МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет)__Автоматизації і комп'ютерних систем____
Кафедра _____Інформаційних систем_____

«До захисту в ЕК»	**«До захисту допущено»**
Директор інституту(декан факультету)	Завідувач кафедри
_____ ___Форсюк А.В.___	_____ ___Чумаченко С.М.___
(підпис) (прізвище та ініціали)	(підпис) (прізвище та ініціали)
«___» _____ 2021р.	«___» _____ 2021р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

зі спеціальності__122 «Комп'ютерні науки»_____
(код та назва спеціальності)
освітньо-професійної програми___«Комп'ютерні науки»_____
_____
на тему:_«Розроблення автоматизованої системи теоретичної та практичної____ підготовки IT-спеціалістів»_____

Виконав: здобувач__4_ курсу, групи__3__ Кривець Олександр Юрійович
(прізвище та ініціали)

Керівник М'якшило Олена Михайлівна _____
(прізвище та ініціали) (підпис)

Консультанти М'якшило Олена Михайлівна
(прізвище та ініціали) (підпис)

_____ _____
(прізвище та ініціали) (підпис)

_____ _____
(прізвище та ініціали) (підпис)

Рецензент Клименко Олег Миколайович _____
(прізвище та ініціали) (підпис)

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Здобувач_____
(підпис)

Київ - 2021р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) ____*Автоматизація і комп'ютерних систем*_____
Кафедра _____*Інформаційних систем*_____
Освітній ступінь_____*Бакалавр*_____
Спеціальність____*122 «Комп'ютерні науки»*_____

(код і назва)

Освітньо-професійна програма ____*«Комп'ютерні науки»*_____

(назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інформаційних систем
____С.М.Чумаченко___

"29" квітня 2021 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

_____*Кривець Олександр Юрійович*_____
(прізвище, ім'я, по батькові)

1. Тема роботи ___*«Розроблення автоматизованої системи теоретичної та практичної підготовки IT-спеціалістів»*_____
керівник роботи_____*М'якшило Олена Михайлівна, доцент, к.т.н.*_____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від "29" квітня 2021 р. № 248-кс
2. Строк подання здобувачем роботи___*3 червня 2021 року*_____
3. Вихідні дані до роботи ____*Умови створених завдань, інформація про зареєстрованих користувачів, результати проходження завдань*_____
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)__ *системний аналіз, технічне завдання, постановка задачі, модель документів бази даних, база даних, опис процесу розробки сервоної та клієнтської частин додатку, опис реалізованих функцій, інструкція користувача, висновки*_____
5. Перелік графічного матеріалу_*Модель колексцій документів бази даних, знімки інтрефейсу користувача, фрагменти коду*_____

6. Консультанти розділів роботи

Розділи	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	к.т.н., доцент. М’якшило О.М.		
2	к.т.н., доцент. М’якшило О.М.		
3	к.т.н., доцент. М’якшило О.М.		
4	к.т.н., доцент. М’якшило О.М.		

7. Дата видачі завдання_____15 березня 2021 року_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Системний аналіз процесу теоретичної та практичної підготовки ІТ-спеціалістів	15.03.2021-07.04.2021	Виконано
2	Розробка задач автоматизації процесу підготовки ІТ-спеціалістів	08.04.2021-19.04.2021	Виконано
3	Розробка інформаційної системи	20.04.2021-23.05.2021	Виконано
4	Оформлення пояснювальної записки та створення презентації	20.05.2021-1.06.2021	Виконано

Здобувач _____ ___Кривець О.Ю.__
(підпис) (прізвище та ініціали)

Керівник роботи _____ __М’якшило О.М.__
(підпис) (прізвище та ініціали

АНОТАЦІЯ

Головною метою даної бакалаврської роботи є розроблення автоматизованої системи теоретичної та практичної підготовки ІТ-спеціалістів для пришвидшення та спрощення процесів створення та перевірки завдань.

Дипломна робота представляє собою Web-додаток, що допомагає користувачам покращити свої знання та ділитися ними з іншими.

Додаток має зручний інтерфейс для створення та виконання практичних та теоретичних завдань. Реалізовано можливість автоматизованої перевірки завдань.

Для розробки даної інформаційної системи використовувалися сучасні та популярні підходи, мови програмування, бібліотеки та фреймворки.

Об’єктом дослідження є процес автоматизації при підготовці ІТ-спеціалістів.

Бакалаврська робота містить 89 сторінок, 18 таблиць, 26 рисунків, 3 додатки і 19 літературних джерел.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНА СИСТЕМА, КОРИСТУВАЧ, ЗАВДАННЯ, БАЗА ДАНИХ, БІБЛІОТЕКА, REACT, JAVASCRIPT, TYPESCRIPT, NEST.

.

ANNOTATION

The main purpose of this bachelor's thesis is to develop an automated system of theoretical and practical training of IT specialists to speed up and simplify the process of creating and testing tasks.

Thesis is a Web application that helps users improve their knowledge and share it with others.

The application has a user-friendly interface for creating and performing practical and theoretical tasks. The possibility of automated verification of tasks is implemented.

Modern and popular approaches, programming languages, libraries and frameworks were used to develop this information system.

The object of research is the process of automation in the training of IT specialists.

The bachelor's thesis contains 89 pages, 18 tables, 26 figures, 3 appendices and 19 references.

KEYWORDS: INFORMATION SYSTEM, USER, TASKS, DATABASE, LIBRARY, REACT, JAVASCRIPT, TYPESCRIPT, NEST.

ЗМІСТ

ВСТУП

Інформація в сучасному світі перетворилася в один із найбільш важливих ресурсів, а інформаційні системи стали необхідним інструментом практично в усіх сферах діяльності. Різноманітність завдань, що вирішуються за допомогою ІС, призвела до появи множини систем різного типу та призначення, які відрізняються принципами побудови, використаними в процесі розробки технологіями, кросплатформністю, закладеними в них правилами обробки інформації.

Розробка Web-додатків є вкрай популярною в наш час. Сьогодні користувачі володіють різними типами електронних пристроїв, починаючи від комп'ютерів, закінчуючи телевізорами та планшетами. Майже всі сучасні гаджети підтримують можливість користування браузером, саме тому кількість користувачів Web-додатками є неймовірно великою.

Процес навчання також зараз проходить здебільшого з використанням різноманітних електронних пристроїв. Електронні засоби навчання є доволі зручними, вони допомагають ефективно поглиблювати знання в певній сфері, не виходячи з дому. Кількість різноманітних інтернет курсів також зростає і всі вони потребують інформаційної підтримки. Зокрема в ІТ-сфері такий тип курсів є найбільш популярним, але він потребує значних ресурсів в перевірці знань студентів. Але цей процес можна спростити та пришвидшити за рахунок використання спеціалізованих додатків. Саме тому не виникає питань в актуальності таких інформаційних систем.

РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ПРОЦЕСУ ПРАКТИЧНОЇ ТА ТЕОРЕТИЧНОЇ ПІДГОТОВКИ ІТ-СПЕЦІАЛІСТІВ.

1.1. Необхідність підготовки нових спеціалістів для сфери інформаційних технологій

Інформаційні технології уже давно стали невід'ємною частиною розвитку суспільства. Необхідність створювати, розвивати та удосконалювати програмне забезпечення актуалізує потребу у нових спеціалістах. Заклади вищої освіти збільшують кількість місць та розширюють напрями підготовки майбутніх ІТ-фахівців.

Український ринок інформаційно-технічного обслуговування швидко зростає, зокрема за останні 3 роки експорт ІТ послуг з України збільшився на 20%. За даними компанії HackerRank, наша країна посідає 11 місце серед 50 держав із найвищим рівнем розробки ПЗ у світі та 6 місце у рейтингу кращих програмістів.

За даними кадрового порталу HeadHunter, впродовж останніх років попит на кваліфікованих ІТ-фахівців є не просто стабільним, а постійно зростаючим й наразі він значно перевищує пропозицію, адже в умовах сьогодення розвиток будь-якого підприємства пов'язаний з його ІТ-інфраструктурою, яка, звісно, потребує сервісного обслуговування. Якщо на початку використання інформаційних технологій на підприємстві працівникам відділу кадрів було достатньо самостійно підшукати декілька фахівців, окресливши претендентам їх основні завдання та обов'язки. Наприклад, створення й підтримка роботи сайту підприємства, встановлення необхідного програмного забезпечення на всіх комп'ютерах працівників підприємства, забезпечення «життєздатності» та ефективності інформаційних ресурсів підприємства. То з часом по мірі ускладнення ІТ-інфраструктури підприємства виникає необхідність в розширенні штату ІТ-фахівців, що вже вимагатиме створення ІТ-підрозділу,

який без допомоги професіонала навряд чи вдасться укомплектувати. Тобто окрім пошуку нових фахівців виникає потреба в підготовці існуючих.

У зв'язку з появою нових чи розширенням існуючих підприємств та компаній збільшується попит на кваліфікованих спеціалістів. Саме тому виникає необхідність в якісній подготовці нових кадрів.

1.2 Особливості підготовки фахівців сфери інформаційних технологій

Щорічно українські вищі навчальні заклади випускають близько 16 000 фахівців у сфері інформаційних технологій, але тільки близько 5000 з них потім працюють за фахом. І попри те, що ринок відчуває брак фахівців. Роботодавці скаржаться, що не так просто знайти кваліфікованого програміста для виконання їхніх завдань, якого не треба довчати протягом значного часу.

Якою ж має бути підготовка програмістів, щоб вирішити проблему в повному обсязі та задовольнити попит? Перш за все, слід зазначити, що підготовку програміста можна розділити на дві частини.

Перша частина – фундаментальна (теоретична) підготовка. Вона включає дискретну математику, основи алгоритмізації, основи об'єктно-орієнтованого підходу, який на даний момент представляє головну парадигму програмування, дві-три конкретні мови програмування з числа найбільш популярних (Javascript, C#, Java).

З цією частиною зазвичай все краще організовано в навчальних закладах, ніж у навчальних центрах, які організовують курси. Це обумовлено наявністю у перших відповідних ресурсів – викладачів-теоретиків і часу. Курси, як правило, орієнтовані на основний запит клієнтів – тривалість підготовки повинна бути максимально короткою. Водночас навчальні заклади можуть дозволити собі вивчати матеріал досить докладно.

Друга частина – практична підготовка. Вона передбачає вивчення сучасних технологій, що охоплюють завдання з різних областей, отримання практичного досвіду розробки, формування навичок використання існуючих API (інтерфейсів прикладного програмування), навичок написання грамотного, універсального, розширюваного та відмовостійкого коду. Ця частина зазвичай краще відпрацьована у комерційних цент-рах, орієнтованих на виконання студентами нав-чальних проектів різної складності під керівництвом ку-раторів-менторів з числа програмістів-прак-тиків.

Якісно забезпечити обидва згаданих боки під-готовки з нуля на належному рівні можуть ті віт-чизняні ВНЗ, що вирішили бути в тренді ринку праці. Для досягнення високого рівня освіти потрібно виконати дві умови: взаємодія з компаніями, що займаються розробкою програмного забезпечення, і постійна орієнтація на найсучасніші технології.

1.3. Харектеристика, організаційна структура підприємства

1.3.1. Загальна характеристика компанії ДЕВ ЕДЬЮКЕЙШН

ДЕВ ЕДЬЮКЕЙШН — освітній некомерційний проект. Мета проекту - навчити IT спеціальності максимальну кількість зацікавлених і допитливих людей по всьому світу та дати їм можливість отримати високооплачувану професію у максимально короткий термін. Навчальні та розробницькі центри розташовані у Києві, Харкові, Дніпрі, Санкт-Петербурзі та Баку.

Компанія надає кваліфікованих IT-фахівців на outstaff-умовах. Головний офіс знаходиться в Дніпрі, і ще десятки філій відкриті по всьому світу.

Компанія надає якісні освітні послуги. В шататі компанії багато викладачів є діючими спеціалістами з великим практичним досвідом.

В компанії налічується понад 1200 спеціалістів, серед яких: бізнес аналітики, бухгалтери, програмісти, тестувальники, менеджери. Тому структура підприємства є досить великою і складною.

1.3.2 Структура ДЕВ ЕДЬЮКЕЙШН

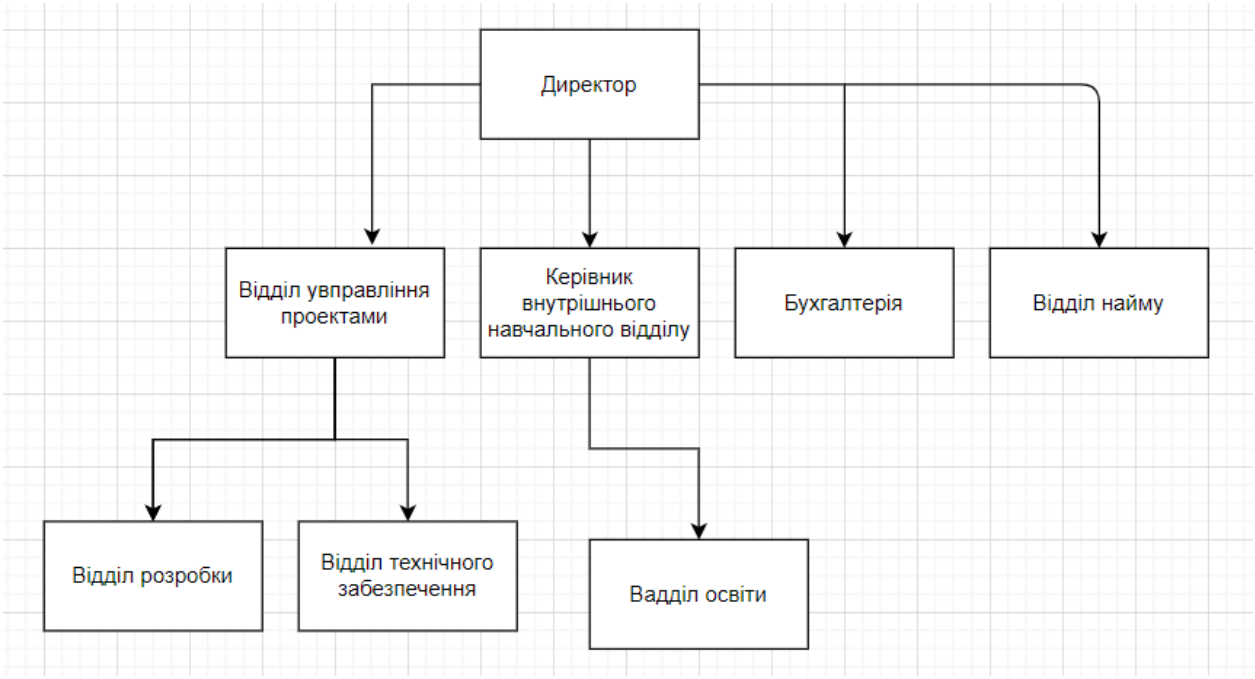

Рис. 1.1. Структура компанії ДЕВ ЕДЬЮКЕЙШН

1.3.3. Аналіз роботи та функцій компанії ДЕВ ЕДЬЮКЕЙШН

Таблиця 1.1. Робота та функції компанії

№	Задачі	Функції
1.	Мета та завдання компанії	• Надання освітніх послух в сфері IT • Впровадження найефективніших рішень; • Розробка програмного забезпечення; • Надання комфортних робочих умов для працівників компанії;
2.	Функції відділу найму	• Пошук та відбір резюме кандидатів • Проведення співбесід

		• Укладення договорів з працівниками
4.	Функції відділу управління проектами	• Планування термінів виконання проекту • Впровадження нових проектів • Комунікація з замовниками
	Функції керівника внутрішнього навчального відділу	• Організація навчальних курсів • Відкриття нових освітніх коледжів • Утвердження навчальних планів
5.	Функції відділу освіти	• Проведення внутрішніх освітніх курсів для працівників компанії • Проведення освітніх конкурсів • Проведення тренінгів • Проведення зовнішніх курсів для студентів • Розробка освітніх програм
6.	Функції бухгалтерії	• Вирішення всіх фінансових питань • Видача заробітної плати для працівників компанії • Участь у підписанні договорів між клієнтом та компанією • Підтримка працівників фірми у фінансових питаннях.
7.	Функції відділу технічного забезпечення	• Забезпечення необхідною технікою працівників компанії • Закупівля необхідної техніки для компанії • Ремонт комп’ютерної техніки

8.	Функції відділу розробки	• Розроблення ПЗ • Тестування ПЗ • Підтримка ПЗ

1.3.4 Взаємодія структурних підрозділів із відділом розробки

Взаємодію між відділом розробки й іншими підрозділами показано в табл.

Таблиця 1.2. Взаємодія підрозділів

№	Підрозділ	Взаємовідносини
1.	Відділ технічного забезпечення	Отримання необхідної техніки необхідної для розробки програмного забезпечення, отримання допомоги по системних питаннях.
2.	Відділ управління проектами	Спільна участь у розробці програмного забезпечення, отримання плану та термінів розробки, комунікація з замовниками по питаннях бізнес логіки проекта.
3.	Відділ найму	Пошук нових кваліфікованих працівників
4	Відділ освіти	Підвищення кваліфікації для працівників компанії та підготовка на навчання нових спеціалістів

1.4. Аналіз та виявлення проблем підготовки ІТ-спеціалістів

Швидка оновлюваність ІТ вимагає постійного саморозвитку, навчання і підвищення кваліфікації викладачів або використання висококваліфікованих програмістів для проведення занять. Студентам не вистачає практичних занять, де б вони могли знайомитися з сучасними технологіями, новими бібліотеками, фреймворками, писати міні-проекти. Окрім цього, перевірка практичних завдань займає доволі багато часу. Також доволі часто процес контролю знань студентів проходить з використаннять паперових матеріалів, що безумовно ускладнює процес перевірки.

З метою підвищення рівня підготовки студенти старших курсів паралельно проходять навчання при великих IT-компаніях, де викладаються сучасні технології і є хороші можливості для подальшого працевлаштування. Але такі курси не дають повноційнної освіти, окільки доволі часно компанії нехтують теоретичною базою на користь практичних навичок які потрібні для роботи.

Нажаль наявне програмне забазбечення не вирішую всіх проблеми, саме тому є необхідність в більше детальному дослідженні проблем, формуванню задач автоматизації та розробці нової інформаційної системи, що вирішує ці проблеми.

1.5. Виявлені недоліки

Отже, за наявного стану процесу підготовки виявлено наступні проблеми:

- Оскільки процес навчання та контролю знань інколи проходить з використанням паперових матеріалів, то це зповільнює та поскладнює процес перевірки;
- Не вистачає практичних та теоретичних завдань;
- Рідко використовуються сервіси та веб додатки, де студенти можуть знайомитися та практикуватися з сучасними технологіями.

1.6. Задачі автоматизації

Для вирішення виявлених проблем було вирішено автоматизувати певні процеси підготовки IT-спеціалістів.

Задачі для автоматизації:

- Створення практичних та теоретичних завдань;
- Можливість автоматичної перевірки завдань;
- Можливість аналізувати помилоки;

1.7. Огляд існуючих систем-аналогів для практичної та теоретичної підготовки IT-спеціалістів

Для огляду були обрані найпопулярніші освітні інформаційні системі за даними різноманітних сервісів.

1.7.1. Інтерактивні онлайн курси HTML Academy

Відомий веб сервіс, що допомагає навчитися програмувати на Javascript, php, освоїти HTML, CSS, React, Vue, Node.js. Вони обрали одне направлення, так званий Front end та зробили детальні тренажери по веб-технологіям. Сервіс налічую понад 1500 завдань[2].

Основні функції сервісу:

- Освоєння теоритичних завдань;
- Закріплення знань на практиці після кожного пункту теорії;
- Надає сертефікати всім хто закінчив певний курс;

Недоліки засобу:

- Тільки перша частина кожного курсу є безкоштовною;
- Не має можливості створювати власні завдання;
- Я орієнтованим на певну тему, в саме веб розробку;
- Не має можливості перевірити теоретичні знання;

1.7.2. JavaRush

JavaRush – це онлайн курс навчання програмуванню на Java, що на 80% складається з практики. Курс JavaRush включає 1200 практичних завдань зростаючої складності[3].

Основні функції сервісу:

- Можливість читати теоретичний матеріал;
- Можливість вирішувати практичні завдання;
- Навчання проходить у вигляді гри;
- Автоматична перевірка практичних завдань;

- Має власний форум та співтовариство, що налічує багато учасників;
- Після закінчення курсу надає сертифікат та можливість стажування;

Недоліки засобу:

- Не має можливості створювати власні завдання;
- Я орієнтованим на певну тему, в саме вивчення мови програмування Java;
- Не всі матеріали та завдання є безкоштовними;
- Оскільки навчання йде у вигляді гри та історії то викладення матеріалу є доволі затянутім;
- Не має можливості пропустити певну часниту курсу;
- Не має можливості перевірити теоретичні знання;

1.7.3. CodeWars

CodeWars – це сервіс де кожен може вдосконалити свої практичні навички тренуючись з іншими на реальних випробуваннях коду. Має великий вибір тематик[4].

Основні функції додатку:

- Киристувачі можуть виконувати практични завдання;
- Є можливість створеня власних завдань;
- Є можливість порівнювати свої розв’язки з іншими;
- Перевірка розв’язків здійснюється за допомогою Unit тестів

Недоліки:

- Не має можливості ознайомлюватися з теоретичним матеріалом;
- Не має можливості перевірити теоретичні знання;

1.7.4. Порівняння систем-аналогів

Table 1.3. Порівняння систем-аналогів

Системи / Функції	HTML Academy	JavaRush	CodeWars
Можливість виконання практичних завдань	+	+	+
Можливість виконання тестових завдань	-	-	-
Можливість вибору виконуваного завданя	-	-	+
Має можливість ознайомитися з теоретичною інформацією	+	+	-
Можливість створювати власні завдання	-	-	+
Автоматизована перевірка завдань	+	+	+
Можливість аналізувати свої відповіді та результати	+	+	+
Надає сертифікати після проходження курсу	+	+	-
Має можливість вибору направлення	-	-	+
Витрати	Залежить від обраного курсу	Залежить від обраного курсу	Безкоштовно

Висновок: після дослідження трьох освітніх сервісів: HTML Academy, JavaRush, CodeWars бачимо, що студентів є вибір сервіс для практичної підготовки. Але більшість засобів дає можливості виконувати теоретичні завдання. Із розгялнутих систем тільки CodeWars є безкоштовним, а у всіх інших тільки певна частина є доступною для всіх. Перевагами даних додатків є можливість автоматизованої перевірки завдань, також деяки сервіси видають сертефікати пілся закінчення, що безумовно є плюсом. Нажаль не всі сервіси дають можливість покращити саме теоретичні знання. Також тільки один сервіс

з розгялянутих, а саме CodeWars, дозволяє користувачам створювати власні завдання, окрім цього, тільки цей сервіс має великий вибір направлень, понад 30 мов програмування.

1.8. Обґрунтування доцільності проектування та розробки системи теоретичної та практичної підготовки IT-спеціалістів

На даний момент існує доволі великий вибір систем для підготовки IT-спеціалістів. Багато систем надають гарну теоретичну чи практично підготовку. Нажаль далеко не всі системи маю вибір направлення при підготовці, деякі з них є орієнтованими на щось одне, наприклад веб-розробка. Чудовим є те, що багато сервісів-курсів видають своїм випускникам сертифікати. Але прикро, що більшість наявних систем потребують оплати, та не дозволяють створювати власні завдання. Бальшість систем мають можливість автоматизованої перевірки завдань. Але далеко не всі сервіси надають можливість саме теоретичної підготовки.

З урахуванням цього вирішено створити Web-додаток, який буде повністю задовольняти і виконувати поставлені задачі при підготовці IT-спеціалістів. Додаток має стати зручним сервісом теоретичної та практичної підготовки, та зможе бути використаний як для самостійного навчання так і в рамках освітнього процесу у вищих навчальних закладах.

1.9. Концептуальна модель системи

Метою створюваного освітнього сервісу пришвидшення, покращення та спрощення процесів теоретичної та практичної підготовки IT-фахівців. Можливість автоматизованої перевірки завдань значно спростить роботу вкиладачів. Процес розповсюдження завдань також зпроститься, оскільки викладач зможе надсилати посилання студентам. Використання студентами такого додатку дозволить їм покращити свої теоретичні та практичні знання.

Безумовно з появою такого сервісу з’явиться велика кількість доступних теоретичних та практичних завдань.

При розробці даної концептуальної моделі були створені нові вхідні та вихідні дані.

Вхідні дані:

- Інформація про користувача;
- Дані для створення завдання;
- Сервіс перевірки практичних завдань;

Вихідні дані:

- Створені завдання;
- Теоретичні матеріали;
- Результати виконання завдань;

Побудована модель TO-BE процесу розробки автоматизованої системи практичної та теоретичної підготовки електронної, складається з таких процесів:

- проектування бази даних;
- розробка серверного додатку;
- розробка клієнтського додатку;
- тестування та перевірка роботи додатку,

які показано на першому рівні декомпозиції. Після розробки серверної та клієнтської частини, в процесі перевірки роботи додатку реєструється тестовий користувач, здійснюється вхід до системи та симулюються процеси додавання нових завдань і здійснюються спроби їх виконання.

Для розвитку даного додатку в майбутньому планується додати декілька вбудованих сервісів для перевірки практичних завдань. Також планується додати сторінку де б користувач могли бачити статистику проходження їх

завдань. Також для розриширення додатку планується реалізувати переклад інтерфейсу на інші мови.

1.10. Переваги та недоліки дистанційного навчання

Переваги:

- Гнучкість. Можливість займатися в зручне для себе час, у зручному місці і темпі. Нерегламентований відрізок часу для освоєння дисципліни.

- Модульність. Можливість з набору незалежних навчальних курсів – модулів формувати навчальний план, що відповідає індивідуальним чи груповим потребам.

- Охоплення. Одночасне звертання до багатьох джерел навчальної інформації (електронним бібліотекам, банкам даних, базам знань і т.д.) великої кількості що навчаються. Спілкування через мережі зв’язку один з одним і з викладачами.

- Економічність. Ефективне використання навчальних площ, технічних засобів, транспортних засобів, концентроване й уніфіковане представлення навчальної інформації і мультидоступ до неї знижує витрати на підготовку фахівців.

- Технологічність. Використання в освітньому процесі новітніх досягнень інформаційних і телекомунікаційних технологій, що сприяють просуванню людини у світовий постіндустріальний інформаційний простір.

- Охоплення. Одночасне звертання до багатьох джерел навчальної інформації (електронним бібліотекам, банкам даних, базам знань і т.д.) великої кількості що навчаються. Спілкування через мережі зв’язку один з одним і з викладачами.

- Економічність. Ефективне використання навчальних площ, технічних засобів, транспортних засобів, концентроване й уніфіковане представлення навчальної інформації і мультидоступ до неї знижує витрати на підготовку фахівців.

- Технологічність. Використання в освітньому процесі новітніх досягнень інформаційних і телекомунікаційних технологій, що сприяють просуванню людини у світовий постіндустріальний інформаційний простір.

Недоліки

- Індивідуалізація зводить до мінімуму обмежене в навчальному процесі живе спілкування викладачів і студентів між собою, пропонуючи їм спілкування у вигляді діалогу з комп'ютером. Це приводить до того, що студент, який активно користується живою мовою, надовго замовкає при роботі із засобами освітніх електронних видань і ресурсів, що особливо характерно для людей, що навчаються дистанційно.

- Орган об'єктивізації мислення людини – мова виявляється виключеним протягом усіх років навчання. Студент не одержує достатньої практики діалогічного спілкування, формування й формулювання думки професійною мовою.

- Використання освітніх електронних видань і ресурсів скорочує кількість соціальних контактів, скорочує кількість практики соціальної взаємодії й спілкування, індивідуалізм.

1.11. Техніко-економічне обґрунтування

Вихідні дані для розрахунку:

- Ступінь новизни розроблюваних задач – "В" – використання типових проектних рішень за умови їх змін.
- Група складності алгоритму – 1.
- Узагальнені дані вхідної та вихідної інформації системи «Автоматизована система теоретичної та практичної підготовки» показані у табл. 1.4.

Таблиця 1.4. Узагальнені дані

Вид інформації	**Позначення**	**К-сть наборів даних**
Змінна інформація	ЗІ	m=7
Нормативно-довідкова інформація	НДІ	n=3
Банк (база) даних	БД	p=1
Обробка в режимі реального часу	РЧ	так
Забезпечення телекомунікаційної обробки даних і управління віддаленими об'єктами	ТОУ	ні

Витрати часу на систему, а саме на розробку ескізного проекту Т1 і технічного завдання Т2, будуть наступні табл. 1.5.

Таблиця 1.5. Визначення витрат часу

Вид системи	**Стадія розробки системи**	
	Передпроектне дослідження	**Технічне завдання**
	В	В
Управління науково-технічною інформацією	Т1= 67	Т2 =24

Визначається базове значення витрат часу для стадій "Технічний проект", "Робочий проект" і "Впровадження".

Вхідними даними для визначення є:

- Кількість форм вхідної інформації – В1 = 3,
- Кількість форм вихідної інформації – В2 = 4,
- Базове значення витрат часу для стадій "Технічний проект" – $Т_{Б3}$= 77;
- Базове значення витрат часу для стадій "Робочий проект" – $Т_{Б4}$= 215;
- Базове значення витрат часу для стадій "Впровадження" – $Т_{Б5}$= 75.
- Базове значення витрат часу – $Т_Б$ коригується за допомогою поправочних коефіцієнтів для всіх стадій розробки автоматизованої системи.

Розрахунок витрат часу для стадії "Технічний проект" (Тз).

Коефіцієнт трудомісткості робіт kп визначається за формулою:

Кп [redacted]

Таблиця 1.6 Коефіцієнти k1, k2, k3 для стадії "Технічний проект"

Вид використаної інформації	**Ступінь новизни**
	В
k1 (ЗІ)	1.0
k2 (НДІ)	0.72
k3 (БД)	2.08

Таблиця 1.7. Коефіцієнт ступеню новизни проекту, kO

Стадія розробки системи	**Вид обробки**	**Ступінь новизни**
		В
Технічний проект	РЧ	1.26
Робочий проект	РЧ	1.32
Впровадження проекту	РЧ	1.21

Коефіцієнт ступеню новизни проекту, kO, визначається з урахуванням того, що процес обробки здійснюється в режимі реального часу.

Витрати часу для стадії “технічний проект” T_3:

$$T_3 = T_{Б3}*k_п*k_о = 77*0.997*1.26 = 96.728;$$

Розрахунок витрат часу для стадії "Робочий проект" (T_4) для системи.

Для визначення витрат часу на стадії "Робочий проект" використовується формула. Де $k_п$ – коефіцієнт, що враховує вид використаної інформації.

Кп [redacted]

Таблиця 1.8. Коефіцієнти k1, k2, k3 для стадії "Робочий проект".

Вид використаної інформації	**Група складності алгоритму**	**Ступінь новизни**
		В
k1 (ЗІ)	1	1.1
k2 (НДІ)	1	0.58
k3 (БД)	1	0.48

Коефіцієнт, що враховує вид обробки інформації на стадії "Робочий проект" показано в табл. 1.8.

Коефіцієнт складності контролю вихідної та вхідної інформації визначається на стадії "Робочий проект" та "Впровадження", $k_с = 1.00$

Витрати часу T_4 вимірюються в людино-днях:

$$T_4 = T_{Б4}*k_п*k_о*k_с = 215*0.962*1.32*1.00 = 273.015;$$

Поправочні коефіцієнти мають такі ж значення, як і при обрахунку T_4.

Для стадії визначення загальних витрат часу на "Впровадження" T_5 (люд-днів) використовують формулу $T_5 = T_{Б5}*k_п*k_о*k_с = 75*0.962*1.21*1.00 = 87.301;$

Отже, загальні витрати на проектування системи складають:

$$T_\sum = 70+43+96.728+273.015+87.301 = 570.046_{(люд\text{-}дн).}$$

Для кваліфікаційної роботи кількість робочих годин складає 530 ураховуючи 7 годинний робочий день, тому на розроблення проекту виділено Ф, днів:

$$Ф = [redacted] = 75$$

Отже, тоді визначаємо кількість місяців якщо на місяць припадає 25 робочих днів. Кількість місяців на розробку - М:

$$М = \frac{Ф}{25} = \frac{75}{25} = 3$$

Отже, для виконання нашого проекту необхідно чисельність виконавців Ч, що обраховується за формулою:

$$Ч = \frac{T\sum}{[redacted]} \quad 570.046$$

Якщо прийняти, що оплата програміста здійснюється в розмірі 17500 грн. (середня оплата праці програміста в Україні), то оплата праці всіх робітників, складе:

$$V'_1 = Ч*М*ЗП_{пр} = 8*3*17500 = 420000 \text{ грн.}$$

Витрати, пов’язані з розробкою інформаційнох системи:

Дійсний річний фонд часу ПК в годинах дорівнює в середньому 5год/міс + 6 роб. днів/рік.

$$T_{ПК} = 3723-(6*8+5*12) = 3615 \text{ год.}$$

Оскільки під час виконання кваліфікаційної роботи студент в середньому витрачає приблизно 450 год. м. часу, то величина фонду часу ПК дорівнює:

$$T'_{пк} = 3615 * [redacted] = 436.9 \text{ год}$$

Поточні витрати на експлуатацію V1:

ЦР – ринкова вартість ПК, орієнтовно складає 17000 грн., КУН – коефіцієнт, що враховує витрати на установку і налагодження ПК і дорівнює: $Ц_{пк} = Ц_р * (1 + К_{ун}) = 17000 * (1 + 0,12) = 19040$ грн.

Амортизаційні відрахування використання ПК, АМ, норма амортизаційних відрахувань, яка для ПК дорівнює НА = 5:

$З_{АМ} =$ [redacted] $= 3808$ грн.

Витрати на електроенергію, споживану ПК:

$$З_{ЕЛ} = Р_{ПК} * Т'_{пк} * Ц_{ЕЛ} * А = 0.032* 436.9 * 2.1 * 0.9 = 26.42 \text{ грн}$$

ЗР – витрати на поточний ремонт і технічне обслуговування ПК визначаються як 6% від балансової вартості ПК, ЦПК.

$$З_Р = Ц_{ПК} * 0.06 = 19040 * 0.06 = 1142.4 \text{ грн.}$$

ЗМАТ – це непрямі витрати, пов'язані з експлуатацією ПК. Визначаються як 5% від балансової вартості ПК ЦПК.

$$З_{МАТ} = Ц_{ПК} * 0.05 = 19040 * 0.05 = 952 \text{ грн.}$$

Таким чином, маємо, що заробітна плата обслуговуючого персоналу:

ЗОП = 3360 грн, ЗАМ = 896 грн, ЗЕЛ = 26.42 грн,

Поточні витрати на експлуатацію V_1'', грн, визначаються:

$$V_1'' = З_{ОП} + З_{АМ} + З_{ЕЛ} + З_Р + З_{МАТ} = 3360 + 3808 + 26.42 + 1142.4 + 952 = 9289 \text{грн}$$

Отже, загальні витрати на розроблення інформаційної системи розраховуються за формулою та складуть:

$$V = V'_1 + V_1'' = 420000 + 9289 = 429289 \text{ грн}$$

Нехай, витрати на придбання і установку ПК V_2

$$V_2 = Ц_{ПК} = 19040 \text{грн.}$$

Витрати на підготовку приміщення V_3.

Оскільки, ми маємо пристосоване приміщення, то:

$$V_3 = 0 \text{грн.}$$

Витрати на навчання персоналу V_4

В середньому навчання персоналу триває декілька днів, нехай в нашій ситуації це буде 3 дня, тому можна вважати, що:

$$V_4 = 2550_{грн.}$$

Загальна вартість розробки та впровадження системи V_Σ, вираховується за формулою:

$$V_\Sigma = V_1 + V_2 + V_3 + V_4 = 420000 + 19040 + 0 + 2550 = 450879 \text{ грн.}$$

Оскільки норма амортизаційних втрат для комп'ютерних систем $H_A = 5$, тому для обрахування річного економічного ефекту необхідно брати до розгляду величину:

$$Vp = [redacted] = 90175.8_{грн.}$$

Таблиця 1.9. Основні джерела прибутку від впровадження Web-додатку.

№	**Джерела прибутку**	**∑(сумма)**
1	Скорочення затрат на друк	15233 грн
2	Оптимізація роботи викладачів	10135 грн
3	Зменшення витрат на оренду приміщення	21310 грн
4	Скорочення затрат на придбання різноманітних курсів	11456 грн
		58134 грн

Коефіцієнт економічної ефективності розробки:

$$К_{ЕФ} = 58134/90175.8 = 0.64$$

Термін окупності розробки ІС дорівнює: $Т_{ОК} = 1/0.64 = 1.56$

Виходить, що термін окупності інформаційної системи складе майже **півтора** роки.

РОЗДІЛ 2. ТЕХНІЧНЕ ЗАВДАННЯ НА ПРОЕКТУВАННЯ

2.1. Загальні положення

Найменування системи: «Автоматизована система теоретичної та практичної підготовки IT-спеціалістів»

2.2. Призначення і цілі створення системи

Метою даного проекту є пришвидшення, покращення, автоматизація та спрощення процесів теоретичної та практичної підготовки IT-спеціалітів.

Головним завданням є автоматизація процесів створення, виконання та перевірки завданя, та розробка зручного інтерфейсу користувача.

Призначенням є автоматизація процесів теоретичної та практичної підготовки IT-спеціалістів з подальшим впровадженням даного додатку в рамках навчального процесу у вищих навчальних закладах.

2.3. Характеристика об’єкта автоматизації

Об’єктом автоматизації є процеси теоретичної та практичної підготовки IT-спеціалістів. Базовий об’єкт впровадження – в рамках навчального процесу в вищих навчальних закладах.

2.4. Вимоги до системи

2.4.1. Вимоги до системи в цілому

2.4.1.1. Вимоги до структури і функціонування системи

Система повинна мати клієнт-серверну архітектуру, що використовує єдину базу даних (надалі — БД). Пропонуються використовувати такі паттерни проектування: MVC, декоратор, фабрика, одинак, щоб система була масштабованою. Фабрика – для створення нових модулів серверного додатку.

Використати декоратори для реалізації класів котроллерів, сервісів, репозиторіїв та та методів серверного додатку. MVC - при написанні клієнтської частини розділити програму на 3 частини: модель – все що відноситься до сховища даних, зображення – логі що відповідає за відображення графічного інтерфейсу та контроллер – основна логіка прогами, де здійснюється запити на сервер та відбувається обробка даних, та виклики методів збереження даних чи чи відображення елементів інтерфейсу.

Розроблювана система повинна мати зручний інтерфейс. Є необхідність в розробці сторінок авторизації.

Система повинна мати панель навігації, для зручного переходу між сторінками.

В графічному інтерфейсі мають переважати темні кольори, що користувачам було зручно довго працювати в даній системі.

Всі написи та тексти що відображаються користувачу засобами графічного інтерфейсу мають бути подані англійською мовою, щоб сервісом мали змогу користуватися користувачі з різних країн.

Для зручного користування на тих сторінках де це потрібно мають бути реалізовані пошуки та фільтри.

2.4.1.2. Вимоги до чисельності і кваліфікації персоналу

Користувачам даного Web-додатку є всі хто бажає покращити свої знання в напрямку IT чи поділитися власними знаннями, в тому числі студенти та викладачі університету.

2.4.1.3. Показники призначення

Додаток повинен виконувати показники призначення описані в п. 2.1.

2.4.1.4. Вимоги до надійності

Всі функції системи виконуються дискретно. База даних повинна буди захищеною паролем та при розміщенні даного додатку на хостинг повинна знаходитися в незалежному Docker контейнері. Є необхідність в можливості налаштування даних потрібних для підключення до БД, за допомогою файл оточення - .env. Також є необхідним робити так звані «snapshots» (знімки), щоб про всяк випадок мати резервні копії бази даних.

2.4.1.5. Вимоги до безпеки

Для забезпечення безпеки при експлуатації, налагодженні, монтажі, обслуговуванні і ремонті технічних засобів системи потрібно дотримуватись вимог ДСТУ: ДСТУ 2293-99, ДСТУ ISO 6309:2007, ДСТУ 12.0.230:2008, ДСТУ 7237:2011, ДСТУ 7238:2011, ДСТУ 7239:2011; по доступним рівням освітленості, вібраційних і шумових навантажень слід дотримуватися вимог відповідно ДСТУ Б А.3.2-15:2011, ДСТУ EN 14253:2018, ДСТУ 2867-94.

2.4.1.6. Вимоги з ергономіки та технічної естетики

Загальні ергономічні і естетичні вимоги до системи повинні відповідати держстандартам ДСТУ 8604:2015, ДСТУ 7298:2013. Освітленість робочого місця повинна відповідати ДСТУ EN 12464-1:2016, ДБН В.2.5-28-2006. Засоби відображення повинні розміщуватися таким чином, щоб кут спостереження екрану складав не більше, ніж 45 градусів, мінімальна відстань спостереження екрану — 0,3 м, рекомендована — 0,5 м.

При розробці графічного інтерфейсу бажано використовувати темні кольори, для запобігання втомлюваності користувача.

2.4.1.7. Вимоги по експлуатації, технічного обслуговування, ремонту і зберігання компонентів системи

Для розміщення веб додатку на хостингу, потрібно налаштувати окремі Docker контейнери для MongoDB, серверної та клієнтської частини.

2.4.1.8. Вимоги до захисту інформації від несанкціонованого доступу

Для захищення персональних даних та є необхідність в розробці процесів авторизації. Паролі від особистих кабінетів користувачів повинні зберігатись в базі даних у закодованому вигляді. Реалізація процесів авторизації користувача повинен бути реалізованим на основі стандарту JWT.

2.4.1.9. Вимоги щодо збереження інформації при аваріях

Необхідно передбачити засоби резервного збереження БД в архіві після коригування і можливість завантажити БД з архіву у випадку її руйнування.

2.4.1.10. Вимоги по захисту від впливу зовнішніх діянь

Засоби, що виключають вплив шкідливих факторів на функціонування технічних засобів, повинні бути спроектовані згідно з ДБН В.2.2-9-2009. Обчислювальні засоби зі стійкості до зовнішніх впливів повинні відповідати ДСТУ 2506-94.

2.4.1.11. Вимоги до патентної чистоти

При створенні даної інформаційної системи патентні дослідження не проводяться.

2.4.1.12. Вимоги по стандартизації і уніфікації

Авторизація користувачів повинна бути реалізована за стандартом JWT. Кодування поролю користувач повинно здійснювати за стандартом SHA-256.

2.4.2. Вимоги до функцій

2.4.2.1. Перелік функцій із зазначенням вхідної та вихідної інформації

Таблиця 2.1. Перелік функцій, вхідної та вихідної інформації

№ п/п	Найменування функції	Вхідна інформація	Вихідна інформація
1.	Створення тестових завдання	Дані для створення тестового завдання	Новий документ «Завдання» в БД
2.	Створення практичних завдань	Дані для створення практичного завдання	Новий документ «Завдання» в БД
3.	Надавати можливість виконання завдань	Документ «Завдання» в БД	Новий документ «Результат» в БД
4.	Перегляд та аналіз результатів	Документ «Результат» в БД	Форма з відображенням результатів та оцінки
5.	Автоматизована перевірка завдань	Відповідь що дав користувач	Форма з відображенням результатів та оцінки

2.4.3. Вимоги до видів забезпечення

Оскільки, створювана інформаційна система це Web-додаток, то система повинна безвідмовно працювати у всіх популярних браузерах, а саме: Google Chrome, Mozilla Firefox, Opera, Microsoft Edge.

Додаток повинен мати простий, швидкий та зручний інтерфейс, зручну навігація, наявність пошуку та фільтрів, в інтерфейсі повинні бути використані переважно темні кольорі.

Програмний продукт повинен мати стабільне підключення Інтернету тільки в той момент коли користувач здійснює переходи між сторінками чи виконує процеси збереження, видалення даних.

Необхідно створити базу даних MongoDB та налаштувати підключення серверної частини додатку до бази даних за допомогою створення так званих змінних оточення в файлі .env.

Таблиця 2.2. Вимоги до технічного забезпечення системи

№ п/п	**Основні характеристики комп'ютера**
Технічне забезпечення для сервера	
1	Intel Core i3 1,8 GHz\ RAM: 2048 \HDD 10 Gb\ LAN 1 Gbit
Технічне забезпечення для клієнта	
1	Монітор 13"
2	Миша USB
3	Клавіатура USB

Контроль і прийняття рішень при аварійних ситуаціях при експлуатації системи здійснює відповідальний за систему.

2.5. Склад і зміст робіт по створенню системи

Стадії створення системи і терміни виконання робіт наведені в таблиці 3.

Таблиця 2.3. Найменування робіт при створенні системи

№ п/п	**Найменування робіт**	**Строки виконання робіт**
1	Дослідження об'єкта автоматизації	01.04.2021
2	Технічне завдання	15.04.2021

3	Технічний проект	05.05.2021
4	Оформлення документації	15.05.2021

2.6. Порядок контролю і приймання системи

Система вводиться в рамках навчального процесу ІТ-спеціальностей в университетах чи спеціалізованих навчальних курсах.

Випробування для визначення рішення про можливість приймання системи в експлуатацію проводять розробники разом із замовником. Програму випробувань складає розробник і затверджує замовник. Здача експлуатацію здійснюється на основі технічного завдання та інструкції користувача. За результатами дослідної експлуатації формується перелік доробок і рекомендацій та час їх виконання. Введення в експлуатацію системи оформлюється актом здачі-прийому.

2.7. Вимоги до складу і змісту робіт із підготовки до введення системи в дію

- проводить укомплектування технічних засобів для студентів;
- розповсюджує інструкцію користувача серед студентів
- вводить систему в експлуатацію;

2.8. Вимоги до документації

На систему розробляється ряд документації у складі: технічне завдання та технічний проект.

Документація розробляється у відповідності з вимогами Державних стандартів серії 19 «Єдина система програмної документації» та серії 24 «Єдина система стандартів автоматизованих систем управління».

2.9. Джерела розробки

При розробленні технічного завдання на систему використано наступні документи:

- ДСТУ 3008-2015. Документація. Звіти у сфері науки і техніки. Структура та правила оформлювання; розповсюджує інструкцію користувача серед студентів
- ДСТУ 3973–2000 Система розроблення та поставлення продукції на

вир ДСТУ Б В.2.5–82:2016 Електробезпека в будівлях і спорудах.

- Вимоги до захисних заходів від ураження електричним струмом.

обництво;

РОЗДІЛ 3. РОЗРОБЛЕННЯ КОМПЛЕКСУ ЗАДАЧ АВТОМАТИЗАЦІЇ

3.1 Інформаційне забезпечення системи.

MongoDB - це надійна база даних, яка рекомендується під час розробки веб-додатків, які масштабуються та потребують великої бази даних для зберігання величезної кількості саме неструктурованих даних, саме тому вона ідеально підходить для нашого проекту. Нажаль не має зручних інструментів для проектування бази даних MongoDB. Тому проектування моделей колекцій документів здійснювалося вручну. Пізніше, в процеці розробки серверної частини, всі ці моделі були описані у вигляді схем.

Моделі документів бази даних:
Документ «User» містить у собі інформацію про зареєстрованих в системі користувачів.

Структуру даної моделі документу описано в табл. 3.1.

Таблиця 3.1. Структура документу «User»

№	Назва поля	Тип даних
1	_id	ObjectId
2	email	String
3	nickname	String
4	lastName	String
5	firstName	String
6	age	Number
7	hash	String

Модель документу «Task» містить у собі дані про створенні завдання.

Структуру даної моделі описано в табл. 3.2.

Таблиця 3.2. Структура документу «Task»

№	Назва поля	Тип даних

1	_id	ObjectId
2	title	String
3	description	String
4	linkForCheck	String
5	showAnswers	Boolean
6	numberOfAttempts	Number
7	timeLimit	Number
8	creatorId	ObjectId
9	questions	Array(ObjectId)
10	type	String

Дані про запитання для тестових завдань зберігаються в документі «Questions».

Структуру даної моделі описано в табл. 3.3.

Таблиця 3.3. Структура документу «Questions»

№	**Назва поля**	**Тип даних**
1	_id	ObjectId
2	title	String
3	type	String
4	options	Array(String)
5	answers	Array(String)

Результа виконання завдань зберігаються в документі «Results»

Структуру даної моделі описано в табл. 3.4.

Таблиця 3.4. Структура документу «Questions»

№	**Назва поля**	**Тип даних**
1	_id	ObjectId
2	taskId	ObjectId

3	mark	Number
4	maxMark	Number
5	resultDate	Date
6	usedTime	Number
7	userId	ObjectId
8	answers	Array(ObjectId)

Відповіді користувачів на тестові завдання зберігаються в документі «UserTestAnswers»

Структуру даної моделі описано в табл. 3.5.

Таблиця 3.5. Структура документу «UserTestAnswers»

№	**Назва поля**	**Тип даних**
1	_id	ObjectId
2	userId	ObjectId
3	questionId	ObjectId
4	answers	Array(String)
5	taskId	ObjectId

Відповіді користувачів на практичні завдання зберігаються в документі «UserPracticalSolutions»

Структуру даної моделі описано в табл. 3.6.

Таблиця 3.6. Структура документу «UserPracticalSolutions»

№	**Назва поля**	**Тип даних**
1	_id	ObjectId
2	userId	ObjectId
3	solution	String
4	rightSolution	String
5	taskId	ObjectId

Створена база даних призначена для зберігання інформації про зареєстрованих користувачів, завдання та результати виконная завдань.

3.2. Алгоритмізація та реалізація комплексу задач автоматизації

3.2.1. Розробка, генерація та заповнення бази даних

Для генерації бази даних, в процесі розробки серверної частини було описано схеми кожно документу засобами бібліотеки mongoose. Приклад схеми документу для моделі User:

```
export const UsersSchema = new mongoose.Schema({
  email: {
    type: String,
    required: true,
    lowercase: true,
    validate: (value: string) => {
      return validator.isEmail(value);
    },
  },
  nickname: String,
  lastName: String,
  firstName: String,
  age: Number,
  hash: String,
});
```

В процеці створення схеми документу чітко указуються типи кожного поля, це зроблена для того, щоб якщо в процесі розробки ми захочемо в БД додати документ іншого формату ти виникне помилка. Тобто ми описуємо схеми для того щоб структура кожного документку була чітко визначеною.

3.2.2. Розробка серверної частини веб додатку

Серверна часнита додатку розроблена засобами фреймворку Nest. Цей фреймворк поєднує в собі декілька технологій: мова програмування Typescript, декоративний підхід, модульна архітектура.

Архітектура серверного дотатку спроектована таким чином, що за всі оперції повязані з деякою обраною сутністю відповідає певний модуль. Наприкалад, за створення та оновлення нових користувачів відповідає модуль Users.

```
@Module({
  imports: [DatabaseModule],
  controllers: [UserController],
  providers: [UsersService, UsersRepository, ...usersProviders],
  exports: [UsersService, UsersRepository],
})
```

Кожен модуль додатку може імпортувати певні класи інших модулів та віддавити на експорт свої класи. За потреби, кожен модуль може мати власний контроллер. В класі контроллері за допомогою спеціальних декораторів оголошуться методи які відповідають за відстеження запитів на певні адреси. Пропоную розглянути детальніше на прикладі контроллеру UserController. Пропоную звернути увагу на приклад створення котроллеру за допомогою декоратору @Controller(‘user’) для класу UserController, параметр в дужках вказає ім’я контроллеру. Також в цьому контроллері оголошено декілька методів, наприклад, getProfile. За допомогою декоратору @Get(‘profile’) ми відмічаємо, що даний метод буде реагувати на всі HTTP, GET запити за адресою http//:localhost/user/profile, де localhost – доменне ім’я, user – ім’я контроллеру, profile - маршрут методу. Тобто параметри які мі передаємо в декоратори Controller та Get формують адресу, за якою нам потрібно буде звертатися для виклику пеного методу, в даному випадку для отримання даних про користувача. Пропоную розглянути наступний метод, updateUserById. Даний метод відслідковує адресу http//:localhost/user/:id, де id – це параметр адреси, який може приймати різні значення, в даному випадку це ідентифікатор користувача. Даний метод викликає мотод іншого об‘єкту, а саме UserService. Сервіси це класи, в яких описані методи в яких відбувається головна безнес логіка: робота з даними, фільтрація, обробка даних та багато іншого.

```
@ApiTags("user")
@Controller("user")
export class UserController {
```

```
constructor(private usersService: UsersService) {}

@ApiOperation({ summary: "Get profile" })
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Get("profile")
getProfile(@Request() req) {
  return req.user;
}

@ApiOperation({ summary: "Update user by id" })
@UseGuards(JwtAuthGuard)
@ApiBearerAuth()
@Put(":id")
updateUserById(
  @Param() { id }: UserUpdateParams,
  @Body(new JoiValidationPipe(UpdateUserSchema)) body: UserUpdateModel
) {
  return this.usersService.updateUserById(id, body);
}
```

Приклад стоворення сервісу, в даному випадку це UsersService, відображено в наступному фрагменті коду. Сервіс створений за допомогою декоратору @Injectable().

```
@Injectable()
export class UsersService {
  constructor(private usersRepository: UsersRepository) {}

  async findOne(id: string): Promise<GetUserDto> {
    return await this.usersRepository.findOne(id);
  }

  async findOneByEmail(email: string): Promise<GetUserDto> {
    return await this.usersRepository.findOneByEmail(email);
  }

  async createUser({ email, password }: CreateUserDto): Promise<boolean> {
    const isAlreadyCreated = await this.findOneByEmail(email);

    if (isAlreadyCreated) {
      throw new BadRequestException({
        message: "User with this email is already registered",
      });
    }
```

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

    await this.usersRepository.createUser({ email, password: hash });
    return true;
  }
```

Пропоную звернути увагу на метод createUser, він відповідає за створення нового користувача. Оскільки для створення нового користувача потрбіно звертатися до бази даних, то для цього спеціальна сутність – репозиторій. Репозиторій - це клас в якому описані методи що здійснють запити до бази даних. В наступному уривку відображено приклад створення UsersRepository. В даному репозиторії реалізовано запити на створення оновлення та пошукт даних в БД.

```
@Injectable()
export class UsersRepository {
 constructor(@Inject("USER_MODEL") private userModel: Model<User>) {}

 async findOne(_id: string): Promise<GetUserDto> {
  return this.userModel.findOne({ _id }).lean();
 }

 async findOneByEmail(email: string): Promise<GetUserDto> {
  return this.userModel.findOne({ email }).lean();
 }

 async createUser({ email, password }: CreateUserDto): Promise<void> {
  const newUser = new this.userModel({ email, hash: password });

  await newUser.save();
 }

 updateUserByCond = async (
  condition: UpdateUserDto,
  updateModel: UserUpdateModel
 ) => {
  return this.userModel.updateOne(condition, { $set: updateModel });
 };
}
```

Конкретно для запитів до БД використовується бібліотека mongoose. На самому початку здійснюється підлючення до БД, за допомогою методу connect(). Код підключення до БД:

```
export const databaseProviders = [
  {
    provide: "DATABASE_CONNECTION",
    useFactory: async (
      configService: ConfigService
    ): Promise<typeof mongoose> => {
      const connectionString = configService.get<string>(
        "database.connectionString"
      );

      return mongoose.connect(connectionString, {
        useNewUrlParser: true,
        useCreateIndex: true,
        useUnifiedTopology: true,
        useFindAndModify: false,
      });
    },
    inject: [ConfigService],
  },
];
```

Для здійснення запитів до БД створюються моделі. Для створення моделей використовуються схеми(див. п. 3.2.1.). Кожна схеме та модель відповідають певному документа та саме модель використовується для здійснення запитів до БД. Приклад створення моделі:

```
export const usersProviders = [
  {
    provide: "USER_MODEL",
    useFactory: (mongoose: Mongoose) => mongoose.model("Users", UsersSchema),
    inject: ["DATABASE_CONNECTION"],
  },
];
```

Модель підключається до відповідного сутності репозиторію. Оскільки ми використовуємо бібліотеку mongoose, то завдяки цьому кожна модель може викликати багато різноманітних методів, наприклад find() – для пошуку потрібного документу, update() – для оновлення документу, save() – для

збереження нового документу та багато інших. Всі ці методи значно зпрощують маніпуляції з БД.

Для кодування поролю користувачів використовується бібліотека bcrypto. Приклад використання:

```
async createUser({ email, password }: CreateUserDto): Promise<boolean> {
  const isAlreadyCreated = await this.findOneByEmail(email);

  if (isAlreadyCreated) {
    throw new BadRequestException({
      message: "User with this email is already registered",
    });
  }

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

  await this.usersRepository.createUser({ email, password: hash });
  return true;
}
```

Для кожної з сутностей чи цілей створено власний модуль. Наприклад для авторизації користувачів створено AuthModule.

Пропоную розглянути реалізацію процесу авторизації. Nest підтримує JWT авторизацію, тому є спіціальні методи задопомогою яких можна сгенерувати підпис(токен) на основі даних користувача. Підпису надсилається на клієнтську частину у відповіді на запит авторизації. Потім цей підпис зберігається на стороні клієнта та при необхідності звернення до сервера відправляється в кожному запиті. Після чого сервер отримує даний підпис та розкодовує його відповідно ключа який зберігається на сервері. Таким чином сервер може розуміти який із клієнтів є авторизованим а який ні. Реалізація сервісу авторизації:

```
@Injectable()
export class AuthService {
  constructor(
    private usersService: UsersService,
    private jwtService: JwtService
```

```
  ) {}
  async login({ email, password }: LoginUserDto) {
    const user = await this.usersService.findOneByEmail(email);
    if (!user) {
      throw new BadRequestException({ message: "User not found" });
    }
    const isMatch = await bcrypt.compare(password, user.hash);
    if (!isMatch) {
      throw new UnauthorizedException({
        message: "Email or password is not correct",
      });
    }
    return {
      accessToken: this.jwtService.sign(user),
    };
  }
```

Для контролю чи є користувач авторизованим чи ні, реалізовано спеціальний клас JwtStrategy:

```
@Injectable()
export class JwtAuthGuard extends AuthGuard('jwt') {
  canActivate(context: ExecutionContext) {
    return super.canActivate(context);
  }

  handleRequest(err, user) {
    if (err || !user) {
      throw err || new UnauthorizedException();
    }
    return user;
  }
}
```

Даний клас використовується як декоратор @UseGuards(JwtAuthGuard) для тих методів котроллерів які повині бути доступними тільки для авторизованих користувачів. Приклад використання:

```
  @ApiOperation({ summary: "Get profile" })
  @ApiBearerAuth()
  @UseGuards(JwtAuthGuard)
  @Get("profile")
  getProfile(@Request() req) {
    return req.user;
  }
```

Для можливості налаштування серверного додатку використовується ConfigModule. Передбачена можливість налаштування серверного додатку за допомогою файл оточення .env. За допомогою нього можна налаштувати ключ для кодування підписів, порт та хост серверу, час життя підпису, посилання для підлючення до БД. Приклад файлу з налаштуваннями:

```
export default () => ({
  host: process.env.HOST || "localhost",
  port: parseInt(process.env.PORT, 10) || 3000,
  accessTokenSecret:
    process.env.ACCESS_TOKEN_SECRET || "YqBvBZSizLlCtGe_-s1bpUBk",
  accessTokenLife: parseInt(process.env.ACCESS_TOKEN_LIFE, 10) || "30m",
  database: {
    connectionString:
      process.env.DB_CONNECTION_STRING ||
      "mongodb://127.0.0.1:27017/project-db",
  },
});
```

Для реалізації перевірки даних на коректність використовується бібліотека hapi/joi. За допомогою ней ми перевіряємо дані на коректність та автоматично формуємо повідомлення для помилок. Приклад використання:

```
export const CreateTaskSchema = Joi.object().keys({
  title: Joi.string().required(),
  description: Joi.string().required(),
  type: Joi.string().required(),
  linkForCheck: Joi.string(),
  numberOfAttempts: Joi.number(),
  timeLimit: Joi.number(),
  showAnswers: Joi.boolean(),
});

@ApiOperation({ summary: "Create task" })
  @ApiBearerAuth()
```

```
@UseGuards(JwtAuthGuard)
@Post("create")
createTask(
  @Request() req,
  @Body(new JoiValidationPipe(CreateTaskSchema)) body: CreateTaskModel
) {
  return this.tasksService.createTask(body, req.user._id);
}
```

Також було реалізовано надсилання сервером помилок з необхідними кодами та текстом, дл подальшого відображення на клієнтський частині. Для цього були використані класи: BadRequestException – для некоректних запитів та для помилок валідації, UnauthorizedException – для не авторизованих зверень на захищені адреси. Приклад використання:

```
async login({ email, password }: LoginUserDto) {
  const user = await this.usersService.findOneByEmail(email);

  if (!user) {
    throw new BadRequestException({ message: "User not found" });
  }

  const isMatch = await bcrypt.compare(password, user.hash);

  if (!isMatch) {
    throw new UnauthorizedException({
      message: "Email or password is not correct",
    });
  }

  return {
    accessToken: this.jwtService.sign(user),
  };
}
```

Всі реалізовані модулю підключені до головного модулю. Для запуску серверного додатку реалізовано функцію init(). Код функції запуску:

```
async function init() {
  const app = await NestFactory.create(AppModule);
  const configService = app.get(ConfigService);

  app.setGlobalPrefix('api');
```

```
  const config = new DocumentBuilder()
    .setBasePath('api')
    .setTitle("API docs")
    .setDescription("The API description")
    .setVersion("1.0")
    .addBearerAuth()
    .build();

  const document = SwaggerModule.createDocument(app, config);

  SwaggerModule.setup("api/docs", app, document);

  await app.listen(configService.get("port"));
}

init();
```

Окрім запуску серверу в функції init() здійснюється підключення Swagger – це бібліотека яка допомагає автоматично згенерувати документацію для серверного додатку після запуску серверу. Документацію можна відкрити за допомогою браузера. Документація надає можливість переглядати конструкції всіх запитів та здійснювати їх. Ц. документацію зручно використовувати для відправки тестових запитів та тестування серверного додатку(рис. 3.1., рис. 3.2.).

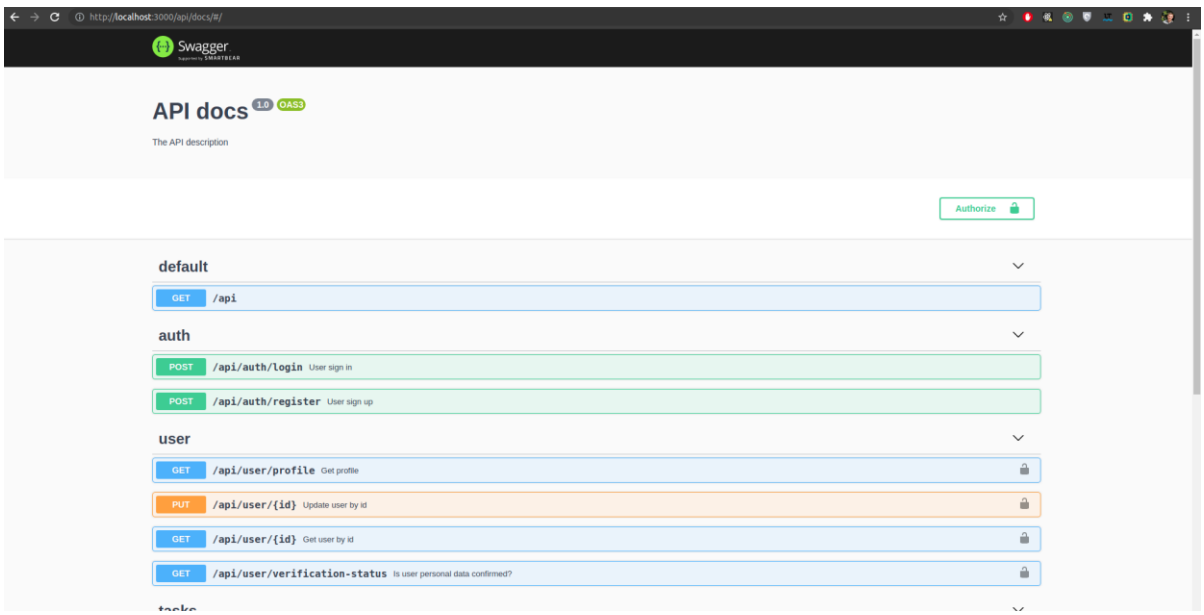

Рис. 3.1. Згенерована документація

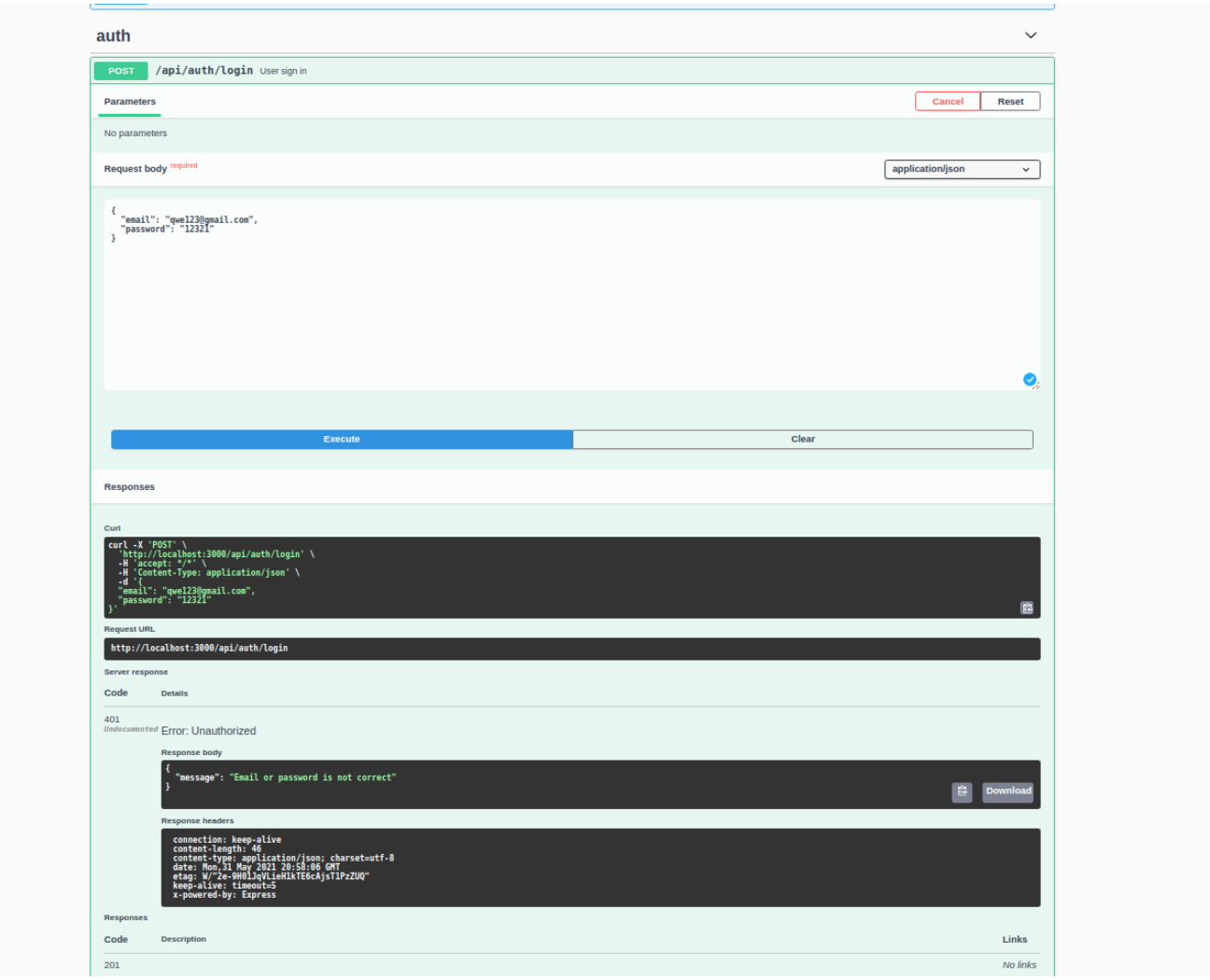

Рис. 3.2. Відправка тестового запиту

3.2.3. Розробка клієнтського додатку.

Клієнтська часнита додатку розроблена засобами мови програмування Javascript та бібліотеки React. Дана бібліотека дає можливість створювати компоненти - в спрощеному варіанті це функції що повертають розмітку. Є декілька основних класифікацій компонентів: розподіл на класові та функціональні, на складні та прості(чисті). Приклад компоненту:

```
import React from 'react';

import Input from '@xcritical/input';

import { ErrorWrapper, Label } from '../../components';
```

```
export const InputField = ({ error, label, ...rest }) => (
  <>
    {label && <Label>{label}</Label>}
    <ErrorWrapper error={error}>
      <Input autoComplete="off" {...rest} />
    </ErrorWrapper>
  </>
);
```

Для відображення інтерфейсу користувача на веб сторінці використовується метод render(). Він формує розмітку сторінки та вставляє її в корневий елемент сторінки, цей процес називається рендером сторінки. Реалізація виклику методу для рендеру сторінки:

```
ReactDOM.render(
  <Provider store={store}>
    <App>
      <Switch>
        <Route exact path={PathNames.login} component={LoginPage} />
        <Route
          exact
          path={PathNames.registration}
          component={RegistrationPage}
        />
        <Route exact path={PathNames.confirmLogin} component={ConfirmLogin} />
        <Route exact path={PathNames.root} component={Dashboard} />
        <Route exact path={PathNames.tasks} component={Tasks} />
        <Route exact path={PathNames.results} component={Results} />
        <Route exact path={PathNames.profile} component={Profile} />
        <Route exact path={PathNames.docs} component={Docs} />
        <Route
          exact
          path={PathNames.completeRegistration}
          component={CompleteRegistration}
        />
      </Switch>
    </App>
  </Provider>,
  document.getElementById('root')
);
```

Для підключення сторінон на певні адреси використовується компонент Route

Для контролю рендеру та збереження даних використовується бібліотека Redux. Вона дає можливість створити структуроване сховище. Щоб підключити сховище. Для того щоб створити сховище використовується метод.

Струкрута сховища та його створення:

```
export const rootReducer = {
  [appReducerNamespace]: appReducer,
  [authReducerNamespace]: authReducer,
  [completeRegistrationReducerNamespace]: completeRegistrationReducer,
  form: formReduce,
  [tasksReducerNamespace]: tasksReducer,
  [dashboardReducerNamespace]: dashboardReducer,
  [errorReducerNamespace]: errorReducer,
  [resultsReducerNamespace]: resultsReducer,
  [profileReducerNamespace]: profileReducer,
};

const store = createStore(rootReducer);
```

Для кращого розуміння структури сховища пропоную розглянути одну з його частин:

```
import { createAction, createSlice } from '@reduxjs/toolkit';

export const initialState = {
  isReady: false,
};

const STORE_NAME = '@auth';

const authStore = createSlice({
  name: STORE_NAME,
  initialState,
  reducers: {
    setIsReady(state, action) {
      state.isReady = action.payload;
    },
  },
});

export const GOOGLE_LOGIN = ${STORE_NAME}/googleLogin;
export const LOCAL_LOGIN = ${STORE_NAME}/localLogin;
```

```
export const LOCAL_REGISTRATION = ${STORE_NAME}/localRegistration;

export const authActions = {
  ...authStore.actions,
  localLogin: createAction(LOCAL_LOGIN),
  localRegistration: createAction(LOCAL_REGISTRATION),
  googleLogin: createAction(GOOGLE_LOGIN),
};

export const authReducer = authStore.reducer;
```

Для обробки всієї так званої бізнес логіки використовуються бібліотека Saga, вона дає можливість при виклику певного методу здійснити певні операції, наприклад зробити запит на сервер, чи відфільтрувати та зберегти дані до сховища та багато іншого.

Структура та підключення саги:

```
export function* rootSaga() {
  yield all([
    watchApp(),
    watchAuthConfirmationSaga(),
    watchAuthSaga(),
    watchCompleteRegistration(),
    watchTasks(),
    watchDashboard(),
    watchResults(),
    watchProfile(),
  ]);
}

store.runSaga(rootSaga);
```

Голова ідея є в тому що в сагах описана вся основна логіка, виклики функцій для зміни сховища, що веде до перерендеру сторінки. Тобто процес відображення інтерфейсу та його оновлення працює за таким принципом: наприклад ми відображаемо сторінку авторизації, припустимо що користувач вводить данів полях форми, в даному випадку викликликаються функції які відповідають за обробку саме цього процесу, ці функції викликають методи для зміни сховища, в результаті чого сховище змінюється і це призводить до оновлення сторінки і відображенні в полях форми нових значень. Якщо користутувач натиснув на

копку авторизації, то в той момент відбувається виклик функції що запускає сагу в який виконується запит на сервер з подальшою обробкою різноманітних випадків.

Приклад компоненту авторизації наведено в додатку В п.1.

Підключення та реалізація саги авторизації:

```
export function* watchAuthSaga() {
  yield takeEvery(GOOGLE_LOGIN, handleGoogleLogin);
}

export function* handleLocalLogin() {
  try {
    const { email, password } = yield select(authSelectors.getLoginFormData);

    const requestModel = {
      email,
      password,
    };

    const errors = validateLogin(requestModel);

    if (errors) {
      yield put(xcriticalFormShowErrors(LOGIN_FORM_NAME, true));
      yield put(xcriticalFormError(LOGIN_FORM_NAME, errors));
      return;
    }

    const { accessToken } = yield authApi.localLogin(requestModel);

    localStorage.setItem(ACCESS_TOKEN_NAME, accessToken);

    yield put(appActions.checkVerificationStatus());

    NotificationManager.success('User successfully sign in');
  } catch (e) {
    NotificationManager.error(JSON.parse(e.message).message);
    console.error(e);
  }
}
```

Компоненти можна робити універсальними, такий компонент можна перевикористовувати в різних частинах проекту. Окрім цього існує багато

бібліотек з реалізованими універсальними компонентами, в даному проекті ула використана бібіотека xcritical. В ній реалізовані багато різноматних компонентів, із яких були використані: Button, Modal, Input, Checkbox, Select, Popover. Ці компоненти були покращені в рамках даного проекту. Для полів які використовуються на формах було додано можливість задати стилізований підпис для поля, також булу реалізовано вспливаючий компонент для відображення помилок валідація для полей форми. Реалізація компоненту для помилок:

```
export const ErrorWrapper = ({ children, error }) => (
  <Popover
    appearance="error"
    position="top right"
    content={error}
    visible={!!error}
  >
    {children}
  </Popover>
);
```

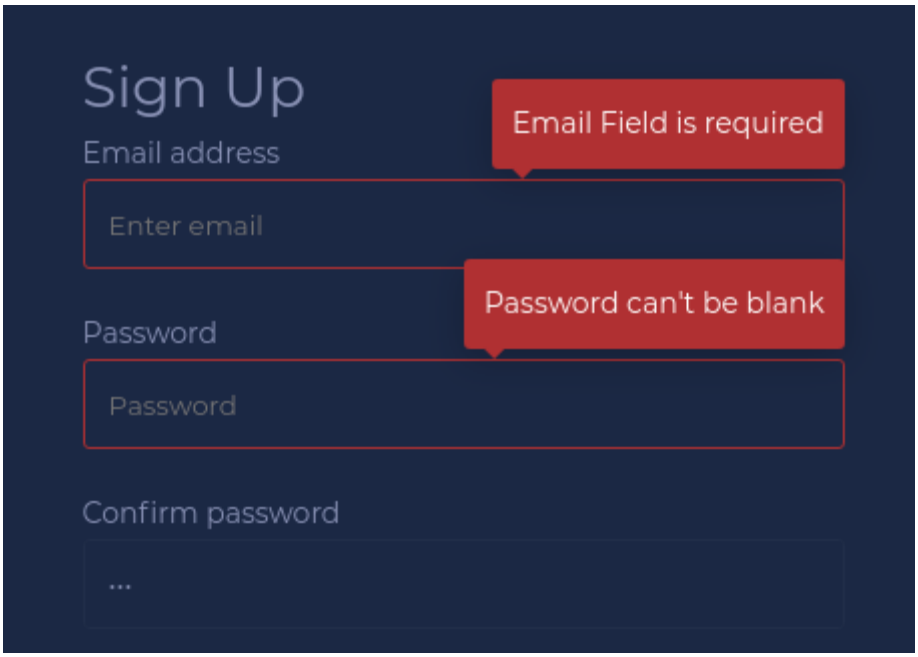

Рис. 3.3. Форма реєстрації з відображеними помилками валідації даних

Для стилізації універсальних компонентів було створено тему – об’єк з описаними стилями для кожного виду компоненті. Для більшості комопонентів можна задати вид, передавши параметр appearance, завдяки цьому стало

можливим описати в темі декільга виглядів кожного із компонентів та за потребі передавати назву виду як параметр до компоненту. Реалізація теми:

Функція створення теми для всіх універсальних компонентів в декількох видах подано в додатку В п.2.

Функція створення теми компоненту кнопки подано в додатку В п.3.

Приклад створення декількох видів кнопки:

```
export const defaultButton = createButtonTheme({
  background: DefaultColors.lightBlue,
  borderColor: DefaultColors.lightBlue,
  color: DefaultColors.white,
  hoverColor: DefaultColors.white,
  borderHoverColor: DefaultColors.blue,
  hoverBackground: DefaultColors.blue,
});

export const secondaryButton = createButtonTheme({
  background: DefaultColors.secondBlue,
  borderColor: DefaultColors.lightBlue,
  color: DefaultColors.lightBlue,
  hoverColor: DefaultColors.white,
  borderHoverColor: DefaultColors.blue,
  hoverBackground: DefaultColors.blue,
});
```

За схожим принципом були стилізовані всі компоненти, тобто для кожного з універсальних компонентів була реалізована функція, що створює тему даного компоненту відповідно до переданих кольорів, потім за допомогою цієї функції було створено декількавиглядів компоненту, а потім ці варіати теми компоненту булі підключені до загальної теми, яка потім за допомогою компоненту ThemeProvider огортує головний компонент.

Для стилизіцї окремих елементів було використано бібліотеку styled-components. Вона дая можливість створювати компоненти на основі HTML тегів з миттєвим присвоєнням стилів. Створені компоненти можуть очікувати різноманітні параметри і в залежності від цього змінювати свої характеристики. Приклад створення та використання такого компоненту:

```
export const Text = styled(LabelWrapper)`
```

```
  font-size: ${({ size }) => size || '14px'};
  word-break: break-word;
  display: flex;
  align-items: center;
  color: ${({ color }) => color || DefaultColors.lightPink};
  font-weight: ${({ weight }) => weight || 400};
`;

<Text size="18px">{title}</Text>
```

Для реалізації валідації форм було використано бібліотеку validate.js. Функції валідації зазвичай викликаються в сагах, перед відправкою запитів на сервер, дані перевіряється і якщо щось введено не коректо но функція валідації повертає текст помилки який зберігається в сховищі та відображається користувачу. Приклад функції валідації реалізованої засобами бібліотеки:

```
import { validate as valid } from 'validate.js';

import { formatError } from '../../../../packages';

const constraints = {
  email: {
    presence: {
      message: 'Field is required',
    },
    email: {
      message: 'is not valid',
    },
  },
  password: {
    presence: true,
    length: {
      minimum: 6,
      message: 'must be at least 6 characters',
    },
  },
};

export const validate = (data) => {
  const errors = valid(data, constraints);

  return errors ? formatError(errors) : null;
};
```

Оскільки браузери не підтримують React у чистому вигляді, то для виконная коду налаштовується зборщик проекту який перетворює код до формату який підтримує браузер. Для цього проекту було використано Webpack. Налаштування webpack подано в додатку В п.4

Для автоматичного форматування написаного коду відповідно обраним правилам було налаштовано eslint. Завдяки цьому код зручно читати, він автоматично форматується вказаним правилам. Налаштування eslint подано в додатку В п.5.

Для реалірації анімації заднього фону було використано технології canvas.(див. додаток В п.6).

3.3 Інструкція користувача

При запуску програми з'являється сторінка авторизації, де користувач може перейти ввести пароль та пошту та ввійти до системи, нові користувачі можуть перейти до форми реєстрації натиснувши «Sign out».

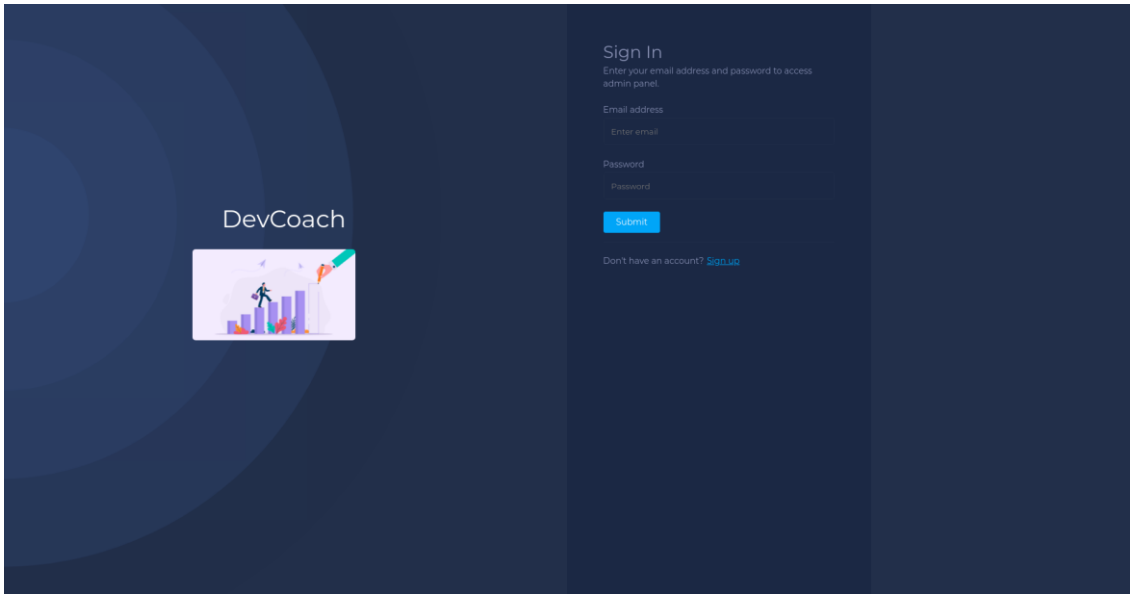

Рис. 3.4. Сторінка авторизації

Після входу, користувач потрапляє на сторінку «Complete Registration». Він мусить заповнити інформацію про себе, іначе він не потрапить до системи.

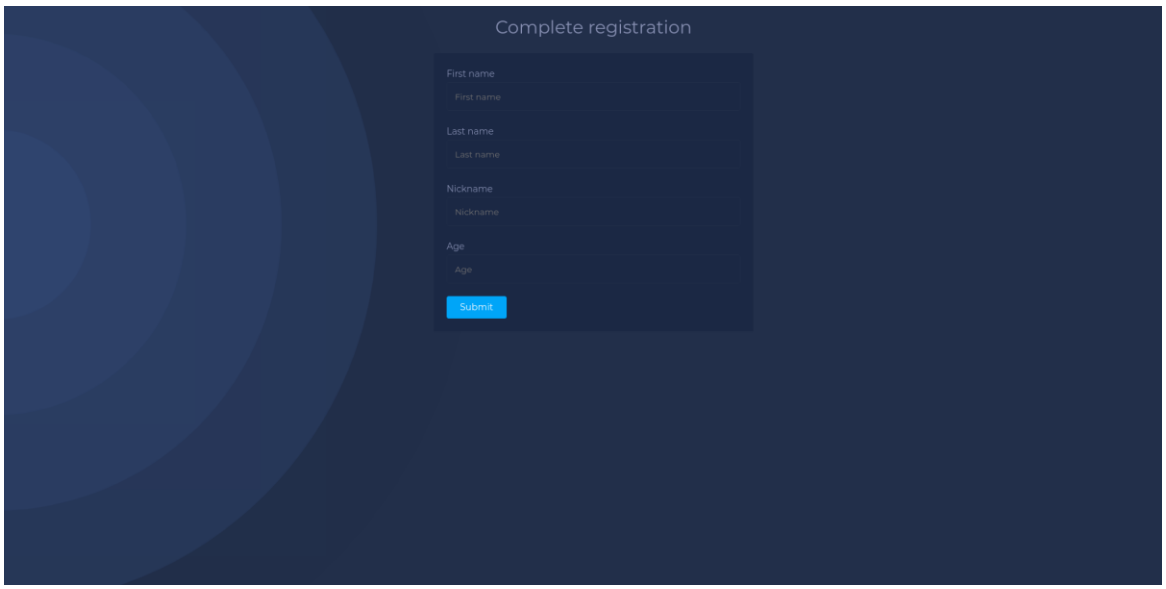

Рис. 3.5. Сторінка Complete Registration

Після успішного заповнення особистих даних користувач потрапляє на сторінку «dashboard» Окрім цього він бачить головне меню, за допомогою якого він може перейти на такі вкладки:

1. Dashboard
2. Tasks
3. Results
4. Profile
5. Integration API
6. Log out

На сторінці «dashboard» користувач бачить невеликий блок статистики, може переглянути, відредагувати, створити чи видалити власні завдання. Також користувач може відфільтрувати створені завдання за типом.

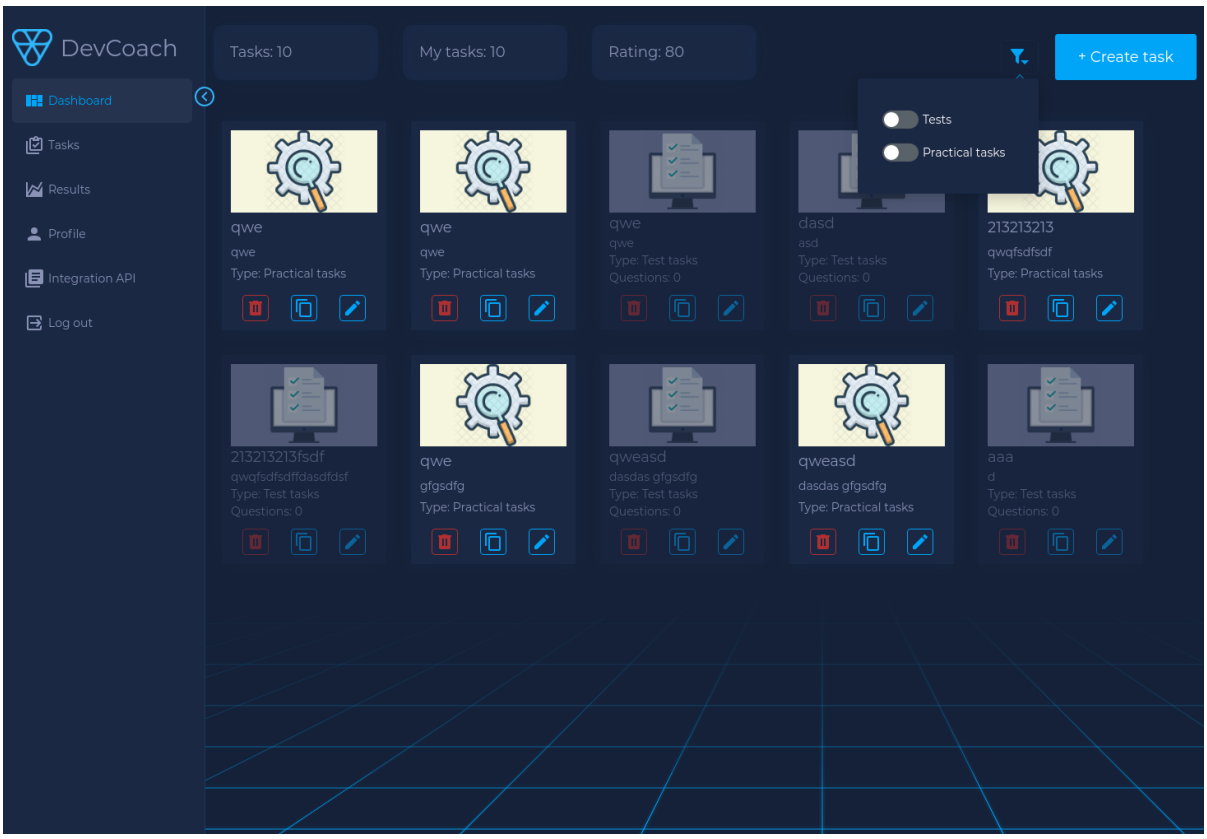

Рис. 3.6. Сторінка Dashboard

Натиснувши на кнопку «Create task» користувач переходить на форму створення завдання.

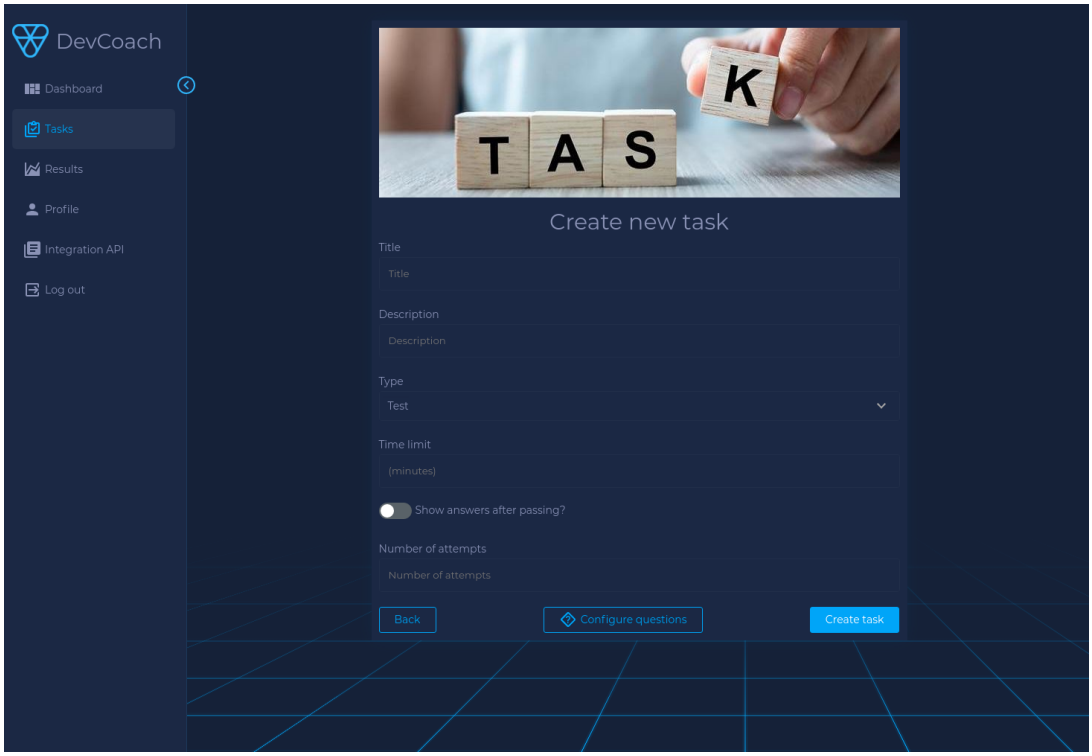

Рис. 3.7. Форма створення завдання

Якщо обрати в полі Type значення Test, знизу з’явиться додаткова кнопка та система запропонує налаштувати питання, якщо в полі Type обрати значення Practical task з’явиться додаткове поле, де потрібно буде вписати url-адресу на сервіс перевірки даного завдання.

Пропоную розглянути форму налаштування запитань для тестового завдання. Користувач може ввести текст запитання, та обрати варіант відповіді: звичайне поле(input), один правильний варіант(radioButton), декілька варіантів відповіді(checkbox). Також при створенні завдання користувач вписує правильну відповідь. При створенні radioButton чи checkbox, користувач може додати варіанти відповіді за допомогою спеціального інтерфейсу, та вибрати правильні відповіді просто натиснувши на них.

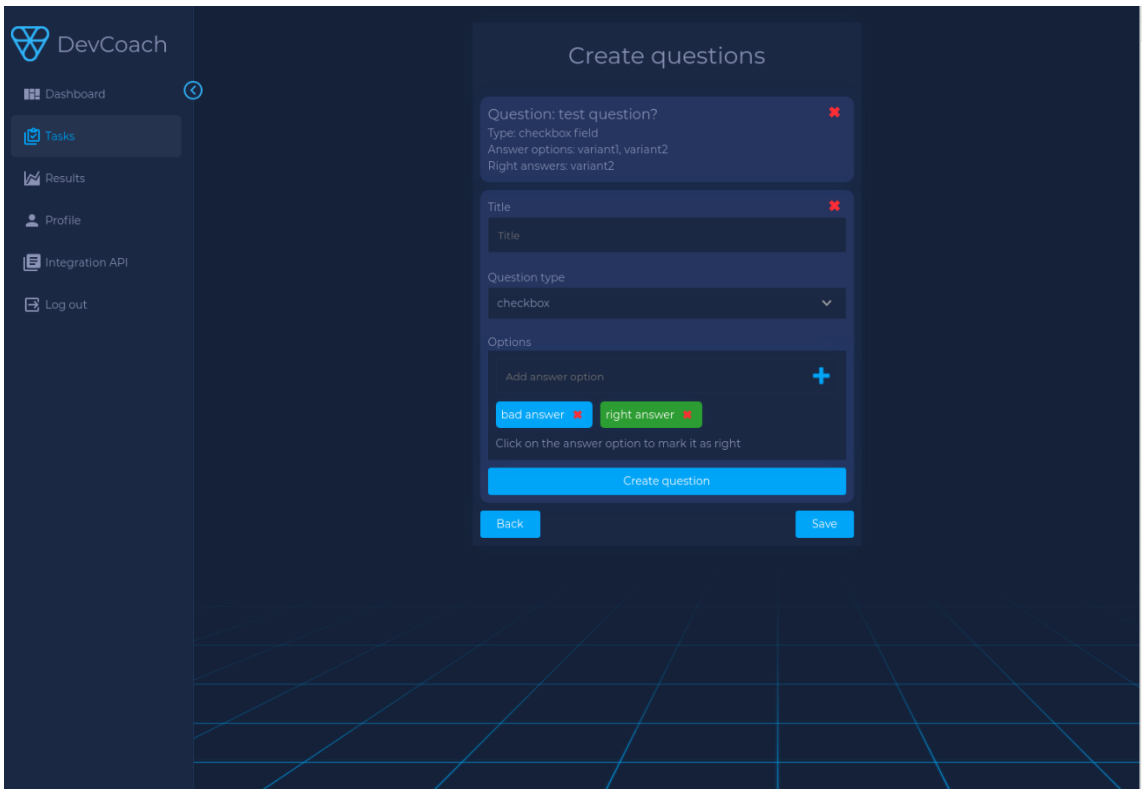

Рис. 3.8. Форма створення запитань для тестового завдання

Скориставшись меню користувач може перейти на сторінку «Tasks», де він може переглянуті створенні завдання, не тільки свої, але і інших користувачів, може відфільтрувати завдання за допомогою фільтрів чи знайти потрібне завдання за допомогою пошуку. Натиснувши на обране завдання користувач в модальному вікні може переглянути додаткову інформацію та почати проходження завдання.

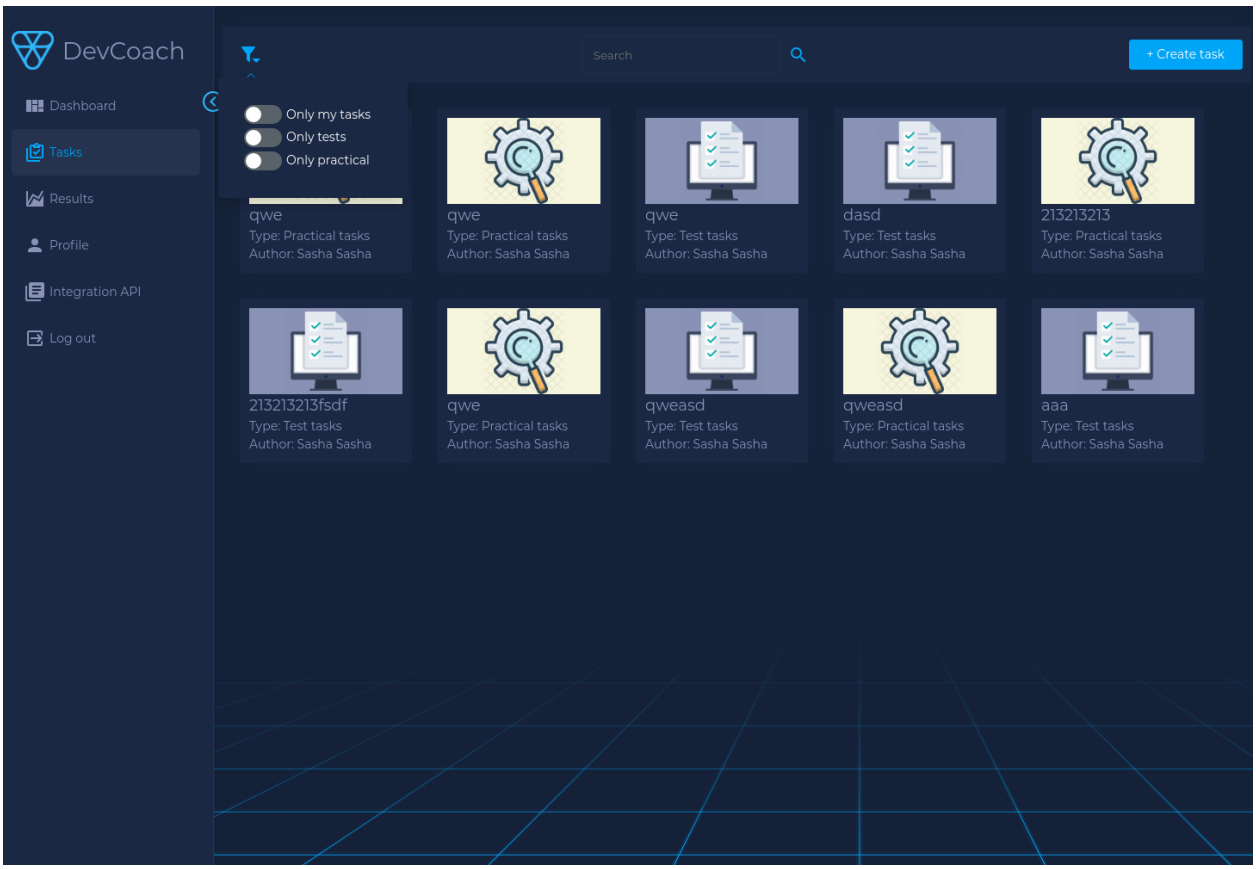

Рис. 3.9. Сторінка Tasks

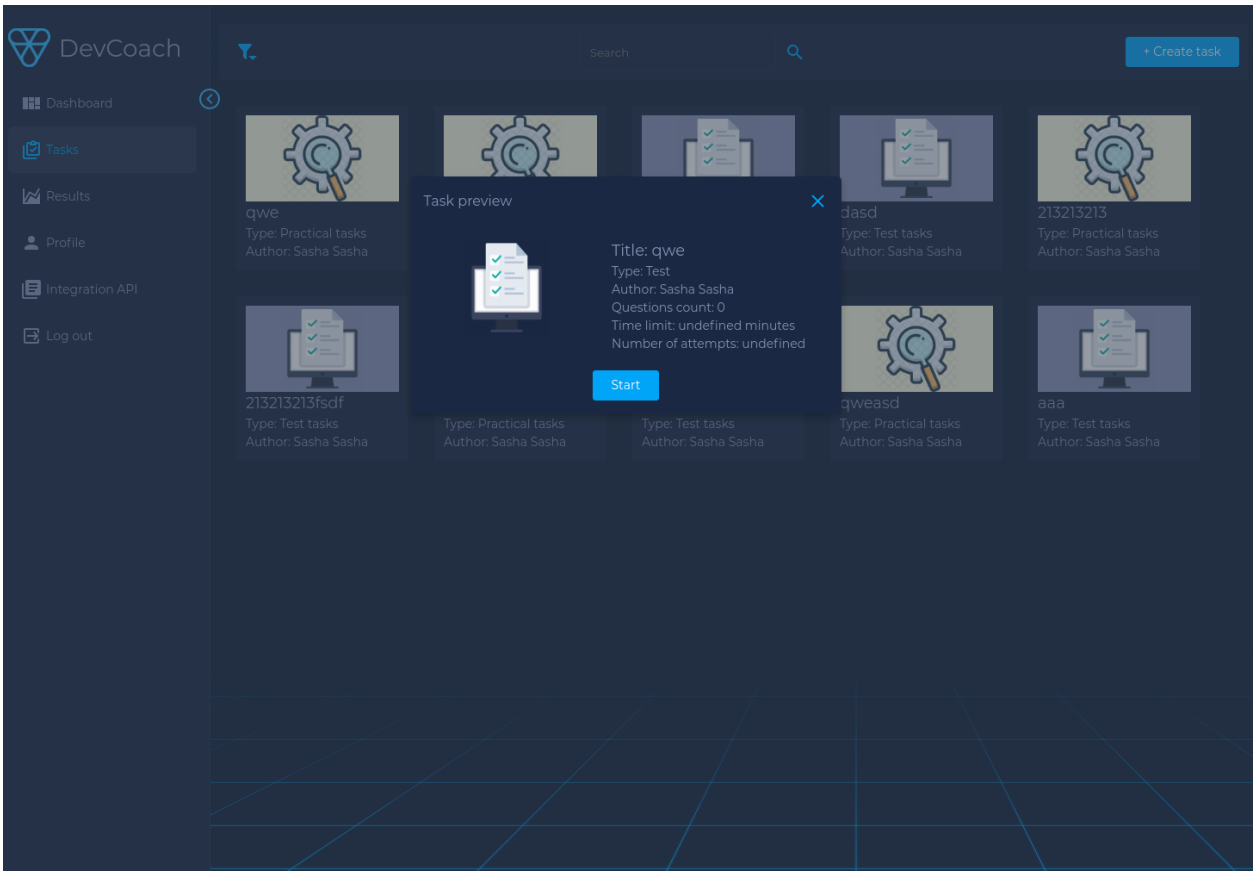

Рис. 3.10. Перегляд обраного завдання

Якщо користувач обрав тестове завдання та почав тестування натиснувши кнопку «Start» він потрапляє на сторінку тесту де він мусить відповісти на запитання, та натиснути кнопку «Submit» щоб перевірити відповіді. Після перевірки відкриється модальне вікно де користувач побачить свій результат.

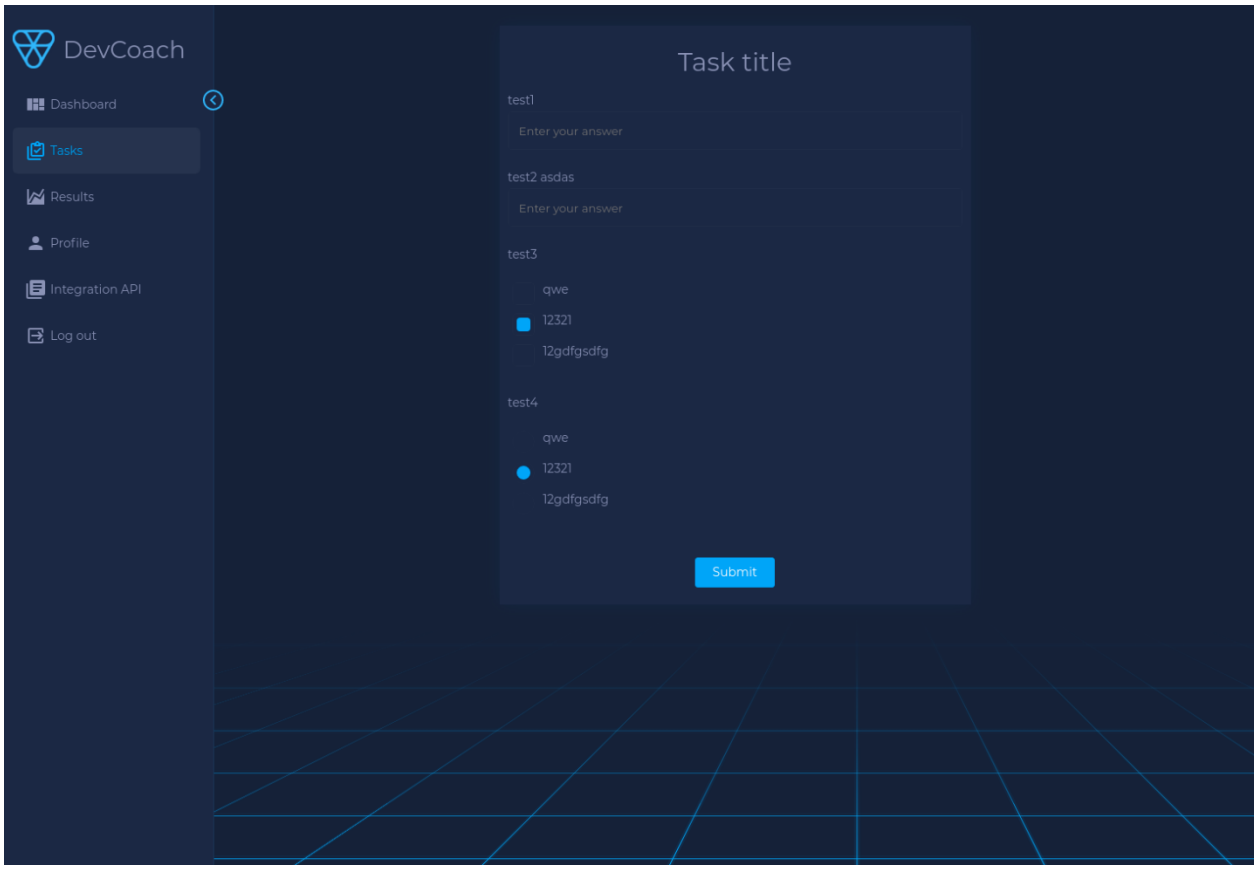

Рис. 3.11. Проходження тесту

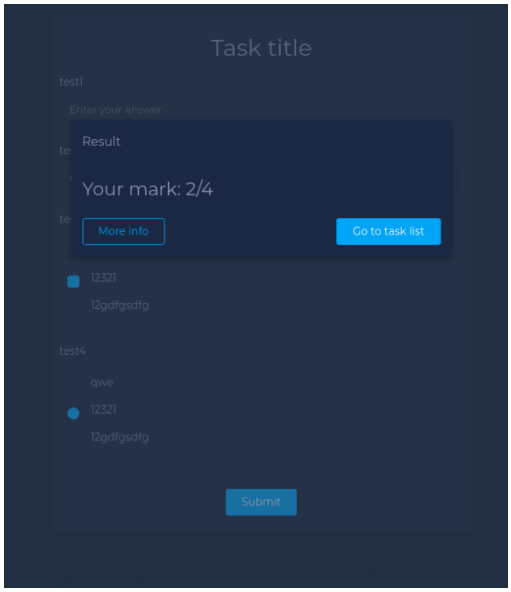

Рис. 3.12. Показ результату після перевірки

Натиснувши на кнопку «More info» користувач може переглянути свої та правильні відповіді.

Якщо користувач обрав та розпочав практичне завдання натиснувши кнопку «Start», то він потрапляє на сторінку де в лівій частині він бачить саме завдання, таймер(час на виконання завдання обмежений), зірочками відображається складність завдання(вона розраховується з оцінок всіх хто проходив дане завдання). Користувач може зупини таймер натиснувши на кнопку «Stop timer». В правій частині реалізовано робоче поле до користувач мусить вписати відповідь. Після завершення, користувач повинен натиснути на кнопку «Check the solution» щоб перевірити відповідь. Після перевірки з'явиться модальне вікно де користувач побачить свою оцінку.

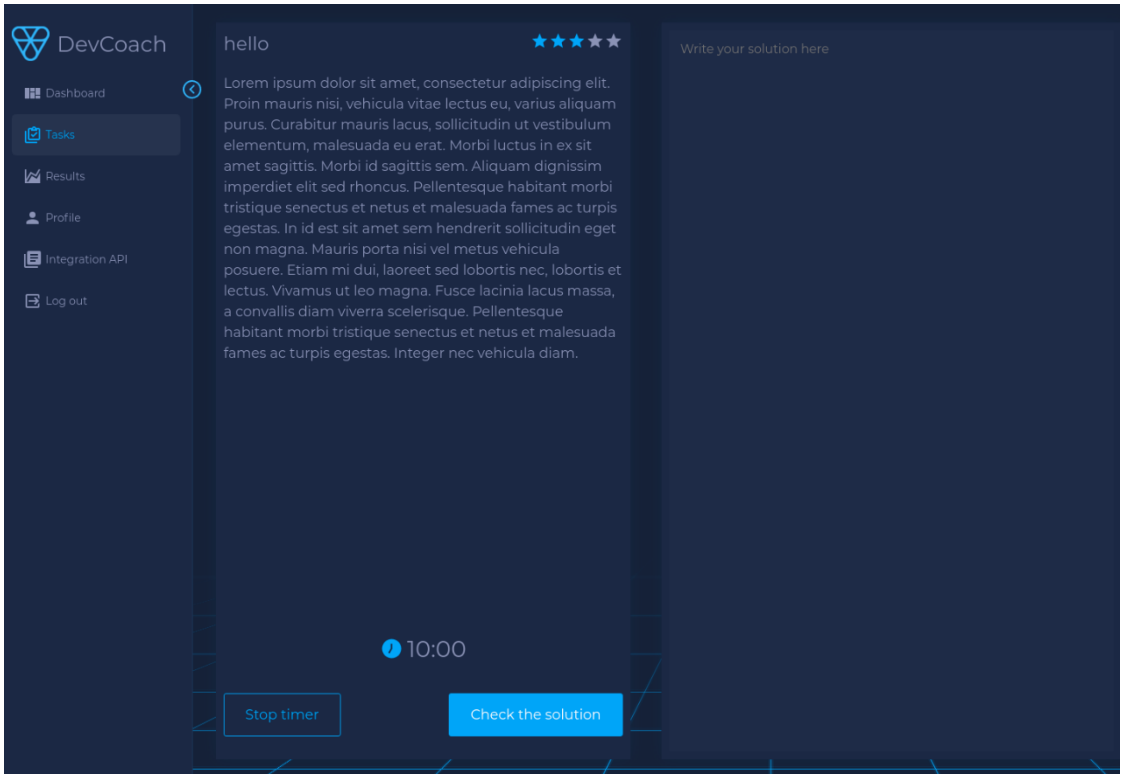

Рис. 3.13. Проходження практичного завдання

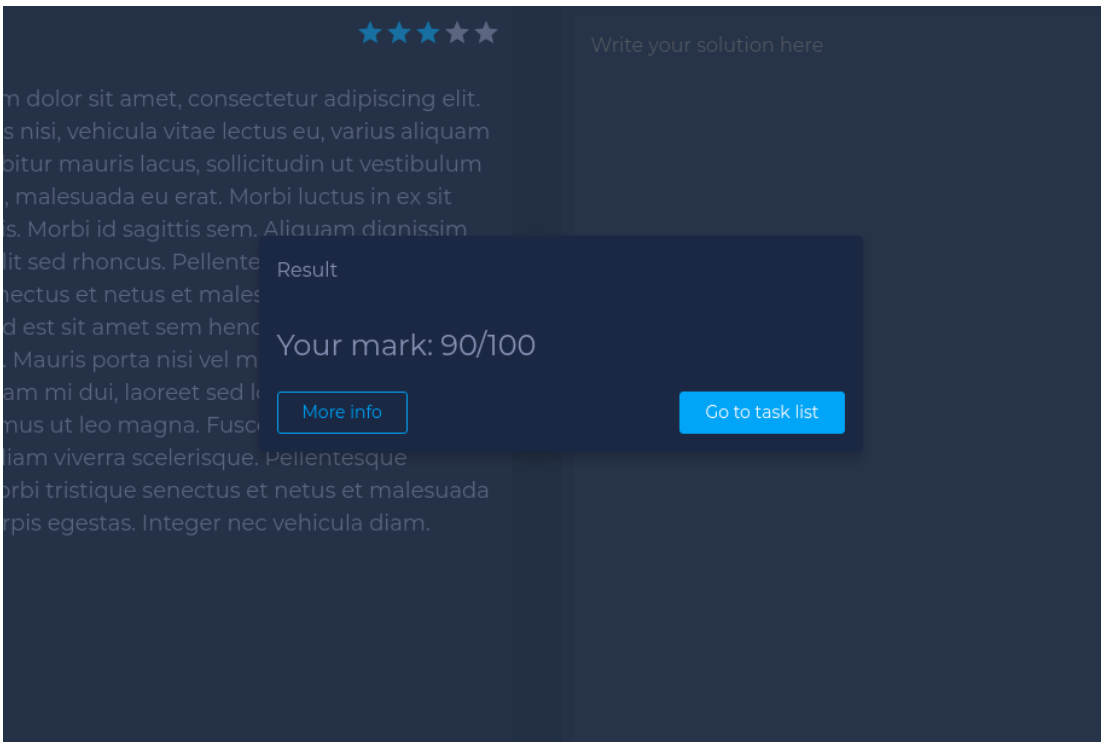

Рис. 3.14. Показ результату

Натиснувши на кнопку «More info» користувач може переглянути свою та правильну відповідь.

На сторінці «Results» користувач може переглянути свої результати, може відфільтрувати завдання за типом. Обравши потрібний результат на натиснувши на нього користувач може переглянути свою та правильну відповідь. Якщо це практичне завдання, то з’явиться модальне вікно, якщо це тестове завдання то відкриється спеціальна сторінка.

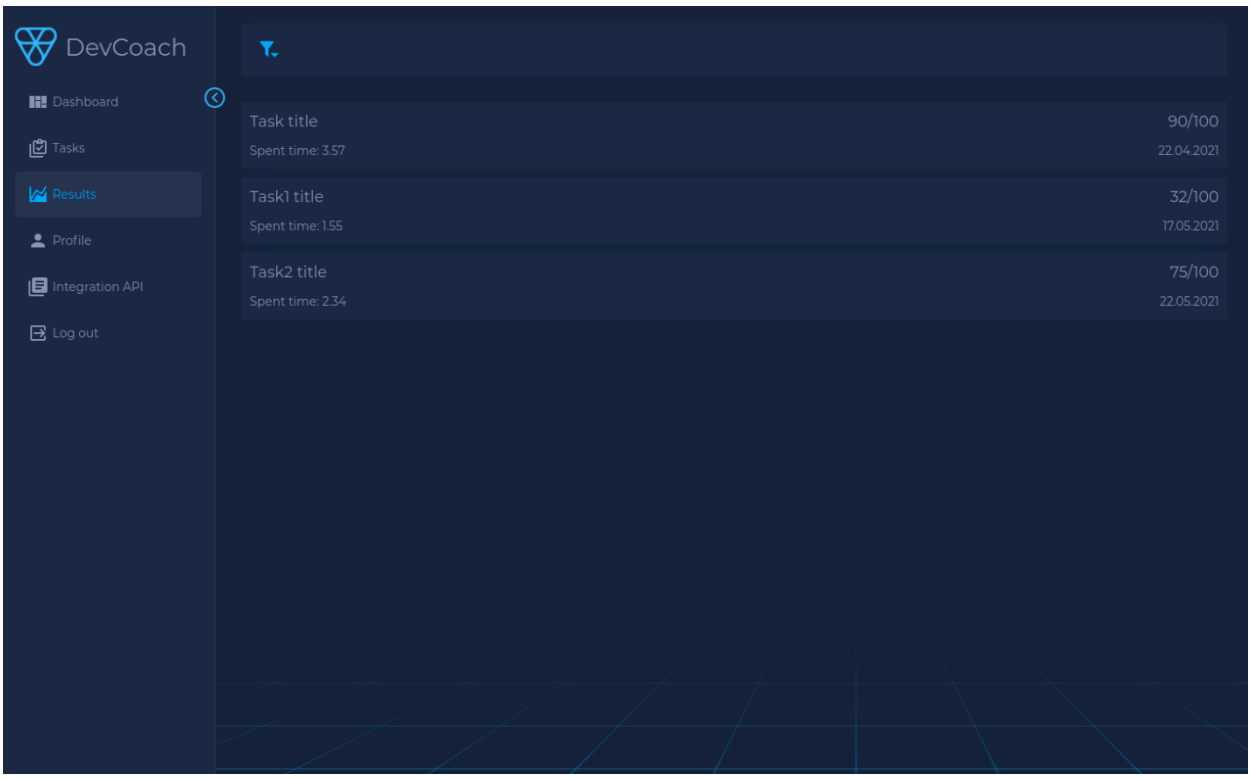

Рис. 3.15. Сторінка Results

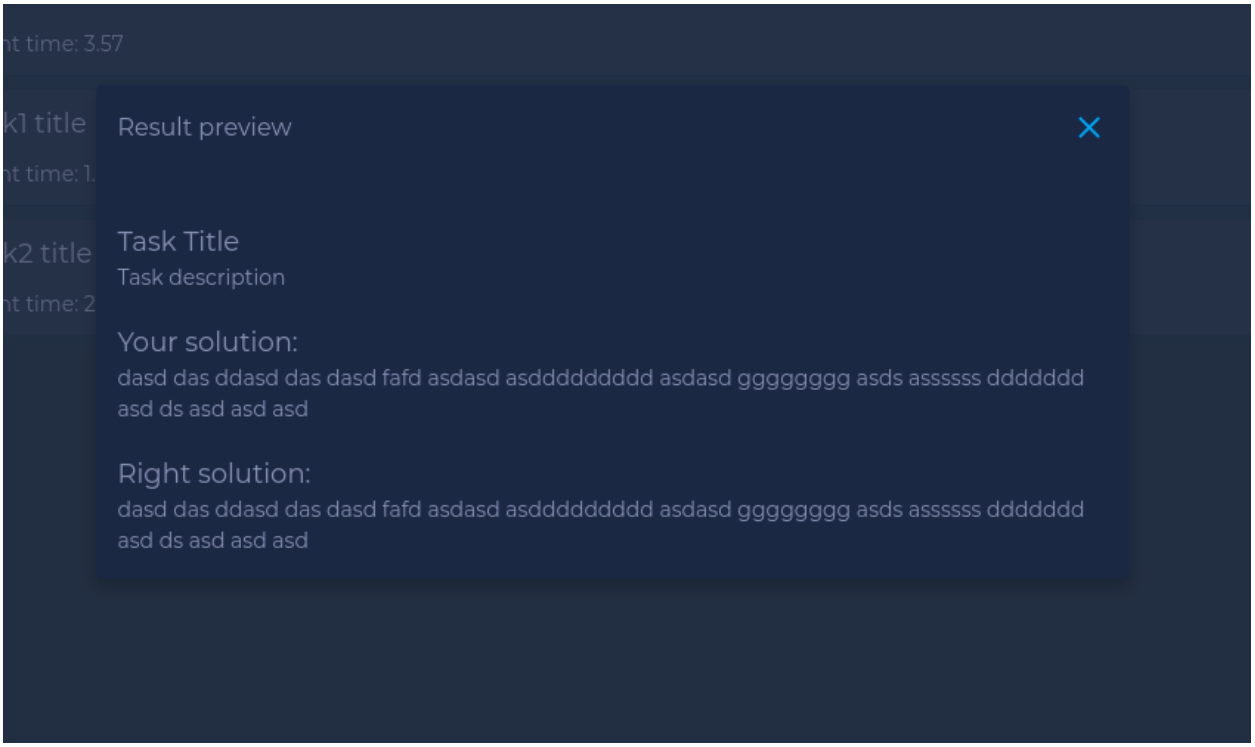

Рис. 3.16. Перегляд результату практичного завдання

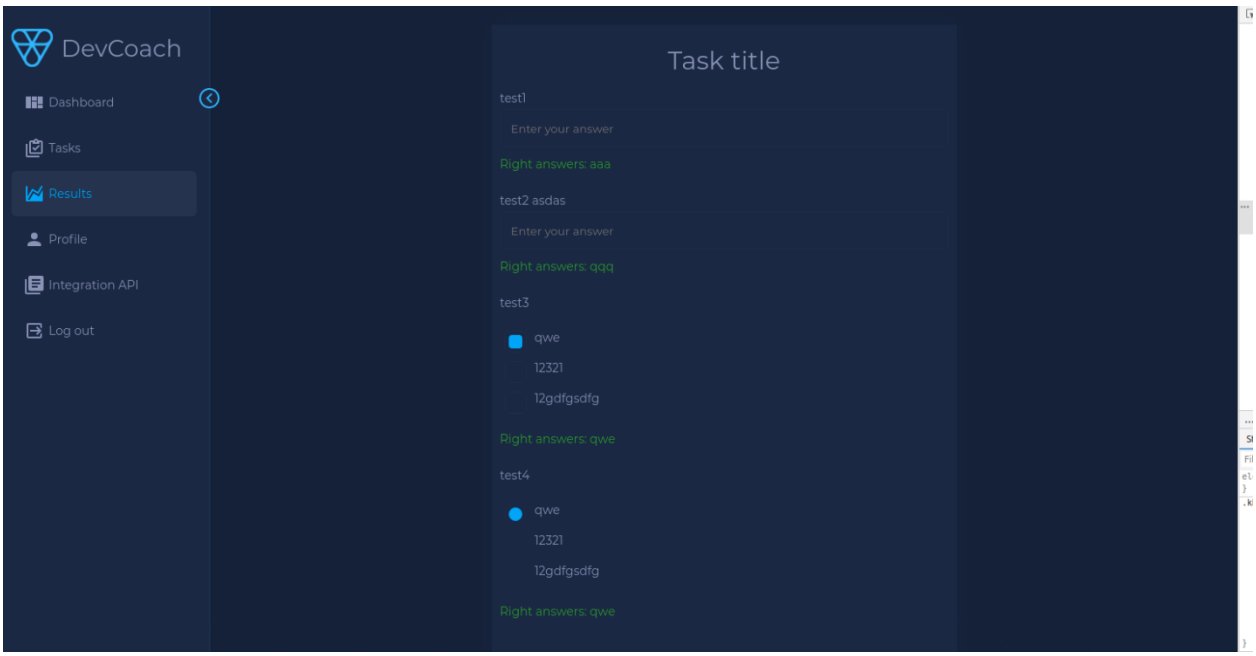

Рис. 3.17. Перегляд результату тестового завдання

Перейшовши на сторінку Profile, користувач може змінити інформацію про себе.

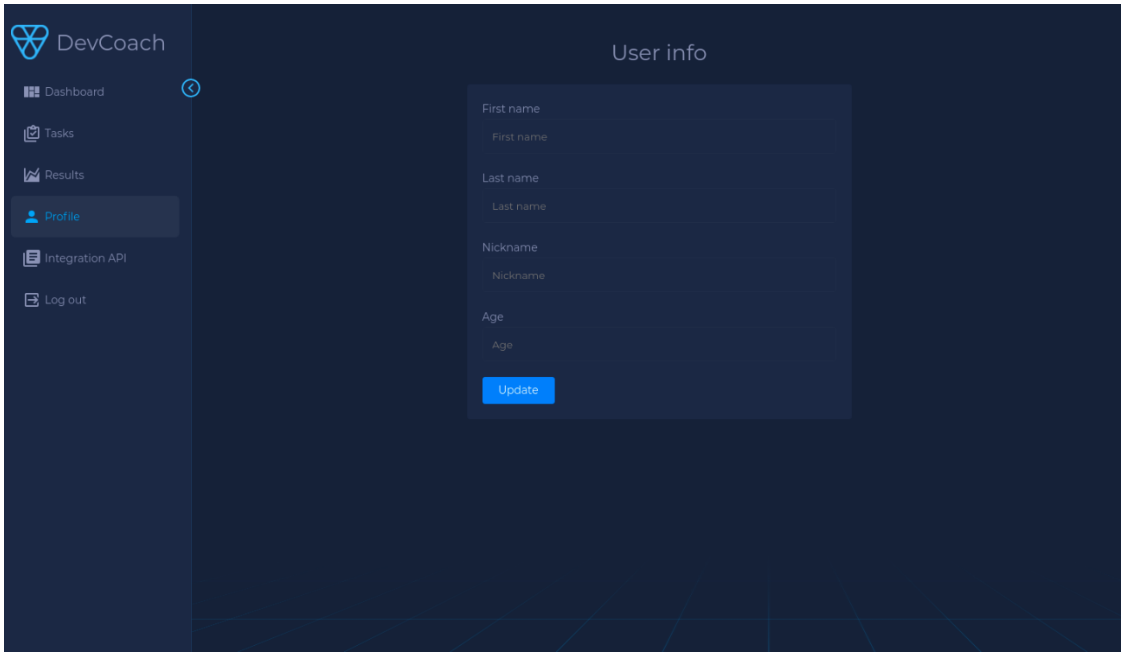

Рис. 3.18. Сторінка Profile

Перейшовши на сторінку Integration API, користувач може ознайомитися всією потрібною інформацією, що потрібна йому для створення власних практичних завдань.

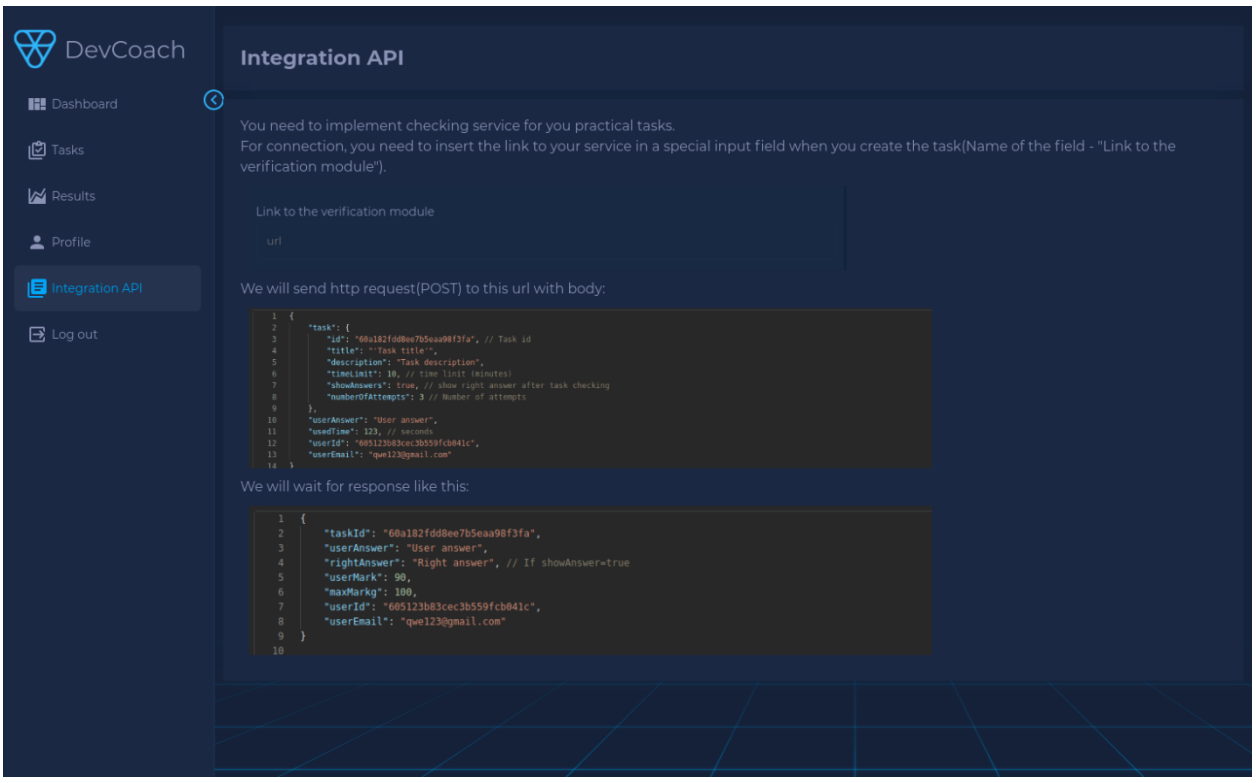

Рис. 3.19. Сторінка Integration API

Натиснувши на пункт меню «Log out» користувач виходить з системи.

3.4. Технічне та системне забезпечення розробки

3.4.1. Обґрунтування вибору засобів для розробки системи

MongoDB – документо-орієнтована система керування базами даних з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів[6].

MongoDB було обрано по декільком причинам. По-перше, ця база даних є дуже зручною в використанні поєднуючи її саме з мовами програмування Javascript та Typescript, оскільки існує багато бібліотеки я роблять маніпуляціє з MongoDB неймовірно зручними, наприклад бібліотека mongoose, яка і використовувалася в рамках даного проекту. По-друге безумовною перевагою є

те, що вона надає можливість легкої роботи з даними у форматі JSON у будь-який частині вашої програми.

MongoCompass – графічний інтерфейс для MongoDB. Дає можливість візуально досліджувати свої дані, виконувати спеціальні запити за лічені секунди. Взаємодія зі своїми даними реалізована з повною функціональністю CRUD. Доступно на Linux, Mac або Windows. Компас дає змогу приймати розумніші рішення щодо індексації, перевірки документації тощо[7].

MongoCompass був використовувася для зручної роботи з MongoDB, а саме задач видалення, редагування, та перегляду даних засобами графічного ынтерфейсу.

WebStorm – інтегроване середовище розробки для JavaScript, HTML та CSS від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA. WebStorm є спеціалізованою версією PhpStorm, пропонуючи підмножину з його можливостей. WebStorm постачається з перед-установленим плаґінами JavaScript (такими як для Node.js), котрі доступні для PhpStorm безкоштовно [8].

WebStorm використовувася як для розробки серверної частини так клієнтської, оскільки, він підтримує мови програмування Javascript та Typescript. Він є нейомвірно зручнім, вміє підсказувати та доповнювати при написанні коду, має можливість довстановлення різноманітних плагінів. Має влаштований термінал для введення команд запуску проекту. Є можливість налаштувати велику кількість комбінацій клавіш які значно пришвидшують розробку, наприклад автоформатування коду.

Для перегляду результату та тестування використувувалися браузери **Google Chrome, Mozilla Firefox, Opera, Microsoft Edge**. Для зручної роботи з бібліотекою Redux був встановлений плагін **ReduxDevTools** для браузеру Google Chrome.

3.4.2. Розробка і обґрунтування стратегії адміністрування системи

Правами адміністратора володіє тальки замовник системи. У веб додатку не передбачено розподілу по правам, адміністратор не моє особливих прав в рамках веб додотку. Особливістю є те, що адміністратор має прямий доступ до бази даних, завдяки цьому він може керувати системою та даними користувачів.

3.4.3. Заходи захисту від несанкціонованого доступу до системи.

Для захищення персональних даних та є необхідність в розробці процесів авторизації. Пароль від особистих кабінетів користувачів повинні зберігатись в базі даних у закодованому вигляді. Реалізація процесів авторизації користувача повинен бути реалізованим на основі стандарту JWT.

РОЗДІЛ 4. ОХОРОНА ПРАЦІ

4.1. Нормативна база.

Сучасний розвиток технічного та технологічного стану виробництва передбачає постійну автоматизацію та оптимізацію виробничих процесів. Сьогодні, напевно, важко уявити компанію, господарська діяльність в якій здійснювалась би без використання комп'ютерної техніки. Через масовий характер робіт, що виконуються працівниками за допомогою комп'ютера, законодавством України чітко врегульовано норми та вимоги до використання комп'ютерної техніки на підприємстві, безпосередньо й охорона праці при роботі з комп'ютером.

Перелік нормативно-правових актів, які регулюють це питання, досить широкий. Наприклад, ст. 21 Кодексу законів про працю України визначає обов'язки роботодавця щодо забезпечення працівникам комфортних та безпечних умов праці, а ст. 13 Закону України «Про охорону праці» закріплює це право з позиції охорони праці [19].

4.2. Вимоги безпеки під час роботи з комп'ютером.

Щодня перед початком роботи оператор повинен: оглянути своє робоче місце: про виявлення ознак пошкодження обладнання інформувати свого безпосереднього керівника; відрегулювати освітленість на робочому місці, переконатися в відсутності відблисків на екрані комп'ютера, відсутності зустрічного світла; перевірити правильність підключення обладнання ЕОМ до електромережі; очистити екран комп'ютера від пилу та інших забруднень; перевірити правильність організації робочого місця й за необхідності провести відповідні коригування.

При виконанні робіт оператор персонального комп'ютера повинен: витримувати відстань від очей до екрана комп'ютером в межах 60 - 70см; дотримуватися внутрішньозмінного режиму праці та відпочинку, регламентованих перерв у роботі, а саме (при 8-годинній денній робочій зміні);

для розробників програм - тривалістю 15 хвилин через кожну годину роботи; інших категорій працівників - тривалістю 15 хвилин через кожні дві години роботи; для операторів комп'ютерного набору - тривалістю 10 хвилин, після кожної години роботи.

4.3. Профілактика вправ для очей.

Наші очі так, як і інші частини нашого тіла, потребують постійного тренування та спеціальних вправ якы підтримують їх здоров`я. Особливо якщо багато часу ви проводите біля екрану.

1. Наближайте та віддаляйте об'єкт від очей.
2. Прикрийте долонею одне око не закриваючи його. Дивіться перемінно на віддалений і ближній об'єкт.
3. Прикрийте долонею одне око не закриваючи його та сконцентруйте свій зір на нерухомому об'єкті.
4. Міцно примружтеся на 5 секунд, потім відкрийте очі і не блимайте 3-5 секунд, вовторіть це 7-8 разів.
5. Фокус на віддаленому об`єкті протягом 10-15 секунд. Потім повільно переведіть погляд на довколишній об`єкт, не рухаючи головою. Фокус знову протягом 10-15 секунд. Потім поверніться на далекий об`єкт.

   Зробіть цю вправу 5 разів.

   Виконуйте ці вправи 2-3 рази на день [18].

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було досліджено процес теоретичної та практичної підготовки IT-спеціалістів. Було проаналізовано та виявленно проблеми в підготовці IT-спеціалістів. Було досліджено системи які частково вирішили знайдені проблеми. На основі проблем були сформовані задачі автоматизацій.

Після проведення системного аналізу було розроблено технічне завдання.

На основі завданять автоматизації та технічного завдання було спроектовано моделі колекцій документів бази даних MongoDB. На основі моделей колекцій документів, в процесі розробки серверної частини додатку, було описано схеми.

Після закінчення розробки серверної та клієнтської частини ми отримали простий, швидки та зручний Web-додаток що вирішив всі завдання автоматизації.

Використання реалізованої інформаційної системи дозволить значно полегшити, пришвидшити та спростити процес теоретичної та практичної підготовки IT-спеціалістів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Загоровська Л. Г., М'якшило О. М., Костіков М. П.* Методичні рекомендації до виконання кваліфікаційної роботи на здобуття освітнього ступеня «Бакалавр» спеціальності 122 «Комп'ютерні науки» освітньо-професійної програми «Комп'ютерні науки» денної та заочної форм навчання. НУХТ, 2020. – 30 с.
2. Офіційний сайт HTML Academy [Електронний ресурс] – Режим доступу: https://htmlacademy.ru/
3. Офіційний сайт JavaRush [Електронний ресурс] – Режим доступу: https://javarush.ru/
4. Офіційний сайт Codewars [Електронний ресурс] – Режим доступу: https://www.codewars.com/
5. Документація по MongoDB [Електронний ресурс] – Режим доступу: https://metanit.com/nosql/mongodb/
6. Інформація про MongoDB [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/MongoDB
7. Офіційний сайт Mongo Compass [Електронний ресурс] – Режим доступу: https://www.mongodb.com/products/compass
8. Офіційний сайт JetBrains [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/ru-ru/
9. *Когаловский М.Р.* Енциклопедія технологій баз даних. 2002. — 800 с.
10. *Девід Фленаган*. JavaScript. Повне керівництво (7-е видання). 2021. – 720 с.
11. *Розенталс Н*. Вивчаємо TypeScript 3. – М.: ДМК Пресс, 2019. – 624 с.
12. Документація по паттернам проектування [Електронний ресурс] – Режим доступу: https://refactoring.guru/ru/design-patterns
13. Документація бібліотеки Mongoose [Електронний ресурс] – Режим доступу: https://mongoosejs.com/
14. Документація бібліотеки React [Електронний ресурс] – Режим доступу: https://ru.reactjs.org/

15. Документація фреймворку Nest.js [Електронний ресурс] – Режим доступу: https://nestjs.com/
16. Документація бібліотеки Validate.js [Електронний ресурс] – Режим доступу: https://validatejs.org/
17. *Жидецький В. Ц.* Основи охорони праці. - Львів: Афіша, 2002. - 320 с.
18. *Пушкін В.* Зарядка для очей ISBN: 978-5-699-50082-6
19. Законодавство України про охорону праці (збірник нормативних документів. У 4 т. - К.: Держнаглядохоронпраці; Основа, 1995.

ДОДАТКИ

ДОДАТОК А «МОДЕЛЬ КОЛЕКЦІЙ БАЗИ ДАНИХ»

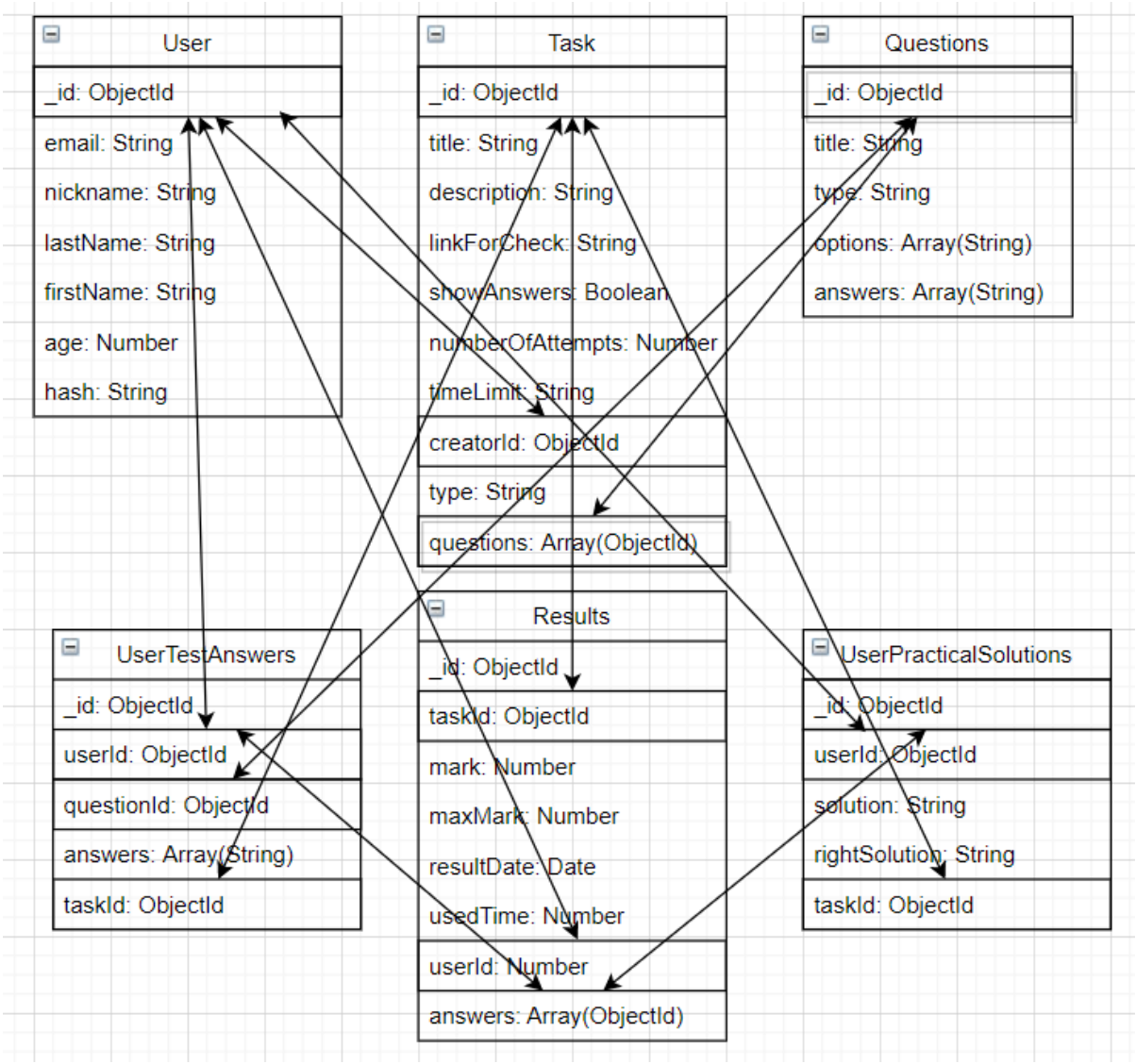

Рис.А.1. Модель колекцій документів бази даних.

ДОДАТОК Б «ЗНІМКИ ЕКРАНУ WEB-ДОДАТКУ»

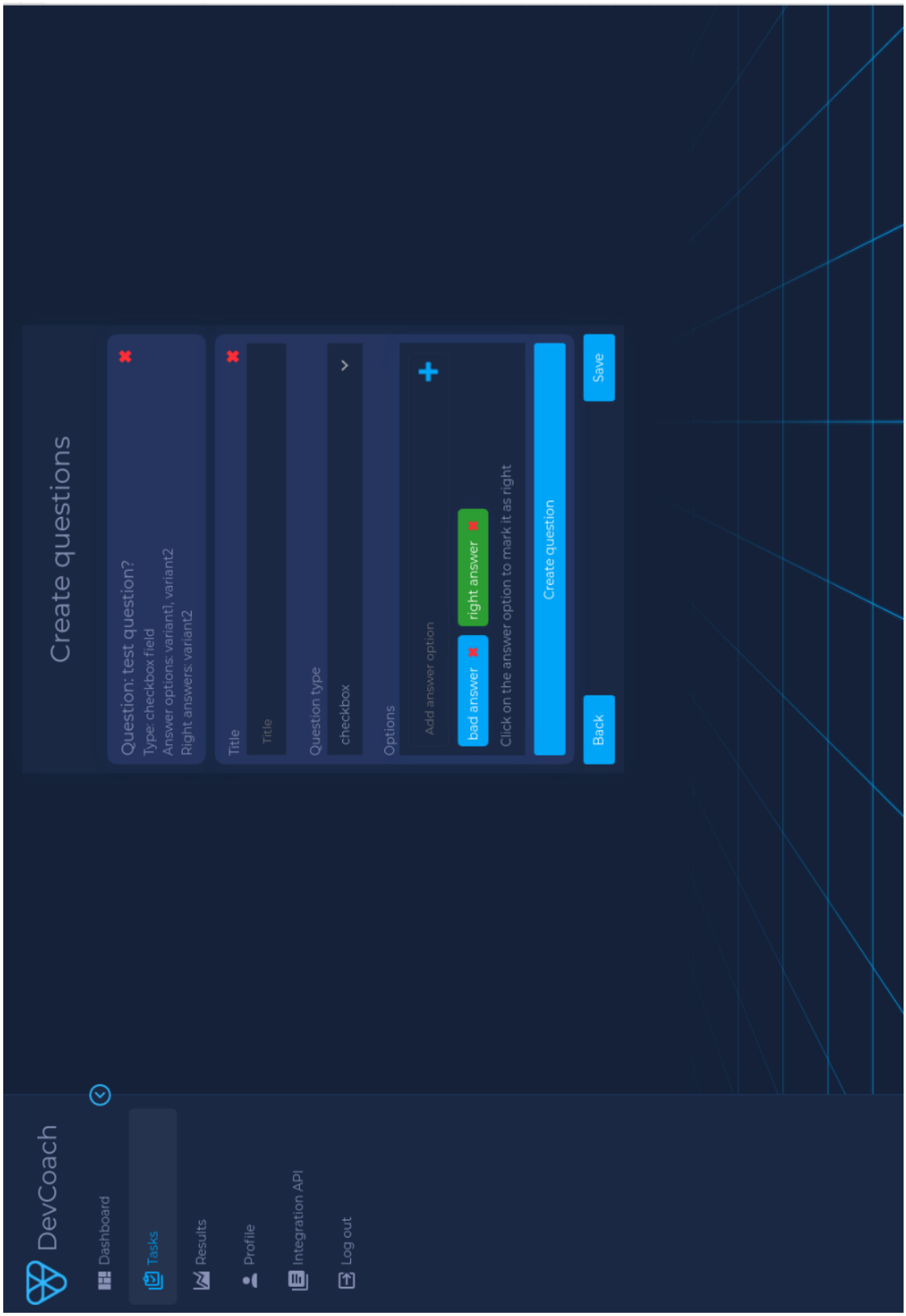

Рис.Б.1. Форма створення запитань для тестового завдання

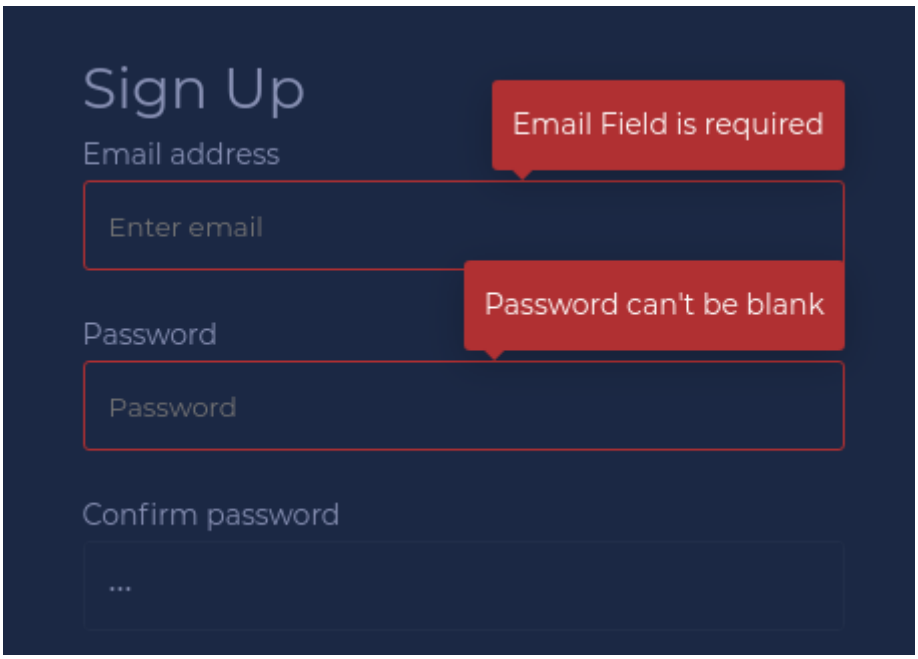

Рис. Б.2. Форма реєстрації з відображеними помилками валідації даних

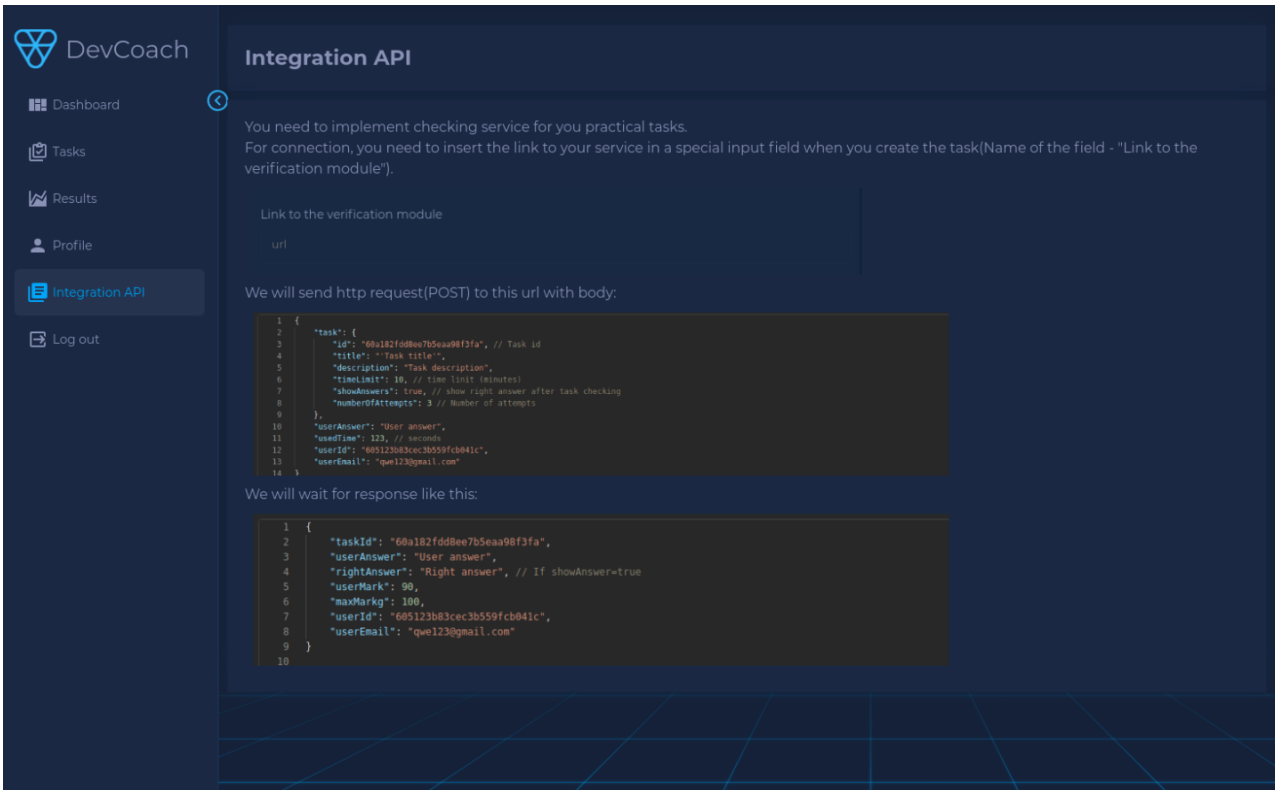

Рис. Б.3. Сторінка Integration API

ДОДАТОК В «ФРАГМЕНТИ КОДУ ПРОГРАМИ»

1. Форма авторизації користувача:

```
export const LoginForm = () => {
  const dispatch = useDispatch();

  const onGoogleLogin = useCallback(() => {
    dispatch(authActions.googleLogin());
  }, [dispatch]);

  const onLocalLogin = useCallback(
    (e) => {
      e.preventDefault();

      dispatch(authActions.localLogin());
    },
    [dispatch]
  );

  useEffect(
    () => () => {
      dispatch(xcriticalFormReset(LOGIN_FORM_NAME));
    },
    []
  );

  return (
    <>
      <SignInTitle>Sign In</SignInTitle>
      <LabelWrapper>
        Enter your email address and password to access admin panel.
      </LabelWrapper>

      <Form name={LOGIN_FORM_NAME} onSubmit={onLocalLogin}>
        <FormFieldWrapper>
          <Form.Field
            name={LoginFields.email}
            label="Email address"
            placeholder="Enter email"
            component={InputField}
            shouldFitContainer
          />
        </FormFieldWrapper>

        <FormFieldWrapper>
          <Form.Field
```

```
            name={LoginFields.password}
            label="Password"
            placeholder="Password"
            type="password"
            component={InputField}
            shouldFitContainer
          />
        </FormFieldWrapper>

        <SubmitWrapper>
          <Button onClick={onGoogleLogin}>Sign in with Google</Button>
          <Button type="submit">Submit</Button>
        </SubmitWrapper>
      </Form>

      <LoginFooter>
        <FooterLinkWrapper>
          <LabelWrapper>Don't have an account?</LabelWrapper>
          <SignUpLink>
            <Link to={PathNames.registration}>Sign up</Link>
          </SignUpLink>
        </FooterLinkWrapper>
      </LoginFooter>
    </>
  );
};
```

2. Функція створення теми для всіх універсальних компонентів в декількох видах:

```
export const getTheme = () => ({
  [buttonThemeNamespace]: {
    appearance: {
      default: defaultButton,
      secondary: secondaryButton,
    },
  },
  [inputThemeNamespace]: {
    appearance: {
      default: defaultInput,
      active: activeInput,
    },
  },
  [selectThemeNamespace]: {
    appearance: {
      default: defaultSelect,
```

```
    },
  },
  [popoverThemeNamespace]: {
    appearance: {
      default: defaultPopup,
      error: errorPopup,
    },
  },
  [modalThemeNamespace]: {
    appearance: {
      default: defaultModal,
    },
  },
  [checkboxThemeNamespace]: {
    appearance: {
      default: defaultCheckbox,
    },
  },
});
```

3. Функція створення теми для компоненту кнопки:

```
export const createButtonTheme = ({
  background,
  borderColor,
  color,
  hoverColor,
  borderHoverColor,
  hoverBackground,
}) => ({
  background,
  borderColor,
  height: 35,
  fontWeight: 500,
  color,
  span: {
    display: 'flex',
    alignItems: 'center',
  },
  svg: {
    marginRight: 5,
  },
  hover: {
    background: hoverBackground,
    color: hoverColor,
```

```
    borderColor: borderHoverColor,
  },
  active: {
    background: hoverBackground,
    color: hoverColor,
    borderColor: borderHoverColor,
  },
  focus: {
    background: hoverBackground,
    color: hoverColor,
    borderColor: borderHoverColor,
  },
  selected: {
    background: hoverBackground,
    color: hoverColor,
    borderColor: borderHoverColor,
  },
});
```

4. Налаштування webpack:

```
module.exports = {
  entry: './src/index.jsx',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'index.js',
    // chunkFilename: '[id].js',
    publicPath: '/',
  },
  resolve: {
    extensions: ['.js', '.jsx'],
  },
  module: {
    rules: [
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        use: ['babel-loader', 'eslint-loader'],
      },
      {
        test: /\.css$/,
        use: ['style-loader', 'css-loader'],
      },
      {
        test: /\.(png|jpe?g|gif|svg)$/,
        loader: 'url-loader?limit=10000&name=img/[name].[ext]',
      },
```

```
    {
      test: /\.(eot|otf|ttf|woff|woff2)$/,
      loader: require.resolve('file-loader'),
      options: {
        name: 'static/media/[name].[hash:8].[ext]',
      },
    },
  ],
},
optimization: {
  runtimeChunk: 'single',
  splitChunks: {
    chunks: 'all',
    maxInitialRequests: Infinity,
    minSize: 0,
    cacheGroups: {
      vendor: {
        test: /[\\/]node_modules[\\/]/,
        name(module) {
          const packageName = module.context.match(
            /[\\/]node_modules[\\/](.*?)([\\/]|$)/
          )[1];
          return npm.${packageName.replace('@', '')};
        },
      },
    },
  },
},
plugins: [
  new HtmlWebpackPlugin({
    template: ${__dirname}/public/index.html,
    filename: 'index.html',
    inject: 'body',
    favicon: './public/favicon.ico',
  }),
  new CopyWebpackPlugin({
    patterns: [{ from: './public/favicon.ico' }],
  }),
],
devServer: {
  host: 'localhost',
  disableHostCheck: true,
  hot: true,
  historyApiFallback: true,
  port: 8090,
  openPage: 'login',
  proxy: {
```

```
      '/api/**': {
        target: env.parsed.SERVER,
        secure: false,
        changeOrigin: true,
      },
    },
  },
};
```

5. Налаштування eslint:

```
{
  "extends": [
    "airbnb",
    "plugin:jsx-a11y/recommended",
    "prettier",
    "prettier/react"
  ],
  "parser": "babel-eslint",
  "globals": {
    "window": true,
    "describe": true,
    "it": true,
    "expect": true,
    "document": true,
    "localStorage": true
  },
  "plugins": ["jsx-a11y", "prettier"],
  "rules": {
    "import/prefer-default-export": "off",
    "no-console": ["error", { "allow": ["error", "log"] }],
    "semi": 0,
    "no-use-before-define": ["error", { "functions": false }],
    "no-param-reassign": [
      "error",
      { "props": true, "ignorePropertyModificationsFor": ["state"] }
    ],
    "react/jsx-filename-extension": [1, { "extensions": [".js", ".jsx"] }],
    "no-underscore-dangle": [
      "error",
      {
        "allow": [
          "REDUX_DEVTOOLS_EXTENSION",
          "REDUX_DEVTOOLS_EXTENSION_COMPOSE"
        ]
      }
    ],
```

```
    "react/jsx-props-no-spreading": "off",
    "react/prop-types": "off",
    "react/no-unescaped-entities": "off",
    "jsx-quotes": ["error", "prefer-double"],
    "quotes": [2, "single"],
    "no-multiple-empty-lines": "error",
    "import/order": [
      "error",
      {
        "newlines-between": "always-and-inside-groups",
        "groups": [
          "builtin",
          "external",
          "internal",
          "parent",
          "sibling",
          "index"
        ],
        "pathGroupsExcludedImportTypes": ["builtin"],
        "pathGroups": [
          {
            "pattern": "@xcritical/**",
            "group": "external",
            "position": "after"
          }
        ]
      }
    ],
    "sort-imports": [
      "error",
      {
        "ignoreCase": false,
        "ignoreDeclarationSort": true,
        "ignoreMemberSort": false,
        "memberSyntaxSortOrder": ["none", "all", "multiple", "single"],
        "allowSeparatedGroups": false
      }
    ],
    "prettier/prettier": [
      "error",
      {
        "semi": true,
        "singleQuote": true
      }
    ]
  }
}
```

6. Логіка малювання сітки що рухається та створює ефект глибини:

```
createAnimation = () => {
  const { createNegativeSpeedAnimation, createPositiveSpeedAnimation } = this;
  const { horizontalSpeed } = this.props;

  if (
    !horizontalSpeed ||
    !this.horizontalLineRefs.length ||
    !this.isVisible
  ) {
    return;
  }

  if (!this.horizontalLineRefs[0].current) {
    return;
  }
  this.animation && this.animation.stop();
  this.animation =
    horizontalSpeed > 0
      ? createPositiveSpeedAnimation()
      : horizontalSpeed < 0
      ? createNegativeSpeedAnimation()
      : null;
  this.animation && this.animation.start();
};

createNegativeSpeedAnimation = () => {
  const { layerRef, negativeAnimationLines } = this;

  return new Konva.Animation(negativeAnimationLines, layerRef.current);
};

createPositiveSpeedAnimation = () => {
  const { layerRef, positiveAnimationLines } = this;

  return new Konva.Animation(positiveAnimationLines, layerRef.current);
};
negativeAnimationLines = (frame) => {
  const { dy, stepY, height, strokeWidth } = this.props;

  this.horizontalLineRefs.forEach((line) => {
    if (!line.current) {
      return;
    }

    const limitHeight = line.current.attrs.isBackgroundLine
```

```
      ? height + (stepY - dy) - (this.strokeBackgroundWidth - strokeWidth) / 2
      : height + (stepY - dy);

    if (line.current.attrs.points[1] >= limitHeight) {
      line.current.attrs.points = line.current.attrs.isBackgroundLine
        ? [...this.lastLinePointsBackground]
        : [...this.lastLinePoints];
    }

    this.setPointsOffset(line);
  });
};

positiveAnimationLines = (frame) => {
  const { dy, stepY, strokeWidth } = this.props;

  this.horizontalLineRefs.forEach((line) => {
    if (!line.current) {
      return;
    }

    const limitHeight = line.current.attrs.isBackgroundLine
      ? 0 - (stepY - dy) - (this.strokeBackgroundWidth - strokeWidth) / 2
      : 0 - (stepY - dy);

    if (line.current.attrs.points[1] <= limitHeight) {
      line.current.attrs.points = line.current.attrs.isBackgroundLine
        ? [...this.firstLinePointsBackground]
        : [...this.firstLinePoints];
    }

    this.setPointsOffset(line);
  });
};

setPointsOffset = (line) => {
  const { horizontalSpeed } = this.props;

  const { points } = line.current.attrs;
  const y = points[1] - horizontalSpeed;
  const y2 = points[7] - horizontalSpeed;
  line.current.attrs.points = [
    points[0],
    y,
    points[2],
    y,
    points[4],
```

```
    y2,
    points[6],
    y2,
  ];
};

toggleAnimation = () => {
  this.isVisible = document.visibilityState === 'visible';

  if (this.isVisible) {
    this.animation && this.animation.start();
  } else {
    this.animation && this.animation.stop();
  }
};
```