розробленого конструктора резюме. Система поділена на три основні рівні: клієнтський (користувач і браузер), серверний (вебсервер з фронтендом і бекендом) та рівень бази даних.

Першим компонентом є користувач, який взаємодіє з системою через браузер на персональному комп'ютері або мобільному пристрої. Саме в браузері відбувається візуалізація інтерфейсу, реалізованого за допомогою React. Увесь обмін даними з сервером відбувається через HTTP-запити, що дозволяє забезпечити швидкий і гнучкий зв'язок між клієнтом і серверною частиною системи.

Серверний рівень системи виконує ключову роль у забезпеченні логіки застосунку, обробці запитів, автентифікації користувачів та взаємодії з базою даних. Він умовно поділяється на два взаємопов'язані компоненти фронтенд-

сервер (клієнтська частина) та бекенд-сервер (серверна логіка), які працюють у тісній зв'язці та забезпечують повноцінне функціонування вебзастосунку.

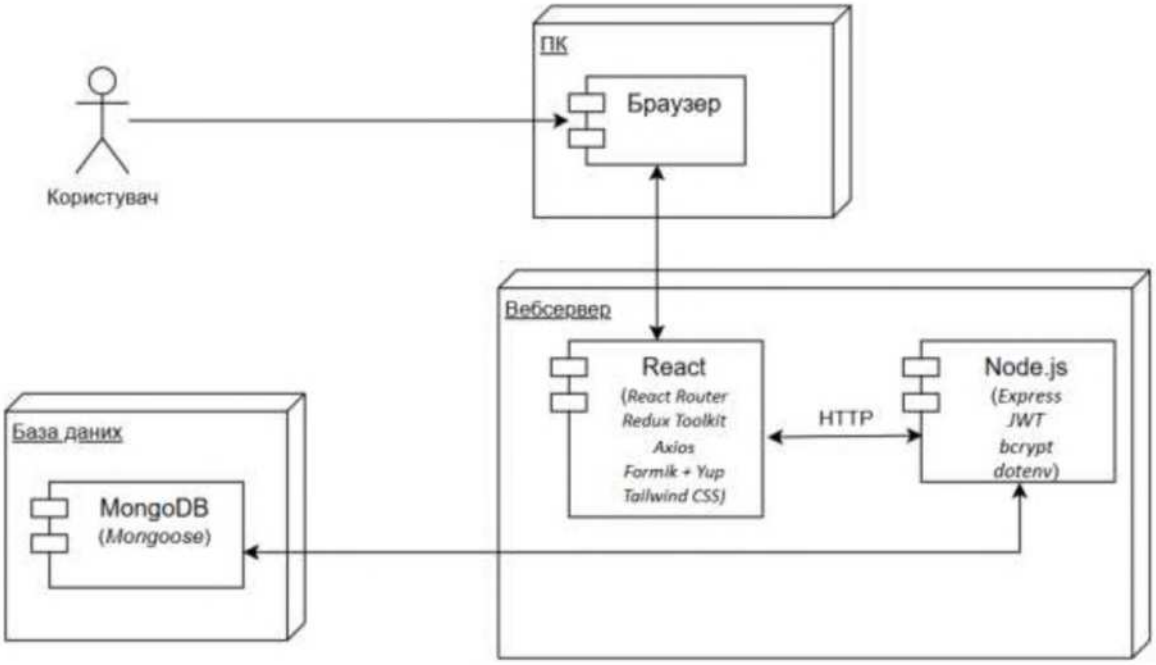


Рис. 2.2 Архітектура системи

Фронтенд реалізовано з використанням JavaScript-бібліотеки React, яка дозволяє будувати динамічні інтерфейси на основі компонентного підходу [25]. Для організації навігації між сторінками застосунку використовується React Router, що забезпечує зручну маршрутизацію без необхідності перезавантаження сторінки [27]. Управління глобальним станом здійснюється через Redux Toolkit оптимізовану бібліотеку для керування станом, що дозволяє ефективно координувати оновлення інтерфейсу у відповідь на зміни в даних [28].

Для комунікації між клієнтською та серверною частиною застосовується Axios бібліотека для надсилання HTTP-запитів, яка забезпечує асинхронну взаємодію з бекендом [2]. Обробка форм та їх валідація реалізовані за допомогою комбінації Formik і Yup, що дозволяє створювати гнучкі, керовані форми з чіткими правилами перевірки введених даних [11]. Візуальне оформлення інтерфейсу виконується з використанням Tailwind CSS сучасного CSS-

фреймворку, який забезпечує адаптивний, привабливий і послідовний дизайн інтерфейсу на всіх пристроях [30].

Бекенд-сервер створено на основі середовища Node.js, яке забезпечує виконання JavaScript на стороні сервера [24]. Основною бібліотекою для побудови REST API є Express легковагий та гнучкий фреймворк, що дозволяє швидко створювати маршрути, обробники запитів і керувати середовищем виконання [10]. Захист даних користувача реалізовано за допомогою JWT (JSON Web Token), який дозволяє формувати маркери доступу та забезпечувати авторизований доступ до приватних маршрутів [17]. Для безпечного зберігання паролів використовується bcrypt криптографічна бібліотека для хешування з додаванням сольових значень, що гарантує захист від простого дешифрування [3]. Конфігураційні параметри, такі як адреси підключення до бази даних або секретні ключі, зберігаються в окремому файлі .env та обробляються через бібліотеку dotenv [8]. Фронтенд відповідає за створення, відображення та взаємодію з інтерфейсом користувача, тоді як бекенд за логіку обробки даних, безпеку, комунікацію з базою даних та обслуговування запитів. Їхня тісна інтеграція забезпечує стабільну роботу всієї системи, дозволяючи реалізувати складний, масштабований і зручний конструктор резюме.

Третій рівень поданий в MongoDB, що виконує роль централізованого сховища даних [21]. Взаємодія з базою реалізована через ODM-бібліотеку Mongoose, яка забезпечує зручну роботу з колекціями, документами та схемами [23]. MongoDB зберігає всі дані користувачів (credentials, профілі, посилання на фото) та створені резюме з усіма вкладеними блоками (досвід, освіта, навички, сертифікати тощо). Комунікація між Node.js і MongoDB здійснюється безпосередньо через запити, які формуються на основі дій користувача в інтерфейсі. Описана архітектура демонструє чітке розділення функціоналу між компонентами: клієнтська частина зосереджена на зручності та інтерфейсі, серверна на логіці та безпеці, а база даних на збереженні та структуризації інформації. Подібна багаторівнева модель є ефективною для побудови масштабованих, продуктивних і безпечних вебзастосунків.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ КОНСТРУКТОРА РЕЗЮМЕ

3.1. Вибір інструментів реалізації

В процесі розробки конструктора резюме було обрано низку сучасних інструментів, які сприяють ефективному створенню вебзастосунків, забезпечують зручність для розробника, високу продуктивність і можливість масштабування. Застосовані технології охоплюють як клієнтську, так і серверну частину, включаючи мови розмітки, стилізації, програмування, середовище розробки та систему керування базами даних. Основу структури вебсторінок проєкту формує мова розмітки HTML, яка слугує каркасом інтерфейсу та визначає базову логіку компонування вмісту сторінки: заголовки, абзаци, списки, кнопки, посилання тощо. Саме HTML забезпечує логічну ієрархію та структурованість усіх елементів інтерфейсу [14]. Для стилізації сторінок використано CSS мову, яка відповідає за візуальне оформлення вебінтерфейсу. Вона дозволяє налаштовувати кольори, шрифти, відступи, розміщення елементів, а також реалізовувати адаптивність інтерфейсу під різні типи пристроїв. Таким чином, CSS забезпечує не лише привабливий зовнішній вигляд, а й покращує загальну зручність користування системою [7]. Для створення інтерактивного функціоналу використано мову програмування JavaScript, яка є стандартом у сфері фронтенд-розробки. Завдяки їй реалізовано динамічні оновлення контенту, обробку подій, взаємодію з API, а також валідацію форм без перезавантаження сторінки. JavaScript дозволяє підтримувати сучасну модель односторінкових додатків (SPA) і тісно інтегрується з усіма іншими інструментами проєкту [16].

Одним із ключових елементів системи є React, JavaScript-бібліотека, створена компанією Meta. Вона реалізує компонентний підхід до розробки інтерфейсів, що дозволяє розбивати весь UI на незалежні, багаторазово використовувані частини. React забезпечує ефективне оновлення інтерфейсу за рахунок використання віртуального DOM та оптимізованої обробки змін стану,

що суттєво підвищує швидкість та зручність взаємодії з додатком [26].

Уся розробка здійснювалася в середовищі Visual Studio Code (VS Code) сучасному, безкоштовному редакторі коду від компанії Microsoft. Це середовище підтримує велику кількість мов програмування, має гнучку систему розширень, інтеграцію з Git, підтримку налагодження, автодоповнення, перевірку синтаксису, а також термінал для керування проєктом. Завдяки кросплатформенності та зручному інтерфейсу VS Code є популярним вибором як для початківців, так і для досвідчених розробників [33]. На серверному рівні обрано документоорієнтовану базу даних MongoDB, розгорнуту на хмарній платформі MongoDB Atlas. Це рішення забезпечує зберігання даних у форматі BSON, автоматичне масштабування, високу доступність, резервне копіювання та вбудований інструментарій для моніторингу. Завдяки Atlas, розгортання та керування базою не вимагає окремої інфраструктури усе виконується через зручний вебінтерфейс, що значно спрощує адміністрування проєкту [20].

3.2. Опис реалізації ключового функціоналу системи

Розроблена система конструктора резюме реалізована у вигляді двох окремих частин клієнтської та серверної, які взаємодіють між собою через REST API. Такий підхід забезпечує чітке розділення функціоналу, спрощує обслуговування коду та дозволяє масштабувати обидві частини незалежно одна від одної (див. рис. 3.1).

Серверна частина написана на платформі Node.js з використанням Express. Вона відповідає за обробку HTTP-запитів, автентифікацію користувачів, роботу з базою даних MongoDB, зберігання зображень та генерацію PDF-файлів резюме. Основні елементи серверної частини:

– server.js головний файл запуску застосунку, в якому ініціалізується сервер, підключення до бази даних та реєстрація маршрутів;

– routes/ каталог, у якому визначені маршрути запитів. Наприклад, authRoutes.js обробляє реєстрацію та вхід, а resumeRoutes.js – CRUD-операції над резюме;

– controllers/ каталог із бізнес-логікою для обробки запитів. authController.js обробляє реєстрацію/логін, resumeController.js – збереження та оновлення резюме;

– middlewares/ проміжні обробники запитів. Зокрема, authMiddleware.js перевіряє токен авторизації (JWT), а uploadMiddleware.js обробляє завантаження зображень;

– models/ моделі даних у форматі Mongoose. User.js містить схему користувача, а Resume.js структуру документа резюме.

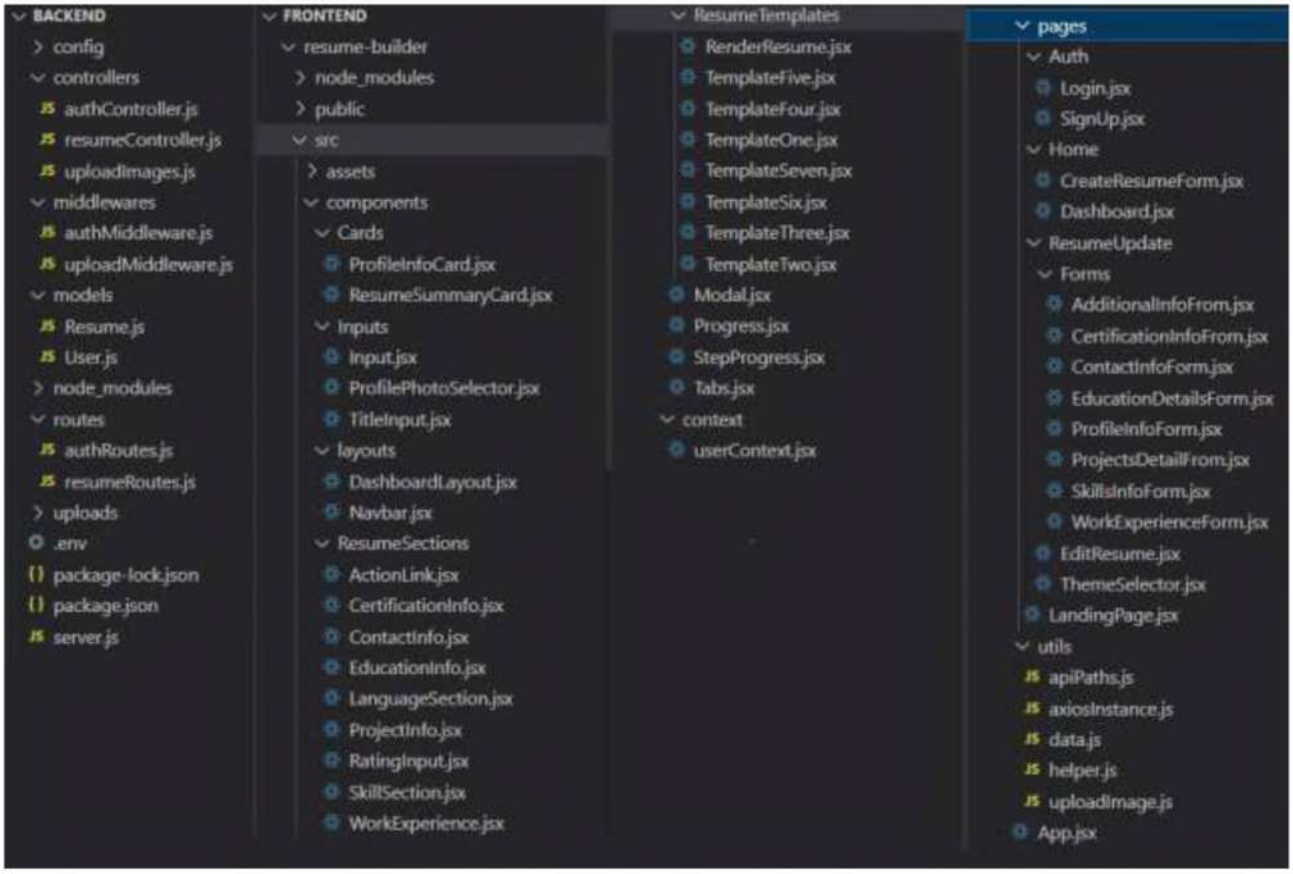


Рис. 3.1 Структура проєкту

Клієнтська частина створена на базі React, вона забезпечує візуальний інтерфейс користувача, взаємодію з сервером, створення та редагування резюме в реальному часі. Основні каталоги клієнтської частини:

– pages/ містить основні сторінки застосунку. У підкаталозі Auth/ реалізовані сторінки входу (Login.jsx) та реєстрації (SignUp.jsx). У Home/ панель користувача (Dashboard.jsx) та форма створення резюме (CreateResumeForm.jsx);

– pages/ResumeUpdate/Forms/ окремі компоненти форми редагування резюме: інформація про користувача (ProfileInfoForm.jsx), досвід (WorkExperienceForm.jsx), навички (SkillsInfoForm.jsx) тощо;

– ResumeTemplates/ шаблони для візуалізації резюме: TemplateOne.jsx, TemplateTwo.jsx, ... TemplateSeven.jsx. Вибір шаблону відбувається динамічно;

– components/ спільні компоненти, зокрема ProfileInfoCard.jsx, ResumeSummaryCard.jsx, Navbar.jsx, DashboardLayout.jsx;

– ResumeSections/ відповідає за вивід структурованої інформації в шаблоні резюме: блоки про освіту (EducationInfo.jsx), сертифікати (CertificationInfo.jsx), проєкти (ProjectInfo.jsx) тощо;

– utils/ службові функції, зокрема apiPaths.js (шляхи до API), axiosInstance.js (конфігурація запитів), helper.js (допоміжні функції), uploadImage.js (завантаження зображення);

– context/userContext.jsx глобальний контекст, який зберігає інформацію про автентифікованого користувача.

Реалізація такого розподілу дозволила досягти високої модульності та повторного використання компонентів. Кожен файл або каталог відповідає за окрему функціональність, що полегшує підтримку та розширення системи.

Щоб уникнути надмірного опису всіх частин проєкту та не повторювати однотипні елементи структури, далі буде зосереджено увагу лише на ключових файлах серверної та клієнтської частини системи. Їхній огляд дозволяє зрозуміти загальний принцип функціонування конструктора резюме, логіку взаємодії між компонентами, обробку даних і структуру додатку без потреби аналізувати увесь код проєкту.

Дослідимо серверну частину. Файл server.js є точкою входу до серверної частини проєкту. Тут відбувається початкова конфігурація: підключення змінних середовища, ініціалізація Express-додатку, під'єднання до бази даних MongoDB, налаштування CORS, реєстрація маршрутів та запуск сервера [6]. Важливо, що застосунок дотримується розділення обов'язків: усі запити до автентифікації та до роботи з резюме передаються відповідним маршрутам, а

запити до завантажених зображень безпосередньо обслуговуються через статичний доступ:

```
require("dotenv").config();
const express = require("express");
const cors = require("cors");
const path = require("path");
const connectDB = require('./config/db');
const authRoutes = require('./routes/authRoutes');
const resumeRoutes = require('./routes/resumeRoutes');

const app = express();

app.use(cors({ origin: process.env.CLIENT_URL || "*", methods: ["GET", "POST", "PUT", "DELETE"], allowedHeaders: ["Content-Type", "Authorization"] }));
connectDB();
app.use(express.json());
app.use("/api/auth", authRoutes);
app.use("/api/resume", resumeRoutes);
}));

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

У цьому фрагменті видно, як здійснюється вся базова настройка. Важливо, що завдяки структурованому розділенню на middleware та маршрути система легко підтримується та масштабується.

Файл authRoutes.js містить маршрути для взаємодії з користувачем: реєстрація, вхід, отримання профілю та завантаження фото. Одним із ключових аспектів є використання middleware protect для захисту особистих даних користувача та middleware upload для обробки файлів [9]. Нижче наведено

фрагмент коду, що демонструє загальну структуру маршрутизації та обробку зображень:

```
const express = require("express");
const { registerUser, loginUser, getUserProfile } = require("../controllers/authController");
const { protect } = require("../middlewares/authMiddleware");
const upload = require('../middlewares/uploadMiddleware');

const router = express.Router();

router.post("/register", registerUser);
router.post("/login", loginUser);
router.get("/profile", protect, getUserProfile);

router.post("/upload-image", upload.single("image"), (req, res) => {
if (!req.file) return res.status(400).json({ message: "No file uploaded" });

module.exports = router;
```

Тут важливо відзначити: шлях /profile захищено JWT-аутентифікацією, а шлях /upload-image дозволяє користувачам завантажувати фото, яке зберігається в локальну папку /uploads, що відкриває можливість додавати аватар у профіль.

resumeRoutes.js файл відповідає за всі дії з резюме: створення, отримання, оновлення, видалення, а також завантаження пов'язаних зображень. Усі маршрути також захищені middleware protect, що гарантує, що тільки авторизований користувач може взаємодіяти зі своїми даними. Нижче подано код, який демонструє головну логіку побудови маршрутизації:

```
const express = require("express");
} = require("../controllers/resumeController");
const { protect } = require("../middlewares/authMiddleware");
const { uploadResumeImages } = require("../controllers/uploadImages");
```

```
const router = express.Router();

router.post("/", protect, createResume);
router.get("/", protect, getUserResumes);
router.get("/:id", protect, getResumeById);
router.put("/:id", protect, updateResume);
router.put("/:id/upload-images", protect, uploadResumeImages);
router.delete("/:id", protect, deleteResume);

module.exports = router;
```

У цьому фрагменті реалізовано повний CRUD-інтерфейс для резюме. Важливим є те, що кожна дія чітко винесена в окремий контролер, що підвищує модульність коду. Окремий маршрут upload-images дозволяє користувачеві додати власне фото безпосередньо до певного резюме, що розширює можливості кастомізації.

Файл authController.js відповідає за автентифікацію користувачів. У ньому реалізовано три основні функції: реєстрація, вхід та отримання профілю користувача. Найважливішим фрагментом є функція registerUser, де виконується вся логіка реєстрації:

```
const registerUser = async (req, res) => {
 try {
 const { name, email, password, profileImageUrl } = req.body;
 const userExists = await User.findOne({ email });
 if (userExists) {
 return res.status(400).json({ message: "User already exists" });
 }
 const salt = await bcrypt.genSalt(10);
 const hashedPassword = await bcrypt.hash(password, salt);
 const user = await User.create({
```

```
name,
email,
password: hashedPassword,
profileImageUrl,
});
res.status(201).json({
_id: user._id,
name: user.name,
email: user.email,
profileImageUrl: user.profileImageUrl,
token: generateToken(user._id),
});
}
};
```

Тут перевіряється, чи існує користувач із вказаною електронною адресою, далі відбувається хешування пароля за допомогою bcrypt, створення нового користувача через Mongoose, та формування JWT-токена для подальшої автентифікації. Інші функції (loginUser, getUserProfile) працюють за схожим принципом: перевіряють дані, ідентифікують користувача та повертають відповідь з токеном або інформацією.

Файл resumeController.js реалізує основну бізнес-логіку, пов'язану з резюме: створення, перегляд, оновлення та видалення. Функція createResume створює шаблон нового резюме зі стандартними порожніми блоками:

```
const createResume = async (req, res) => {
try {
const { title } = req.body;
const defaultResumeData = {
profileInfo: {
profileImg: null,
previewUrl: "",
```

```
fullName: "",
designation: "",
summary: "",
},
contactInfo: {
email: "", phone: "", location: "", linkedin: "", github: "", website: "",
},
workExperience: [{ company: "", role: "", startDate: "", endDate: "",
description: "" }],
education: [{ degree: "", institution: "", startDate: "", endDate: "" }],
skills: [{ name: "", progress: 0 }],
projects: [{ title: "", description: "", github: "", liveDemo: "" }],
certifications: [{ title: "", issuer: "", year: "" }],
languages: [{ name: "", progress: 0 }],
interests: [""],
};
}
};
```

У цій функції ініціалізується новий документ з фіксованою структурою, який потім зберігається у базі даних MongoDB. Інші функції (getUserResumes, getResumeById, updateResume, deleteResume) відповідають за доступ до відповідних даних користувача. Особливо важливою є функція deleteResume, яка, окрім видалення запису з бази, також очищує пов'язані з ним файли з диска (uploads).

Ці два контролери є основою логіки всієї системи: вони забезпечують безпечну реєстрацію та управління вмістом резюме для кожного користувача.

Файл authMiddleware.js відповідає за захист маршрутів, які потребують авторизації. Його мета перевірити наявність і дійсність JWT-токена перед доступом до захищеного ресурсу:

```
const protect = async (req, res, next) => {
```

```
try {
let token = req.headers.authorization;

if (token && token.startsWith("Bearer")) {
token = token.split(" ")[1]; // Extract token
const decoded = jwt.verify(token, process.env.JWT_SECRET);
req.user = await User.findById(decoded.id).select("-password");
next();
} else {
res.status(401).json({ message: "Not authorized, no token" });
}
} catch (error) {
res.status(401).json({ message: "Token failed", error: error.message });
}
};
```

У цьому middleware токен дістається з заголовка Authorization, перевіряється методом jwt.verify, і, якщо все гаразд, у req.user записується об'єкт поточного користувача [5]. Це дозволяє в подальших контролерах знати, хто саме виконує запит.

Файл uploadMiddleware.js реалізує обробку завантажених зображень через multer. Зокрема, задається місце збереження та дозволені типи файлів:

```
const storage = multer.diskStorage({
destination: (req, file, cb) => {
cb(null, 'uploads/');
},
filename: (req, file, cb) => {
cb(null, `${Date.now()}-${file.originalname}`);
},
});
```

Використовується локальне збереження у папці uploads/, а ім'я кожного

файлу доповнюється часовою міткою, щоб уникнути конфліктів. Фільтрація дозволяє лише .jpeg, .jpg та .png.

Файл User.js містить схему користувача для MongoDB, яка описує структуру документів у колекції users. Основні поля ім'я, email, пароль і опціональна URL-адреса до зображення профілю:

```
const UserSchema = new mongoose.Schema(
{
name: { type: String, required: true },
email: { type: String, required: true, unique: true },
password: { type: String, required: true },
profileImageUrl: { type: String, default: null },
},
{ timestamps: true });
```

Також додаються мітки часу створення й оновлення, що дозволяє відстежувати активність користувача.

Файл Resume.js визначає основну структуру резюме. Це один із найскладніших об'єктів у системі, що включає пов'язане поле userId, заголовок, тему, шаблон та різні секції: профіль, контакти, досвід, освіту, навички, проєкти тощо:

```
const ResumeSchema = new mongoose.Schema(
{
userId: {
type: mongoose.Schema.Types.ObjectId,
ref: "User",
required: true,
},
title: { type: String, required: true },
thumbnailLink: String,
template: {
theme: String,
```

colorPalette: [String]

},

profileInfo: { fullName: String, designation: String, summary: String, profilePreviewUrl: String },

contactInfo: { email: String, phone: String, location: String, linkedin: String, github: String, website: String },

Така структура дозволяє зберігати повну інформацію про резюме користувача у вигляді одного документа, що зручно як для оновлення, так і для рендерингу в інтерфейсі. Ці модулі відповідають за безпеку (authMiddleware), збереження медіа (uploadMiddleware) та модельну частину бази даних (User та Resume) і є фундаментальними для стабільної роботи бекенду системи.

Проаналізуємо клієнтську частину. App.jsx є головним компонентом застосунку, що відповідає за конфігурацію маршрутизації, загальну обгортку контексту користувача і ініціалізацію повідомлень (тостів). Саме звідси запускається React-додаток і визначається логіка переходів між основними сторінками:

```
const App = () => {
 return (
 <UserProvider>
 <div>
 <Router>
 <Routes>
 {/* Default Route */}
 <Route path="/" element={<LandingPage />} />
 <Route path="/dashboard" element={<Dashboard />} />
 <Route path="/resume/:resumeId" element={<EditResume />} />
 </Routes>
 </Router>
 </div>
```

```
<Toaster
toastOptions={{
className: "",
style: {
fontSize: "13px",
},
}}
/>
</UserProvider>
);
};
```

У цьому фрагменті реалізовано три основні маршрути:

- / – головна сторінка (LandingPage);
- /dashboard – персональна панель користувача (Dashboard);
- /resume/:resumeId – сторінка редагування конкретного резюме (EditResume), яка приймає ідентифікатор як параметр.

Обертаючи все у <UserProvider>, система забезпечує доступ до інформації про користувача в будь-якій точці застосунку. Компонент <Toaster> відповідає за відображення повідомлень-сповіщень при помилках, виконанні дій тощо. Цей файл є основною точкою входу у візуальну частину застосунку й визначає структуру навігації.

Важливою частиною клієнтської логіки є реалізація інтерфейсу автентифікації, зокрема вхід та реєстрація користувача. Ці процеси реалізовано у компонентах Login.jsx та SignUp.jsx, які обробляють введення даних, валідацію, взаємодію з API та оновлення глобального контексту користувача після успішної авторизації. У файлі Login.jsx передбачено класичну форму входу, яка запитує email та пароль. Попередньо перевіряються базові умови правильність формату email і заповнення поля пароля. У разі успішного логіну за допомогою axiosInstance надсилається POST-запит на /api/auth/login [1]. Якщо отримано токен, він зберігається в localStorage, а дані користувача у глобальному

контексті UserContext, після чого користувач перенаправляється до /dashboard.

Водночас SignUp.jsx реалізує логіку реєстрації. Окрім стандартних полів, він також дозволяє завантажити аватар користувача через окремий компонент ProfilePhotoSelector. Якщо зображення завантажено, воно спершу передається через окремий запит (використовуючи функцію uploadImage), і лише потім разом з іншими даними відправляється POST-запит на /api/auth/register. Після успішної реєстрації застосовується та сама логіка збереження токена і переходу до панелі керування.

Ключова частина логіки обох форм виглядає подібно:

```
const response = await axiosInstance.post(API_PATHS.AUTH.LOGIN,
{
 email,
 password,
 });
const { token } = response.data;
if (token) {
 localStorage.setItem("token", token);
 updateUser(response.data);
 navigate("/dashboard");
 }
```

Загалом обидва компоненти забезпечують простий, але надійний інтерфейс автентифікації з інформативними помилками, валідацією та обробкою стану.

Компонент Dashboard.jsx відповідає за виведення головної сторінки користувача після входу. У ньому реалізовано логіку завантаження всіх наявних резюме користувача, їх перегляд і створення нового. Ключовим тут є використання React-хука () useEffect, що при завантаженні компонента викликає функцію fetchAllResumes, яка надсилає GET-запит до API (/api/resume) для отримання переліку збережених резюме. Отримані дані зберігаються у стані allResumes і використовуються для динамічного формування сітки карток.

Додатково передбачено спеціальний блок для створення нового резюме, який відкриває модальне вікно з компонентом CreateResumeForm:

```
<div className="h-[300px] flex flex-col ...">
 <LuCirclePlus ... />
 <h3 className="...">Створити резюме</h3>
</div>
```

Це вікно дає змогу створити нове резюме через окрему форму, що має лише одне поле назву резюме. Після подання форми (CreateResumeForm.jsx) відбувається POST-запит на сервер (/api/resume), у відповідь на який повертається ID нового резюме. Далі користувача автоматично перенаправляє на сторінку редагування створеного документа:

```
if (response.data?._id) {
 navigate(`/resume/${response.data._id}`);
}
```

Цей механізм забезпечує швидке створення та подальше редагування резюме без перевантаження інтерфейсу, а також ілюструє взаємодію з бекендом через Axios та навігацію в межах SPA на базі React Router.

Компонент EditResume.jsx реалізує повноцінну сторінку редагування резюме у форматі багатокрокової форми, де кожен етап окрема секція з формою: профіль, контакти, досвід роботи, освіта, навички, проєкти, сертифікати, мови та інтереси.

Перед переходом на наступний крок перевіряється коректність введених даних, що забезпечує чистоту введеної інформації та запобігає неповному заповненню форм. Це реалізується функцією validateAndNext, яка залежно від активної сторінки перевіряє поля відповідної секції. Наприклад, для профілю обов'язковими є ПІБ, посада і коротке резюме:

```
const validateAndNext = () => {
 const errors = [];
 const { fullName, designation, summary } = resumeData.profileInfo;
 if (!fullName.trim()) errors.push("ПІБ є обов'язковим");
```

```
if (!designation.trim()) errors.push("Посада є обов'язковою");
if (!summary.trim()) errors.push("Резюме (короткий опис) є обов'язковим");
if (errors.length > 0) {
setErrorMsg(errors.join(", "));
return;
}
setErrorMsg("");
goToNextStep();
};
```

Праворуч на сторінці одразу відображається шаблон майбутнього резюме через компонент RenderResume, який автоматично оновлюється при зміні будь-якого поля. Це дає змогу користувачеві одразу бачити результат своїх змін. При натисканні кнопки «Зберегти» формується зображення резюме, зберігається фото профілю та оновлені дані:

```
const uploadResumeImages = async () => {
fixTailwindColors(resumeRef.current);
const imageDataUrl = await captureElementAsImage(resumeRef.current);
const thumbnailFile = dataURLtoFile(imageDataUrl, `resume-${resumeId}.png`);
// ...формування formData
await axiosInstance.put(API_PATHS.RESUME.UPLOAD_IMAGES(resumeId), formData);
// ...оновлення основних даних
};
```

Через кнопку «Змінити стиль» відкривається модальне вікно з вибором тем оформлення (колірна палітра, шаблон), що зберігається в resumeData.template. Якщо користувач відкриває сторінку редагування з URL /resume/:resumeId,

система завантажує наявне резюме через fetchResumeDetailsById і заповнює всі поля автоматично. EditResume це центральний багатофункціональний компонент, який відповідає за UX створення та редагування резюме: від валідації й перегляду до збереження й друку. Він забезпечує зручний і контрольований процес, де користувач крок за кроком формує своє професійне подання.

У папці Forms/ містяться всі основні компоненти для редагування конкретних блоків резюме. Кожна форма відповідає за окремий розділ і реалізована як самостійний React-компонент із підтримкою динамічного оновлення даних.

ProfileInfoForm.jsx дозволяє користувачеві ввести або змінити особисту інформацію, включно з ім'ям, посадою, фото профілю та коротким описом. Фото завантажується через спеціальний компонент ProfilePhotoSelector. ContactInfoForm.jsx містить поля для введення контактних даних: адреса, email, телефон, посилання на LinkedIn, GitHub і портфоліо. Усі поля зв'язані зі станом форми, що забезпечує миттєве оновлення. EducationDetailsForm.jsx реалізує можливість додавати кілька записів про освіту. Для кожного з них вказується рівень, заклад та дати навчання. Є підтримка динамічного додавання і видалення записів. WorkExperienceForm.jsx аналогічно до форми освіти дозволяє вказати професійний досвід. Кожен запис включає посаду, компанію, опис обов'язків і період роботи. ProjectsDetailForm.jsx надає змогу додати інформацію про індивідуальні або командні проєкти. Для кожного проєкту вводиться назва, опис і посилання. SkillsInfoForm.jsx дозволяє сформувати перелік навичок із градацією рівня володіння. Використовується візуальний компонент оцінки (рейтинг) для зручного вибору. CertificationInfoForm.jsx відповідає за введення сертифікатів: їх назва, видавник і рік отримання. Можна створити кілька записів за аналогічною схемою. AdditionalInfoForm.jsx охоплює введення мов (із зазначенням рівня володіння) та інтересів. Підтримується візуальний рейтинг мовних навичок, а також можливість видаляти й додавати нові елементи.

Усі форми мають спільну логіку: введення даних відбувається через уніфіковані компоненти (Input, RatingInput тощо), керування масивами через

функції addArrayItem, removeArrayItem, updateArrayItem, а оновлення об'єктів через updateSection. Такий підхід забезпечує гнучкість і масштабованість усієї системи резюме.

У застосунку система шаблонів реалізована через окремий компонент RenderResume, який динамічно підключає один із кількох доступних варіантів оформлення резюме залежно від вибраного templateId. Кожен шаблон – це окремий React-компонент (TemplateOne, TemplateTwo, ..., TemplateSeven), який має власне візуальне подання, але однакову структуру отримання даних.

Механізм відображення шаблону, компонент RenderResume отримує такі пропси:

- templateId ідентифікатор шаблону (наприклад, "01", "02", "03");
- resumeData об'єкт, який містить усю інформацію про резюме (профіль, досвід, навички тощо);
- colorPalette набір кольорів, які змінюють вигляд обраного шаблону;
- containerWidth ширина блоку, яка використовується для масштабування.

На основі templateId здійснюється умовне відображення одного з компонентів-шаблонів:

```
switch (templateId) {
 case "01":
 return <TemplateOne resumeData={...} colorPalette={...} />;
 ...
}
```

Таким чином, усі шаблони динамічно підключаються залежно від вибору користувача, що дозволяє в реальному часі переглядати оформлення майбутнього резюме.

Усередині кожного шаблону (наприклад, TemplateOne) дані з resumeData передаються безпосередньо до внутрішніх секцій, що відповідають за вивід певної інформації. Наприклад:

– resumeData.profileInfo – ім'я, посада, фото, самопрезентація;

– resumeData.contactInfo – адреса, пошта, телефон, соцмережі;

– resumeData.education, resumeData.workExperience, resumeData.skills відповідні масиви об'єктів для динамічного рендерингу секцій.

Рендеринг масивів (освіта, досвід, навички) відбувається через .map(), що дозволяє виводити стільки записів, скільки користувач ввів у формі.

Наприклад:

```
{resumeData.education.map((data, index) => (
<EducationInfo
key={`education_${index}`}
degree={data.degree}
institution={data.institution}
duration={`${formatYearMonth(data.startDate)} – ${formatYearMonth(data.endDate)}`}
/>
))}
```

Кожен шаблон підтримує зміну теми за допомогою властивості colorPalette, яка є масивом із 5 кольорів:

```
const DEFAULT_THEME = ["#EBFDFF", "#A1F4FD", "#CEFAFE", "#00B8DB", "#4A5565"];
```

Ці кольори передаються до компонентів і стилів у шаблоні для формування індивідуального вигляду: фонові блоки, кольори тексту й іконок, акценти (підкреслення заголовків, тіні тощо).

Наприклад:

```
<Title text="Освіта" color={themeColors[1]} />
...
<div className="rounded-lg" style={{ backgroundColor: themeColors[2] }}>
...
</div>
```

Це дозволяє користувачеві персоналізувати зовнішній вигляд свого

резюме, а завдяки інтеграції з редактором (ThemeSelector) нові кольори одразу застосовуються на попередньому перегляді без перезавантаження сторінки. Система шаблонів побудована гнучко: кожен шаблон використовує однаковий набір даних, але відображає його з унікальним стилем і розміткою. Завдяки цьому структура резюме залишається сталою, а візуальна частина змінною, адаптивною та інтерактивною.

3.3. Тестування та демонстрація роботи системи

Під час розробки вебзастосунку CV Master було передбачено тестування інтерфейсу безпосередньо у браузері. Це дозволяє розробникові відразу перевірити коректність відображення компонентів, їхню інтерактивність та відповідність очікуваному функціоналу на різних пристроях. На рис. 3.2 зображено головну сторінку застосунку.

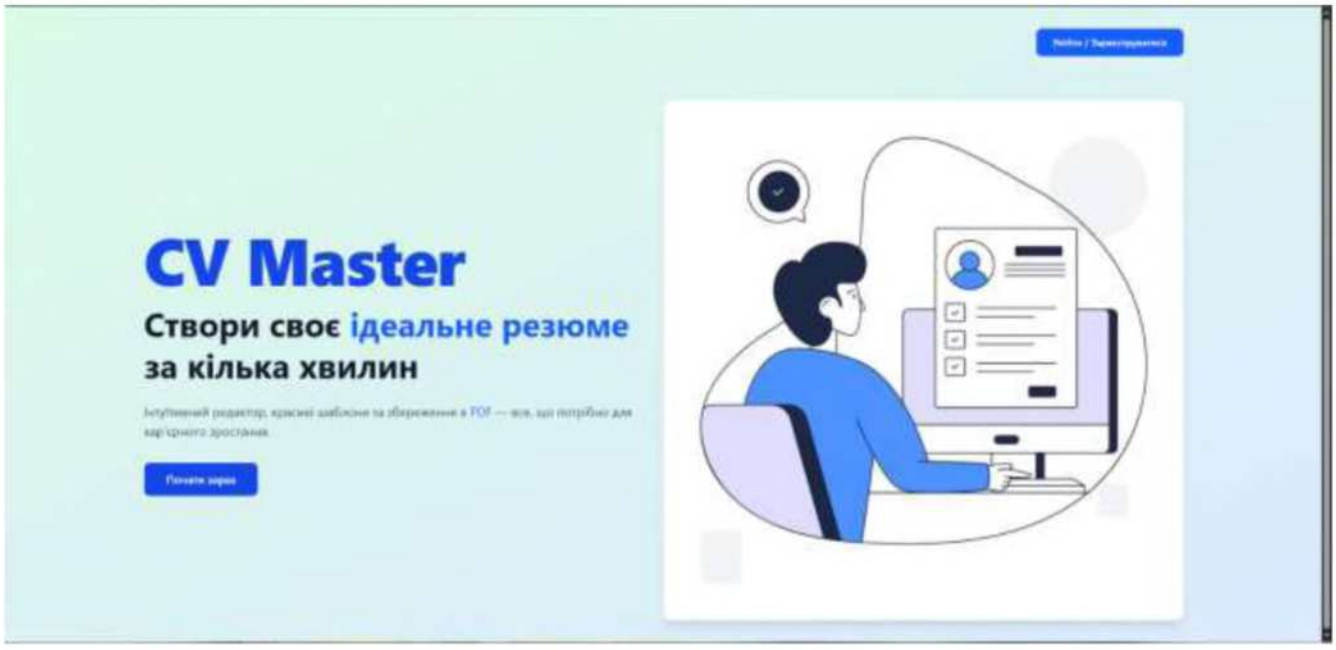


Рис. 3.2 Головна сторінка системи

Вона виконує функцію початкового екрана та слугує вітриною для подання основного функціоналу конструктора. Користувача привертає великий промоційний заголовок, заклик до дії, кнопка «Почати зараз», а також ілюстрація праворуч, яка створює візуальний акцент і задає стиль решти застосунку.

На рис. 3.3 подано модальні вікна для входу та реєстрації.

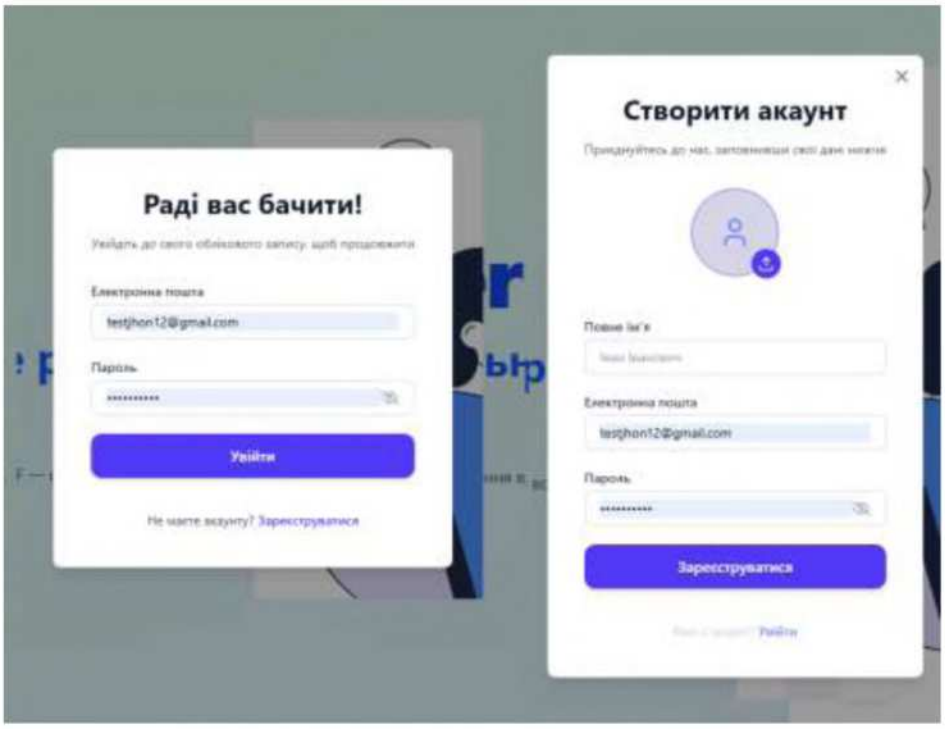


Рис. 3.3 Форми входу та реєстрації

Зліва форма авторизації з полями для електронної пошти та пароля. Праворуч форма реєстрації нового користувача, яка додатково містить поле для повного імені та можливість завантажити аватар. Обидві форми мають сучасний вигляд, адаптивний дизайн та перевірку на коректність введених даних.

Після успішної авторизації або реєстрації користувач автоматично переходить до власної панелі керування, яка подана у вигляді галереї резюме. На екрані користувач бачить всі створені ним резюме, кожне з яких відображається у вигляді окремої картки з мініатюрою шаблону, назвою та датою останнього оновлення. Таке відображення забезпечує швидкий доступ до редагування та перегляду будь-якого збереженого резюме (рис. 3.4).

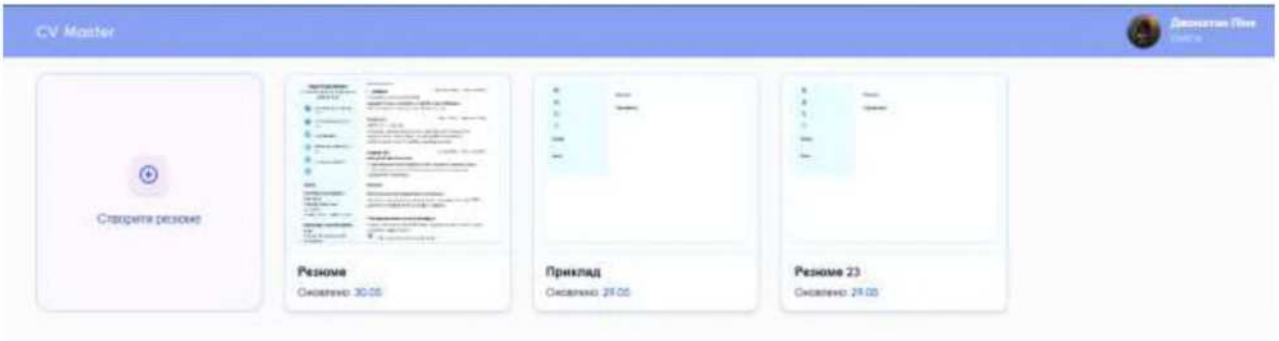


Рис. 3.4 Головна панель користувача

Для створення нового резюме достатньо натиснути на першу картку з позначкою «+». Після цього відкривається модальне вікно з простою формою, в якій користувач вводить назву нового резюме (рис. 3.5).

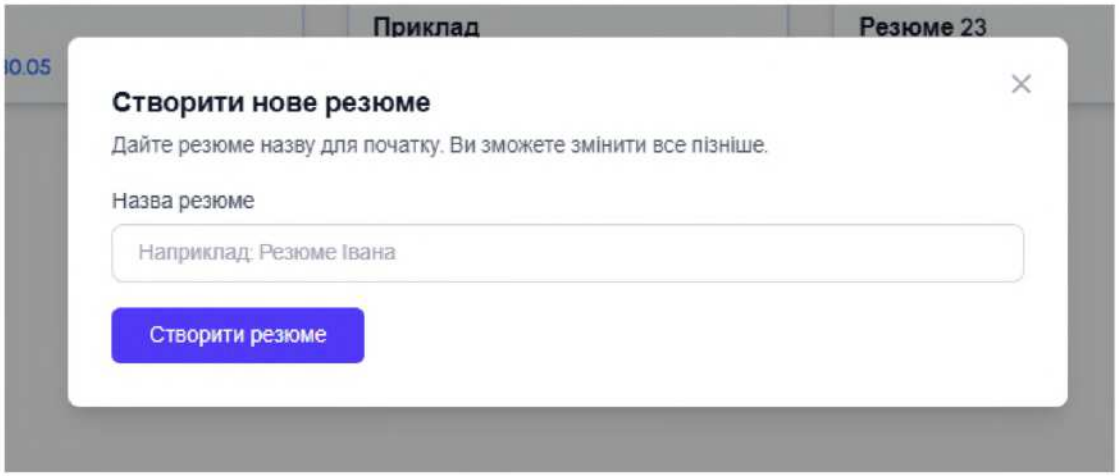


Рис. 3.5 Форма створення назви резюме

Система підказує, що цю назву можна буде змінити пізніше. Після підтвердження створення користувач автоматично перенаправляється до конструктора, де може заповнити структуру резюме.

Після створення нового резюме користувач переходить до інтерфейсу редагування, який поділений на дві основні частини форму заповнення даних (ліва частина) та інтерактивний попередній перегляд шаблону (права частина).

На рис. 3.6 зображена ліва частина, яка містить форму для введення особистої інформації.

Тут користувач заповнює поля «ПІБ», «Посада», «Коротка інформація про себе», а також може завантажити фотографію профілю. Кожне з цих полів має чітку підписану мітку й placeholder. Кнопки знизу дозволяють перейти назад, зберегти та вийти, або перейти до наступного етапу форми.

На рис. 3.7 подано права частина це шаблон резюме, який автоматично оновлюється при кожній зміні в лівій формі. Наприклад, введене ПІБ або завантажене фото одразу відображається в шаблоні. Такий підхід дає можливість бачити, як резюме виглядатиме в реальному часі, що значно полегшує користувачеві процес оформлення.

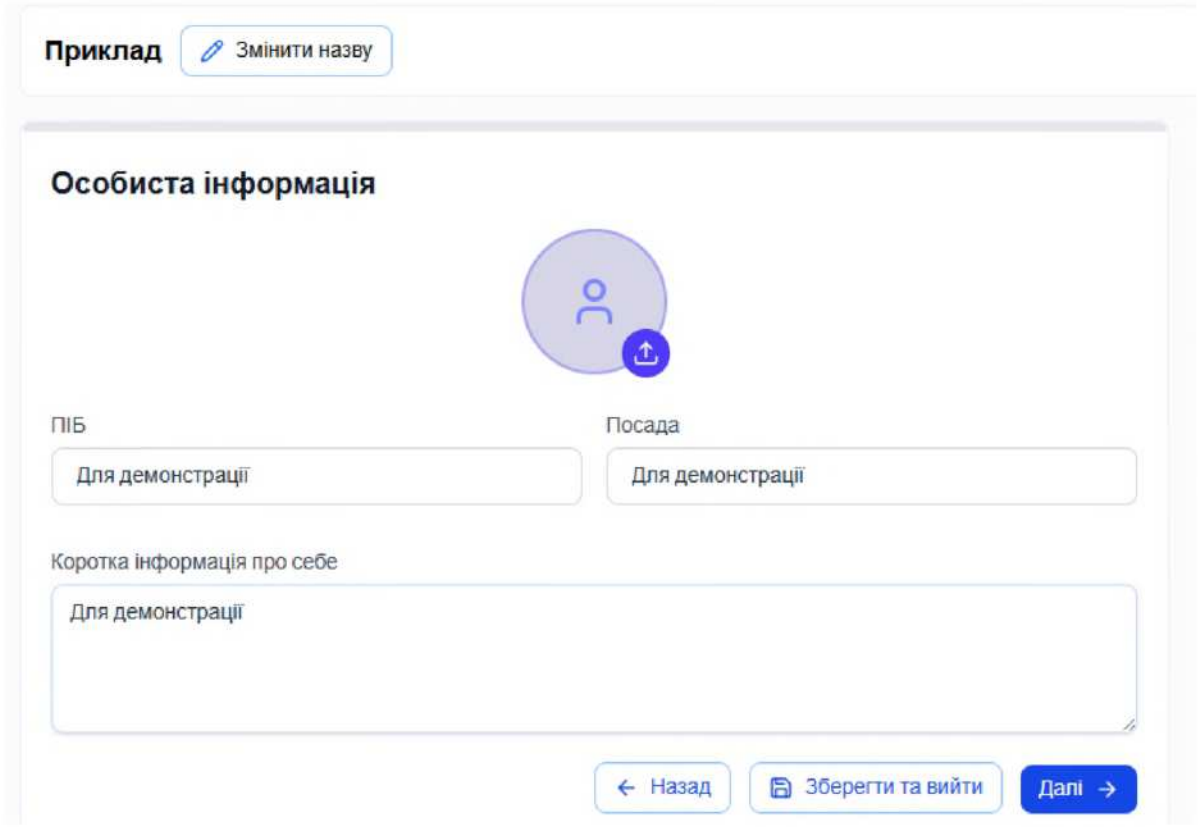


Рис. 3.6 Ліва частина форми створення резюме

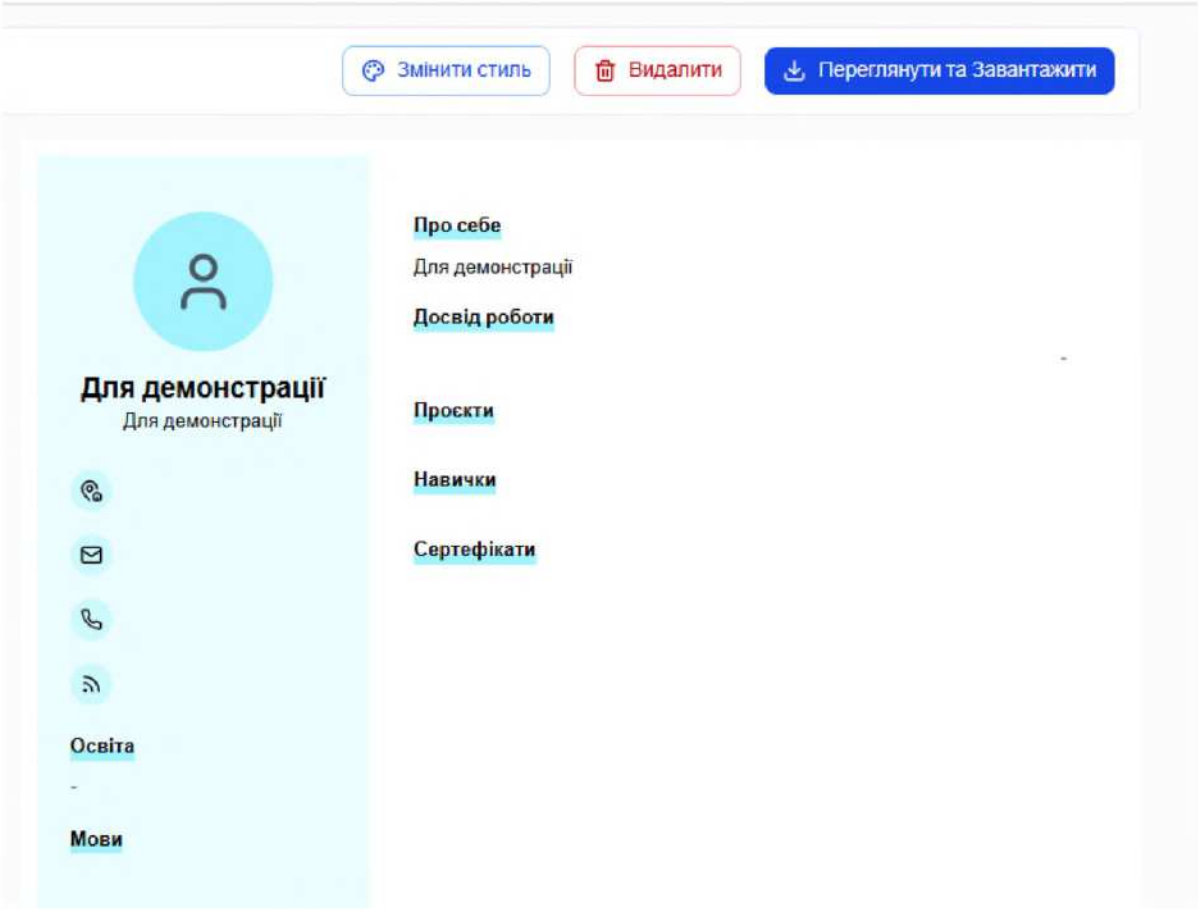


Рис. 3.7 Права частина форми створення резюме

У верхній частині інтерфейсу розташовані кнопки:

- «Змінити стиль» відкриває модальне вікно для вибору шаблону та кольорової палітри;
- «Видалити» дозволяє повністю видалити поточне резюме;
- «Переглянути та Завантажити» дає змогу експортувати резюме у PDF-форматі.

Таким чином, ця частина системи реалізує логіку покрокового заповнення інформації, де кожен крок (форма) відповідає за певну секцію резюме, а динамічний попередній перегляд підвищує зручність та ефективність роботи користувача.

Після заповнення особистих даних користувач переходить до наступного етапу введення професійної інформації. На рис. 3.8 зображена форма додавання досвіду роботи.

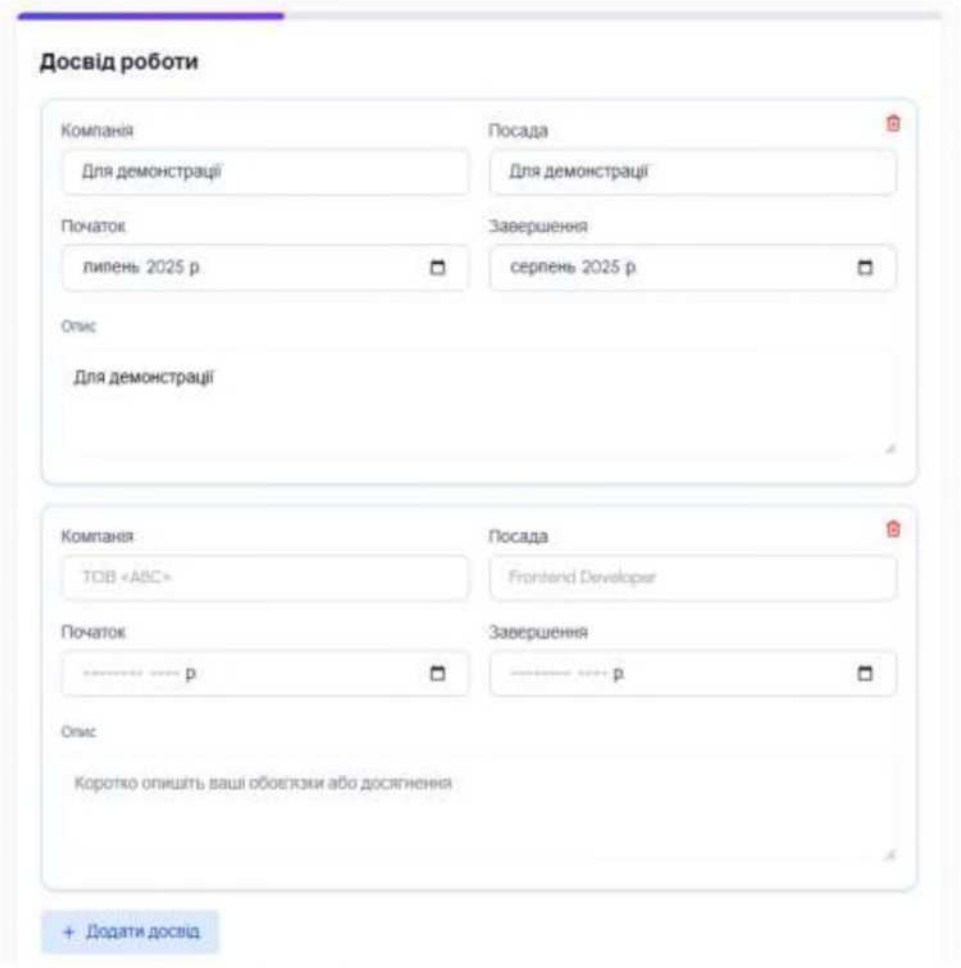


Рис. 3.8 Форма додавання досвіду роботи

Цей розділ дозволяє створити декілька записів про попередні місця праці. Кожен блок містить поля: «Компанія», «Посада», «Початок», «Завершення» та «Опис». Дати обираються за допомогою зручного календаря, а для кожного запису передбачена кнопка видалення. За необхідності користувач може додати новий досвід натисканням на кнопку «+ Додати досвід» у нижній частині блоку. Структура побудована таким чином, щоб забезпечити швидке й інтуїтивно зрозуміле заповнення.

Усі наступні форми освіта, проєкти, сертифікати та інтереси мають подібну побудову: динамічне додавання записів, чіткі назви полів і адаптивний дизайн.

На рис. 3.9 представлено об'єднану інформацію з форм мовними та технічними навичками, тобто скілами ().

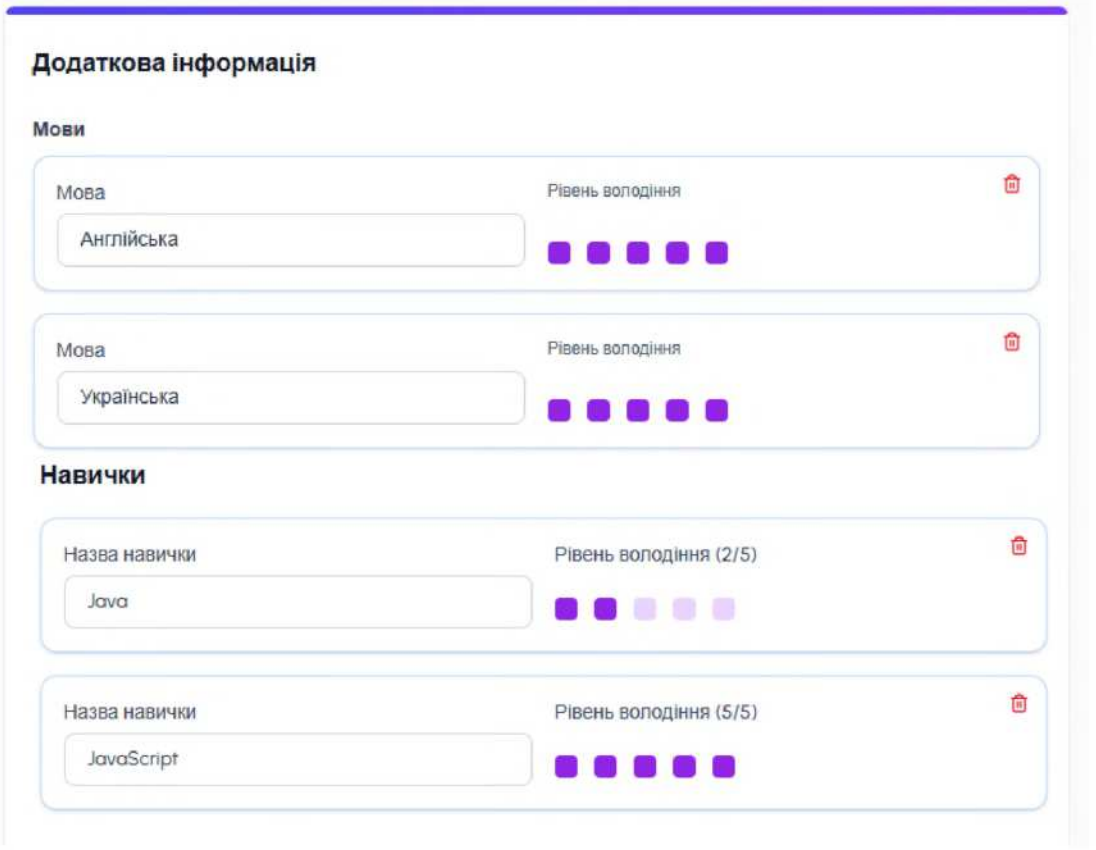


Рис. 3.9 Форма заповнення мов та навичок

У розділі «Мови» користувач обирає мову та вказує рівень володіння за п'ятибальною шкалою. Для кожного рівня використано стильні кольорові

чекбокси, що візуально відображають ступінь володіння. Подібна логіка застосована і до навичок, де можна вписати назву технології або інструменту та оцінити рівень володіння. Такий підхід до заповнення резюме дозволяє зробити процес максимально наочним, інтерактивним і комфортним для користувача, а стильне візуальне оформлення сприяє створенню професійного вигляду підсумкового документа.

У разі, якщо користувач бажає змінити зовнішній вигляд резюме, він може скористатися функцією «Змінити стиль», яка відкриває відповідну форму для налаштувань візуального оформлення. На рис. 3.10 подано інтерфейс вибору шаблону оформлення резюме. Зліва розташований список доступних шаблонів, які користувач може швидко переглянути й вибрати. Після натискання на певний варіант праворуч автоматично оновлюється попередній перегляд, що дозволяє миттєво оцінити візуальні зміни.

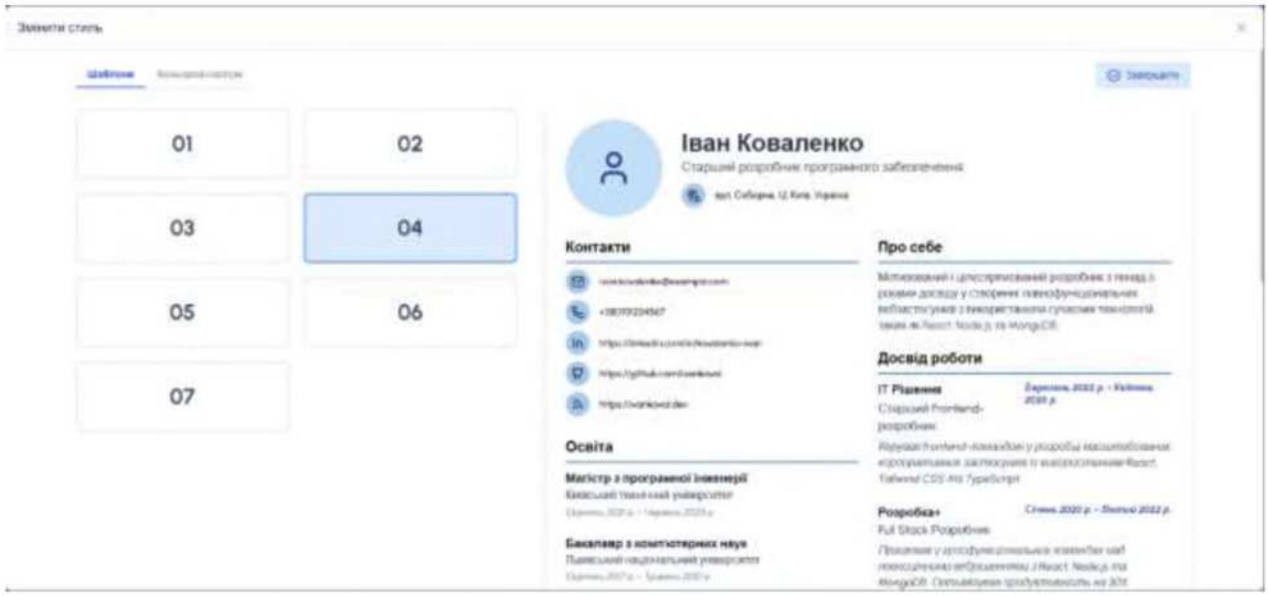


Рис. 3.10 Вибір шаблону оформлення резюме

Таким чином, користувач може підібрати той стиль, який найбільше відповідає його професійній сфері чи особистим вподобанням.

Крім шаблонів, доступний також розділ «Кольорові палітри», що дає змогу налаштувати кольорову гаму резюме (рис. 3.11).

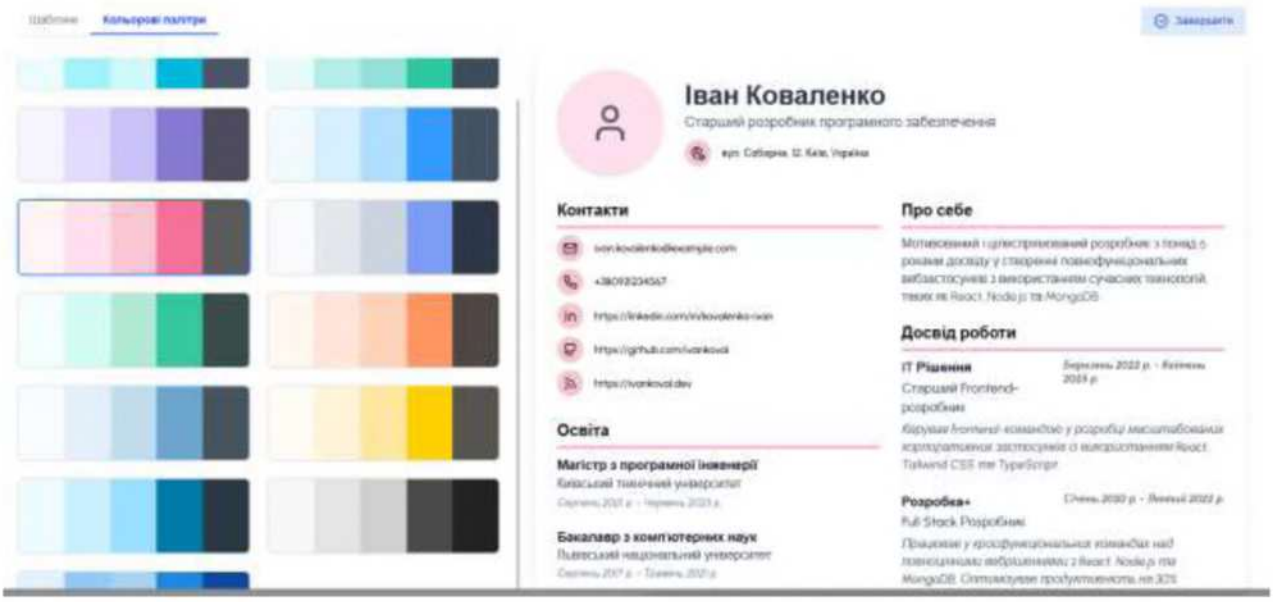


Рис. 3.11 Вибір кольорової палітри для оформлення резюме

Це особливо зручно для тих, хто прагне зробити документ більш виразним або адаптувати його під кольори бренду чи корпоративного стилю.

При виборі палітри змінюється оформлення ключових елементів резюме: заголовків, іконок, роздільників і акцентів. Кольори застосовуються одразу в інтерактивному перегляді, що забезпечує зручність налаштування та дає змогу досягти професійного й естетичного результату без спеціальних навичок дизайну.

Якщо користувач бажає завантажити своє резюме, система пропонує скористатися функцією друку/експорту в PDF. Після натискання кнопки «Переглянути та Завантажити» відкривається стандартна форма друку (рис. 3.12), яка дозволяє зберегти документ у форматі PDF.

Отже, враховуючи результати перевірки всіх ключових функціональних блоків системи від створення та редагування резюме до вибору стилю, кольорового оформлення й експорту у форматі PDF можна зробити висновок, що система працює стабільно та відповідно до заданих вимог. Усі елементи інтерфейсу функціонують коректно, дії користувача обробляються без помилок, а результат виводиться у передбаченому форматі. Таким чином, тестування

системи можна вважати успішним, а розроблений функціонал готовим до використання кінцевими користувачами.

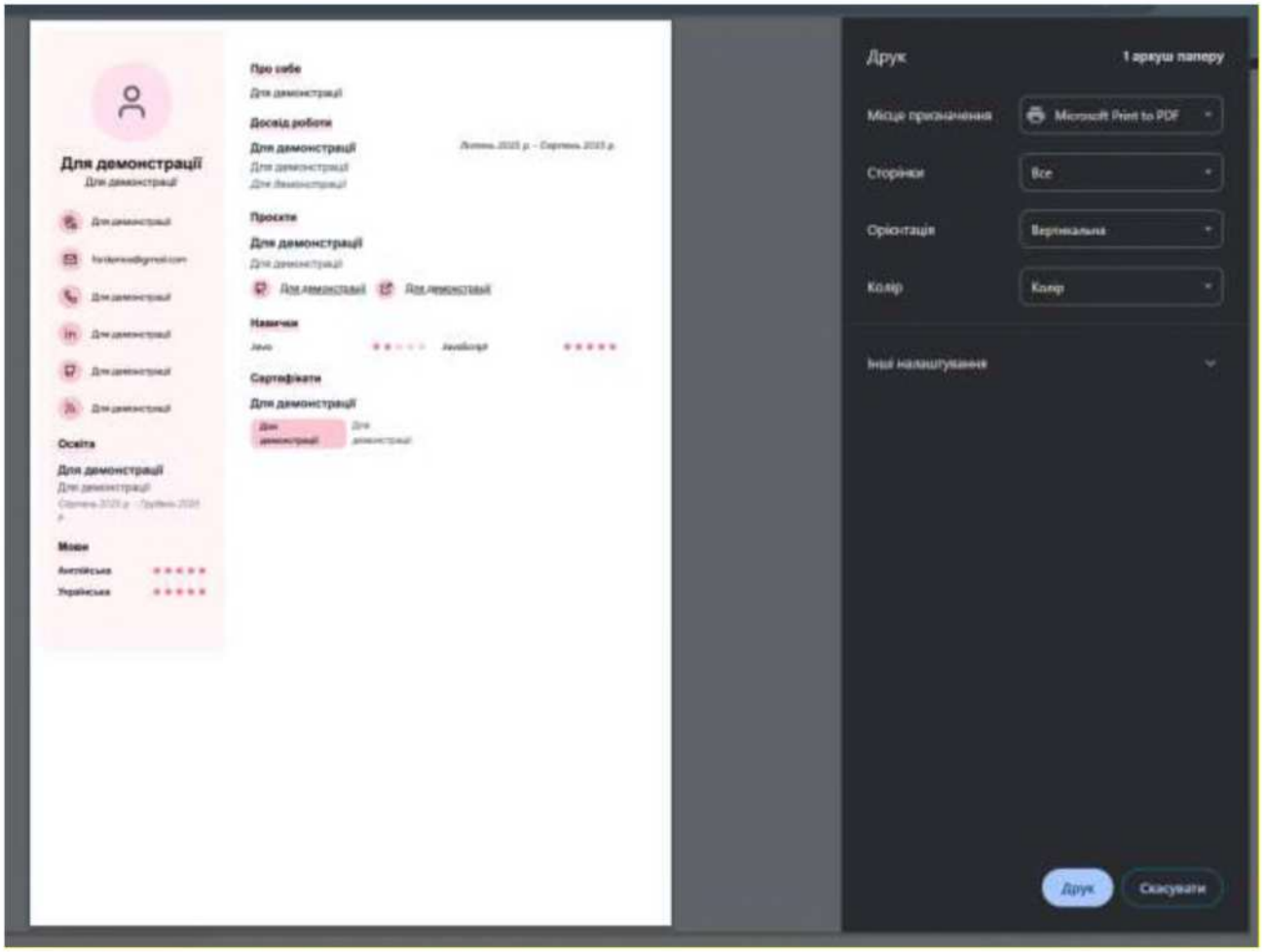


Рис. 3.12 Форма завантаження резюме через друк

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проведено всебічне дослідження сучасних підходів до створення вебдодатків, зокрема систем для формування резюме. Проаналізовано актуальні технології розробки вебсистем, розглянуто приклади наявних аналогів, їхні переваги та недоліки, що дало змогу визначити оптимальні рішення для реалізації власного застосунку. На основі цього було чітко сформульовано мету, завдання та функціональні вимоги до системи.

Під час проєктування детально опрацьовано структуру майбутнього застосунку, зокрема схему організації даних у базі та архітектуру системи. Розроблено логічну модель, яка охоплює основні компоненти: інтерфейс користувача, серверну частину та хмарне збереження даних. Значну увагу приділено інтерактивності, адаптивності та зручності взаємодії з користувачем.

В рамках реалізації створено повноцінну вебсистему конструктора резюме з можливістю реєстрації, редагування особистої інформації, вибору шаблону та кольорової палітри, а також збереження результату у форматі PDF. Описано ключовий функціонал системи, зокрема форми введення даних, попередній перегляд резюме в реальному часі, зміну стилів, структуру бази даних та механізми взаємодії між компонентами. Проведене тестування підтвердило коректність роботи всіх елементів системи та відповідність результатів очікуваним. Усі функції виконуються без помилок, а користувацький досвід є інтуїтивно зрозумілим і зручним.

В результаті реалізовано функціональний і візуально привабливий вебдодаток, що дозволяє створювати якісні резюме без потреби в спеціальних знаннях у сфері дизайну чи верстки. Перспективи подальшого вдосконалення полягають у розширенні бібліотеки шаблонів, додаванні підтримки декількох мов, генерації супровідного листа, а також у створенні мобільної версії для зручності доступу з різних пристроїв.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Axios Documentation. Axios HTTP client – Instance API. URL: https://axios-http.com/docs/instance (дата звернення: 18.01.2026).

2. Axios Documentation. Axios Project. URL: https://axios-http.com/

3. bcrypt npm package. URL: https://www.npmjs.com/package/bcrypt

4. Brits are unlocking their career potential with CV builder Resume.io. Resume.io. URL: https://www.thescottishsun.co.uk/money/12126876/unlock-career-potential-with-resume-cv-builder/

5. Coders Kamrul. React authentication with JWT. *Medium*. 2023. URL: https://medium.com/@coderskamrul/react-authentication-with-jwt-30d57dc4cd6f (дата звернення: 18.01.2026).

6. CORS (Cross-Origin Resource Sharing). MDN Web Docs. URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Guides/CORS (дата звернення: 18.01.2026).

7. CSS: Cascading Style Sheets – MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS

8. dotenv npm package. URL: https://www.npmjs.com/package/dotenv

9. EntekuM Jeffrey. Middleware in Next.js and React: the smarter way to handle protected routes. *Medium*. 2023. URL: https://medium.com/@entekumejeffrey/middleware-in-next-js-and-react-the-smarter-way-to-handle-protected-routes-ec7a966ead9d (дата звернення: 18.01.2026).

10. Express – Node.js web application framework. URL: https://expressjs.com/

11. Formik + Yup Documentation. URL: https://formik.org/docs/overview

12. GOIT Global. MongoDB — що це таке та як працює. URL: https://goit.global/ua/articles/mongodb-shcho-tse-take-ta-yak-prazuie/

13. Gopika R., Kumar A. A smart resume builder: enhancing career opportunities with technology – International Research Journal of Education and Technology, 2025. – Т. 8, № 4. – С. 67–75. – URL:

https://www.irjweb.com/A%20Smart%20Resume%20Builder%20Enhancing%20Career%20Opportunities%20with%20Technology.pdf

14. HTML: HyperText Markup Language – MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTML

15. IBM. What are JSON Web Tokens (JWT)? *IBM Think*. URL: https://www.ibm.com/think/topics/json-web-tokens

16. JavaScript – MDN Web Docs – Mozilla. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript

17. JWT Introduction. Auth0. URL: https://auth0.com/intro-to-jwt/

18. Kickresume Review. 4 Day Week. URL: https://4dayweek.io/blog/kickresume-review

19. Kickresume. Top 10 Resume Builders of 2025: We Tried Them All so You Don't Have to. URL: https://www.kickresume.com/en/help-center/10-best-resume-builders/

20. MongoDB Atlas | The Modern, Multi-Cloud Database. URL: https://www.mongodb.com/atlas

21. MongoDB Manual. MongoDB. URL: https://www.mongodb.com/docs/manual/

22. MongoDB Overview. MongoDB Documentation. URL: https://www.mongodb.com/docs/manual/overview/

23. Mongoose ODM for MongoDB. URL: https://mongoosejs.com/

24. Node.js Documentation. OpenJS Foundation. URL: https://nodejs.org/en/docs

25. React – A JavaScript library for building user interfaces. Meta. URL: https://react.dev/

26. React – The library for web and native user interfaces. URL: https://react.dev/

27. React Router Documentation. Remix Software. URL: https://reactrouter.com/en/main

28. Redux Toolkit Documentation. Redux. URL: https://redux-toolkit.js.org/

29. Sudarsanam M., Ajith J., Annamalai J., Darrel P. Automated resume builder for fresh graduates using HTML and JavaScript – International Journal of Advanced Research in Science Communication and Technology, 2022. – URL: https://www.researchgate.net/publication/360749588_Automated_Resume_Builder_for_Fresh_-_Graduates_Using_Html_and_JavaScript

30. Tailwind CSS Documentation. Tailwind Labs. URL: https://tailwindcss.com/

31. The 10 Best Resume Builders in 2025: Pros & Cons. Resume Genius. URL: https://resumegenius.com/blog/resume-help/best-resume-builder-website

32. Top 10 Web Development Trends [2025] – GeeksforGeeks. URL: https://www.geeksforgeeks.org/top-web-development-trends/

33. Visual Studio Code – Code Editing. Redefined. URL: https://code.visualstudio.com/

34. Web Development Technologies Evolution | Modern Trends & Impact. QuickStart. URL: https://www.quickstart.com/blog/app-development/web-development-technologies-evolution/

35. Ковальський В.А., Романюк О.В. Розробка вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/48611/24036.pdf?sequence=3&isAllowed=y

36. Панченко Т., Тузова І., Тузов О., Чумак О. Хмарні сервіси та огляд їх постачальників. *InterConf.* Березень 2024. С. 554–556. DOI: https://doi.org/10.51582/interconf.19-20.03.2024.053.

37. Підько А. Інтерактивні вебзастосунки як інструмент персонального брендингу // Економіка і бізнес: нові реалії, технологічні тренди, цифрові інновації. Economy and business: new realities, technological trends, digital innovations : І Міжн. наук.-практ. конф. здобув. вищої освіти та молод. вчених, Львів-Ірпінь, 15 трав. 2025 р. : тези доп. – Львів: ЛНУ ім. Івана Франка. – С. 586-588.

38. Ткаченко О., Ткаченко К., Піддубченко М. Аналіз сучасних

тенденцій розробки інформаційних систем. *Кібербезпека: освіта, наука, техніка*. 2024. Том 4, № 24. С. 205–220. URL: csecurity.kubg.edu.ua/index.php/journal/article/view/595

39. Функціональні та нефункціональні вимоги: повний гід. Visure Solutions. URL: https://visuresolutions.com/alm-guide/functional-vs-non-functional-requirements/

40. Чернишев Н.С., Бондар Н.В., Чістяков Д.С., Швець В.В., Романюк О.В. Аналіз технологій для створення сучасних веб-застосунків. URL: ir.lib.vntu.edu.ua/bitstream/handle/123456789/44369/18955.pdf?isAllowed=y&sequence=3

ДОДАТКИ

Додаток А

Лістинг програмного коду
А.1. Серверна частина (Backend)

Файл: backend/server.js
JavaScript

```
require("dotenv").config();
const express = require("express");
const cors = require("cors");
const path = require("path");
const connectDB = require('./config/db')
const authRoutes = require('./routes/authRoutes')
const resumeRoutes = require('./routes/resumeRoutes')

const app = express();

app.use(cors({
  origin: process.env.CLIENT_URL || "*",
  methods: ["GET", "POST", "PUT", "DELETE"],
  allowedHeaders: ["Content-Type", "Authorization"],
}));

connectDB();
app.use(express.json());

app.use("/api/auth", authRoutes);
app.use("/api/resume", resumeRoutes);

app.use("/uploads", express.static(path.join(__dirname, "uploads")));

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Файл: backend/config/db.js
(Підключення до бази даних)
JavaScript

```
const mongoose = require("mongoose");

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI, {});
    console.log("MongoDB connected");
  } catch (err) {
    console.error("Error connecting to MongoDB", err);
    process.exit(1);
  }
};
module.exports = connectDB;
```

Файл: backend/models/User.js
(Модель користувача)
JavaScript

```
const mongoose = require("mongoose");
```

```
const UserSchema = new mongoose.Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    profileImageUrl: { type: String, default: null },
  },
  { timestamps: true }
);
module.exports = mongoose.model("User", UserSchema);
```

Файл: backend/models/Resume.js
(Модель резюме - основна структура даних)
JavaScript

```
const mongoose = require("mongoose");

const ResumeSchema = new mongoose.Schema(
  {
    userId: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
    title: { type: String, required: true },
    thumbnailLink: { type: String },
    template: { theme: String, colorPalette: [String] },
    profileInfo: {
      profilePreviewUrl: String,
      fullName: String,
      designation: String,
      summary: String,
    },
    contactInfo: {
      email: String,
      phone: String,
      location: String,
      linkedin: String,
      github: String,
      website: String,
    },
    workExperience: [{
      company: String,
      role: String,
      startDate: String,
      endDate: String,
      description: String,
    }],
    education: [{
      degree: String,
      institution: String,
      startDate: String,
      endDate: String,
    }],
    skills: [{ name: String, progress: Number }],
    projects: [{
      title: String,
      description: String,
      github: String,
      liveDemo: String,
    }],
    certifications: [{ title: String, issuer: String, year: String }],
    languages: [{ name: String, progress: Number }],
    interests: [String],
  },
```

```
  { timestamps: true }
);
module.exports = mongoose.model("Resume", ResumeSchema);
```

Файл: backend/controllers/authController.js
(Логіка авторизації)
JavaScript

```
const User = require("../models/User");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");

const generateToken = (userId) => {
  return jwt.sign({ id: userId }, process.env.JWT_SECRET, { expiresIn: "7d" });
};

const registerUser = async (req, res) => {
  try {
    const { name, email, password, profileImageUrl } = req.body;
    const userExists = await User.findOne({ email });
    if (userExists) return res.status(400).json({ message: "User already exists" });

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    const user = await User.create({ name, email, password: hashedPassword,
profileImageUrl });

    res.status(201).json({
      _id: user._id,
      name: user.name,
      email: user.email,
      profileImageUrl: user.profileImageUrl,
      token: generateToken(user._id),
    });
  } catch (error) {
    res.status(500).json({ message: "Server error", error: error.message });
  }
};

const loginUser = async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) return res.status(500).json({ message: "Invalid email or password" });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(500).json({ message: "Invalid email or password"
});

    res.json({
      _id: user._id,
      name: user.name,
      email: user.email,
      profileImageUrl: user.profileImageUrl,
      token: generateToken(user._id),
    });
  } catch (error) {
    res.status(500).json({ message: "Server error", error: error.message });
  }
};
```

```
module.exports = { registerUser, loginUser };
```

Файл: backend/controllers/resumeController.js
(Логіка управління резюме)
JavaScript

```
const Resume = require("../models/Resume");

const createResume = async (req, res) => {
  try {
    const { title } = req.body;
    const defaultResumeData = {
      profileInfo: { fullName: "", designation: "", summary: "" },
      contactInfo: { email: "", phone: "" },
      workExperience: [],
      education: [],
      skills: [],
      projects: [],
      certifications: [],
      languages: [],
      interests: [""],
    };

    const newResume = await Resume.create({
      userId: req.user._id,
      title,
      ...defaultResumeData,
    });
    res.status(201).json(newResume);
  } catch (error) {
    res.status(500).json({ message: "Failed to create resume", error: error.message });
  }
};

const getUserResumes = async (req, res) => {
  try {
    const resumes = await Resume.find({ userId: req.user._id }).sort({ updatedAt: -1
});
    res.json(resumes);
  } catch (error) {
    res.status(500).json({ message: "Failed to fetch resumes" });
  }
};

const getResumeById = async (req, res) => {
  try {
    const resume = await Resume.findOne({ _id: req.params.id, userId: req.user._id });
    if (!resume) return res.status(404).json({ message: "Resume not found" });
    res.json(resume);
  } catch (error) {
    res.status(500).json({ message: "Failed to fetch resume" });
  }
};

const updateResume = async (req, res) => {
  try {
    const resume = await Resume.findOne({ _id: req.params.id, userId: req.user._id });
    if (!resume) return res.status(404).json({ message: "Resume not found" });

    Object.assign(resume, req.body);
```

```
    const savedResume = await resume.save();
    res.json(savedResume);
  } catch (error) {
    res.status(500).json({ message: "Update failed", error: error.message });
  }
};

const deleteResume = async (req, res) => {
  try {
    const deleted = await Resume.findOneAndDelete({ _id: req.params.id, userId:
req.user._id });
    if (!deleted) return res.status(404).json({ message: "Resume not found" });
    res.json({ message: "Resume deleted successfully" });
  } catch (error) {
    res.status(500).json({ message: "Delete failed", error: error.message });
  }
};

module.exports = { createResume, getUserResumes, getResumeById, updateResume,
deleteResume };
```

A.2. Клієнтська частина (Frontend)

Файл: frontend/src/utils/axiosInstance.js
(Налаштування мережевих запитів)

JavaScript

```
import axios from "axios";
import { BASE_URL } from "./apiPaths";

const axiosInstance = axios.create({
  baseURL: BASE_URL,
  timeout: 10000,
  headers: { "Content-Type": "application/json", Accept: "application/json" },
});

axiosInstance.interceptors.request.use(
  (config) => {
    const accessToken = localStorage.getItem("token");
    if (accessToken) config.headers.Authorization = `Bearer ${accessToken}`;
    return config;
  },
  (error) => Promise.reject(error)
);

axiosInstance.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response && error.response.status === 401) {
      window.location.href = "/";
    }
    return Promise.reject(error);
  }
);

export default axiosInstance;
```

Файл: frontend/src/context/userContext.jsx
(Глобальний стейт користувача)

JavaScript

```
import React, { createContext, useState, useEffect } from "react";
import axiosInstance from "../utils/axiosInstance";
import { API_PATHS } from "../utils/apiPaths";

export const UserContext = createContext();

const UserProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchUser = async () => {
      const token = localStorage.getItem("token");
      if (!token) return setLoading(false);
      try {
        const response = await axiosInstance.get(API_PATHS.AUTH.GET_PROFILE);
        setUser(response.data);
      } catch (error) {
        clearUser();
      } finally { setLoading(false); }
    };
    fetchUser();
  }, []);

  const updateUser = (userData) => {
    setUser(userData);
    localStorage.setItem("token", userData.token);
  };

  const clearUser = () => {
    setUser(null);
    localStorage.removeItem("token");
  };

  return (
    <UserContext.Provider value={{ user, loading, updateUser, clearUser }}>
      {children}
    </UserContext.Provider>
  );
};
export default UserProvider;
```

Файл: frontend/src/pages/Auth/Login.jsx
(Сторінка входу)
JavaScript

```
import React, { useContext, useState } from "react";
import { useNavigate } from "react-router-dom";
import Input from "../../components/Inputs/Input";
import { UserContext } from "../../context/userContext";
import axiosInstance from "../../utils/axiosInstance";
import { API_PATHS } from "../../utils/apiPaths";

const Login = ({ setCurrentPage }) => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const { updateUser } = useContext(UserContext);
  const navigate = useNavigate();
```

```
  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      const response = await axiosInstance.post(API_PATHS.AUTH.LOGIN, { email, password
});
      if (response.data.token) {
        updateUser(response.data);
        navigate("/dashboard");
      }
    } catch (error) {
      console.error("Login failed");
    }
  };

  return (
    <div className="w-full max-w-md bg-white p-10 rounded-2xl shadow-2xl">
      <h3 className="text-3xl font-bold text-center mb-4">Вхід</h3>
      <form onSubmit={handleLogin} className="flex flex-col gap-5">
        <Input value={email} onChange={(e) => setEmail(e.target.value)} label="Email"
type="text" />
        <Input value={password} onChange={(e) => setPassword(e.target.value)}
label="Пароль" type="password" />
        <button type="submit" className="bg-indigo-600 text-white py-3 rounded-
xl">Увійти</button>
      </form>
    </div>
  );
};
export default Login;
```

Файл: frontend/src/pages/Home/Dashboard.jsx
(Головна панель користувача)
JavaScript

```
import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import axiosInstance from "../../utils/axiosInstance";
import { API_PATHS } from "../../utils/apiPaths";
import DashboardLayout from "../../components/layouts/DashboardLayout";
import ResumeSummaryCard from "../../components/Cards/ResumeSummaryCard";
import CreateResumeForm from "./CreateResumeForm";
import Modal from "../../components/Modal";

const Dashboard = () => {
  const navigate = useNavigate();
  const [openCreateModal, setOpenCreateModal] = useState(false);
  const [allResumes, setAllResumes] = useState(null);

  useEffect(() => {
    const fetchAllResumes = async () => {
      try {
        const response = await axiosInstance.get(API_PATHS.RESUME.GET_ALL);
        setAllResumes(response.data);
      } catch (error) { console.error(error); }
    };
    fetchAllResumes();
  }, []);

  return (
    <DashboardLayout>
```

```
      <div className="grid grid-cols-1 md:grid-cols-5 gap-7">
        <div className="h-[300px] flex flex-col gap-5 items-center justify-center bg-
purple-50 rounded-2xl cursor-pointer"
             onClick={() => setOpenCreateModal(true)}>
          <h3>Створити резюме</h3>
        </div>
        {allResumes?.map((resume) => (
          <ResumeSummaryCard
            key={resume._id}
            title={resume.title}
            onSelect={() => navigate(`/resume/${resume._id}`)}
          />
        ))}
      </div>
      <Modal isOpen={openCreateModal} onClose={() => setOpenCreateModal(false)}>
        <CreateResumeForm />
      </Modal>
    </DashboardLayout>
  );
};
export default Dashboard;
```

Файл: frontend/src/pages/ResumeUpdate/EditResume.jsx
(Головний редактор резюме - логіка перемикання кроків)
JavaScript

```
import React, { useEffect, useRef, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import axiosInstance from "../../utils/axiosInstance";
import { API_PATHS } from "../../utils/apiPaths";
import ProfileInfoForm from "./Forms/ProfileInfoForm";
import WorkExperienceForm from "./Forms/WorkExperienceForm";
import RenderResume from "../../components/ResumeTemplates/RenderResume";

const EditResume = () => {
  const { resumeId } = useParams();
  const [resumeData, setResumeData] = useState(null);
  const [currentPage, setCurrentPage] = useState("profile-info");

  useEffect(() => {
    const fetchResume = async () => {
      const response = await axiosInstance.get(API_PATHS.RESUME.GET_BY_ID(resumeId));
      setResumeData(response.data);
    };
    fetchResume();
  }, [resumeId]);

  const updateSection = (section, key, value) => {
    setResumeData((prev) => ({
      ...prev,
      [section]: { ...prev[section], [key]: value },
    }));
  };

  const renderForm = () => {
    switch (currentPage) {
      case "profile-info":
        return <ProfileInfoForm profileData={resumeData?.profileInfo}
updateSection={updateSection} />;
      case "work-experience":
        return <WorkExperienceForm workExperience={resumeData?.workExperience} />;
```

```
      default: return null;
    }
  };

  return (
    <div className="grid grid-cols-2 gap-5">
      <div className="editor-panel">{renderForm()}</div>
      <div className="preview-panel">
        <RenderResume resumeData={resumeData} />
      </div>
    </div>
  );
};
export default EditResume;
```

Файл: frontend/src/pages/ResumeUpdate/Forms/ProfileInfoForm.jsx
(Форма редагування профілю)
JavaScript

```
import React from 'react';
import Input from "../../../components/Inputs/Input";

const ProfileInfoForm = ({ profileData, updateSection }) => {
  return (
    <div className="px-5 pt-5">
      <h2 className="text-xl font-bold">Особиста інформація</h2>
      <div className="grid grid-cols-2 gap-4 mt-4">
        <Input
          value={profileData?.fullName || ""}
          onChange={(e) => updateSection("fullName", e.target.value)}
          label="ПІБ"
        />
        <Input
          value={profileData?.designation || ""}
          onChange={(e) => updateSection("designation", e.target.value)}
          label="Посада"
        />
        <div className="col-span-2">
          <textarea
            className="w-full border p-2 rounded"
            rows={4}
            value={profileData?.summary || ""}
            onChange={(e) => updateSection("summary", e.target.value)}
          />
        </div>
      </div>
    </div>
  );
};
export default ProfileInfoForm;
```

Файл: frontend/src/pages/ResumeUpdate/Forms/WorkExperienceForm.jsx
(Форма редагування досвіду роботи - динамічні поля)

JavaScript

```
import React from 'react';
import Input from "../../../components/Inputs/Input";

const WorkExperienceForm = ({ workExperience, updateArrayItem, addArrayItem,
```

```
removeArrayItem }) => {
  return (
    <div className="px-5 pt-5">
      <h2 className="text-lg font-semibold">Досвід роботи</h2>
      <div className="mt-4 flex flex-col gap-4">
        {workExperience.map((exp, index) => (
          <div key={index} className="border p-4 rounded bg-white">
            <div className="grid grid-cols-2 gap-4">
              <Input
                label="Компанія"
                value={exp.company}
                onChange={(e) => updateArrayItem(index, "company", e.target.value)}
              />
              <Input
                label="Посада"
                value={exp.role}
                onChange={(e) => updateArrayItem(index, "role", e.target.value)}
              />
              <Input
                type="month"
                label="Початок"
                value={exp.startDate}
                onChange={(e) => updateArrayItem(index, "startDate", e.target.value)}
              />
              <Input
                type="month"
                label="Завершення"
                value={exp.endDate}
                onChange={(e) => updateArrayItem(index, "endDate", e.target.value)}
              />
            </div>
            <button onClick={() => removeArrayItem(index)} className="text-red-500 mt-
2">Видалити</button>
          </div>
        ))}
        <button onClick={() => addArrayItem({ company: "", role: "" })} className="bg-
blue-100 text-blue-800 px-4 py-2 rounded">
          + Додати досвід
        </button>
      </div>
    </div>
  );
};
export default WorkExperienceForm;
```

Файл: frontend/src/components/ResumeTemplates/TemplateOne.jsx
(Приклад шаблону резюме для генерації PDF)
JavaScript

```
import React from "react";
import { formatYearMonth } from "../../utils/helper";

const TemplateOne = ({ resumeData, colorPalette }) => {
  const themeColors = colorPalette || ["#EBFDFF", "#00B8DB", "#4A5565"];

  return (
    <div className="p-5 bg-white min-h-[1000px]">
      <div className="grid grid-cols-12 gap-8">
        <div className="col-span-4 py-10" style={{ backgroundColor: themeColors[0] }}>
          <div className="flex flex-col items-center">
            <h2 className="text-xl font-bold">{resumeData.profileInfo.fullName}</h2>
```

```
            <p className="text-sm">{resumeData.profileInfo.designation}</p>
          </div>
          <div className="px-4 mt-6">
            <p>Email: {resumeData.contactInfo.email}</p>
            <p>Phone: {resumeData.contactInfo.phone}</p>
            <h3 className="mt-4 font-bold">Навички</h3>
            {resumeData.skills.map((skill, i) => (
              <div key={i} className="flex justify-between">
                <span>{skill.name}</span>
                <span>{skill.progress}%</span>
              </div>
            ))}
          </div>
        </div>

        <div className="col-span-8 py-10">
          <h3 className="text-lg font-bold border-b-2">Про себе</h3>
          <p className="mt-2">{resumeData.profileInfo.summary}</p>

          <h3 className="text-lg font-bold border-b-2 mt-6">Досвід роботи</h3>
          {resumeData.workExperience.map((job, i) => (
            <div key={i} className="mt-4">
              <h4 className="font-bold">{job.company}</h4>
              <p className="italic">{job.role}</p>
              <p className="text-sm text-gray-500">
                {formatYearMonth(job.startDate)} - {formatYearMonth(job.endDate)}
              </p>
              <p>{job.description}</p>
            </div>
          ))}
        </div>
      </div>
    </div>
  );
};
export default TemplateOne;
```

Файл: frontend/src/components/ResumeTemplates/TemplateTwo.jsx
(Другий варіант шаблону)
JavaScript

```
import React from "react";
import { formatYearMonth } from "../../utils/helper";

const TemplateTwo = ({ resumeData, colorPalette }) => {
  const themeColors = colorPalette || ["#fff", "#333", "#f0f0f0"];

  return (
    <div className="p-8 bg-white">
      <div className="border-b-2 pb-4 mb-4">
        <h1 className="text-4xl font-bold text-gray-
800">{resumeData.profileInfo.fullName}</h1>
        <h2 className="text-xl text-blue-600">{resumeData.profileInfo.designation}</h2>
        <div className="flex gap-4 text-sm text-gray-500 mt-2">
          <span>{resumeData.contactInfo.email}</span>
          <span>{resumeData.contactInfo.phone}</span>
          <span>{resumeData.contactInfo.location}</span>
        </div>
      </div>

      <div className="grid grid-cols-3 gap-6">
```

```
        <div className="col-span-2">
          <h3 className="text-lg font-bold text-blue-700 uppercase">Досвід</h3>
          {resumeData.workExperience.map((exp, i) => (
            <div key={i} className="mb-4">
              <div className="flex justify-between">
                <h4 className="font-bold">{exp.company}</h4>
                <span className="text-sm text-gray-500">
                  {formatYearMonth(exp.startDate)} - {formatYearMonth(exp.endDate)}
                </span>
              </div>
              <p className="italic">{exp.role}</p>
              <p className="text-sm mt-1">{exp.description}</p>
            </div>
          ))}
        </div>

        <div className="col-span-1 bg-gray-50 p-4 rounded">
          <h3 className="text-lg font-bold text-blue-700 uppercase">Освіта</h3>
          {resumeData.education.map((edu, i) => (
            <div key={i} className="mb-4">
              <p className="font-bold">{edu.institution}</p>
              <p className="text-sm">{edu.degree}</p>
            </div>
          ))}

          <h3 className="text-lg font-bold text-blue-700 uppercase mt-6">Навички</h3>
          <div className="flex flex-wrap gap-2">
            {resumeData.skills.map((skill, i) => (
              <span key={i} className="bg-blue-100 text-blue-800 px-2 py-1 rounded
text-xs">
                {skill.name}
              </span>
            ))}
          </div>
        </div>
      </div>
    </div>
  );
};
export default TemplateTwo;
```

Файл: frontend/src/utils/helper.js
(Допоміжні функції)
JavaScript

```
import html2canvas from "html2canvas";

export const validateEmail = (email) => {
  const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return regex.test(email);
};

export const formatYearMonth = (dateStr) => {
  if (!dateStr) return "";
  const date = new Date(dateStr);
  return date.toLocaleDateString("uk-UA", { year: "numeric", month: "long" });
};

export async function captureElementAsImage(element) {
  if (!element) throw new Error("No element provided");
```

```
  const canvas = await html2canvas(element);
  return canvas.toDataURL("image/png");
}

export const dataURLtoFile = (dataUrl, fileName) => {
  const arr = dataUrl.split(",");
  const mime = arr[0].match(/:(.*?);/)[1];
  const bstr = atob(arr[1]);
  let n = bstr.length;
  const u8arr = new Uint8Array(n);
  while (n--) {
    u8arr[n] = bstr.charCodeAt(n);
  }
  return new File([u8arr], fileName, { type: mime });
};
```