ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «Розробка програмного забезпечення для тестування пацієнтів у психолога з використанням технологій Node.js та React»

Виконав: студент 4 курсу, групи ПД–41

Спеціальності

121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____Луппа О.А.___ _____
(прізвище та ініціали)

Керівник Гаманюк І.М._____
(прізвище та ініціали)

Рецензент_____
(прізвище та ініціали)

Нормоконтроль _____
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення_____

Ступінь вищої освіти - «Бакалавр»

Напрям підготовки -121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного
забезпечення
О.В. Негоденко
«____» _____2022 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Луппа Олексій Андрійович
(прізвище, ім‘я, по батькові)

1.Тема роботи: «Розробка програмного забезпечення для тестування пацієнтів у психолога з використанням технологій Node.js та React»

Керівник роботи Гаманюк І.М. викл. ,

(прізвище, ім‘я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «18» лютого 2022 року №__.

2. Строк подання студентом роботи__03.06.2022_____
3. Вхідні дані до роботи:

Науково-технічна література з питань, пов‘язаних з психологічними тестами

Науково-технічна література з розробки системи тестування_____

Перелік аналогів_____

Науково-технічна література з розробки програмного забезпечення_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1. Аналіз предметного середовища. Існуючі застосунки на ринку_____

4.2. Дослідження методів тестування

4.3 Проектування програмного забезпечення

4.4 Розробка програмного забезпечення та тестування

5. Перелік графічного матеріалу

5.1. Презентація в Power Point Слайд «Мета, Об’єкт та предмет дослідження»

5.2. Презентація в Power Point Слайд «Аналоги»

5.3. Презентація в Power Point Слайд «Технічне завдання»

5.4. Презентація в Power Point Слайд «Програмні засоби реалізації»

5.5. Презентація в Power Point Слайд «Методи та класи програми»

5.6. Презентація в Power Point Слайд «Проектування та реалізація БД»

5.7. Презентація в Power Point Слайд «Апробація результатів дослідження»

5.8. Презентація в Power Point Слайд «Висновки»

5.9. Скріншоти робочої програми

5.10. Скріншоти коду

6. Дата видачі завдання 11.04.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04-14.04	
2	Ознайомлення з психологічними тестами	15.04-17.04	
3	Написання першої глави	18.04-21.04	
4	Написання другої глави	22.04-25.05	
5	Написання третьої глави	26.05-05.05	
6	Розробка власного продукту	05.05-07.05	
8	Вступ, висновки, реферат	07.05-10.05	
9	Розробка демонстраційних матеріалів	11.05-15.05	
10	Попередній захист роботи	16.05-01.06	
11	Подання роботи в деканат	03.06	

Студент _____ Луппа О.А
(підпис) (прізвище та ініціали)

Керівник роботи _____ Гаманюк І.М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Тема бакалаврської роботи: Розробка програмного забезпечення з використанням технологій Node.js та React для тестування пацієнтів у психолога.

ВЕБ-ТЕХНОЛОГІЯ, ТЕСТУВАННЯ ПАЦІЄНТІВ У ПСИХОЛОГА, ВЕБ-ДОДАТОК, АНАЛІЗ ДАНИХ.

Об'єктом є процес автоматизації роботи психолога. Предметом дослідження є розробка програмного забезпечення для тестування пацієнтів у психолога. Ціль роботи – запроектувати та розробити додаток для автоматизації тестування

У роботі актуалізовано проблему ефективності використання сучасних WEB-технологій при реалізації WEB-інтерфейсу користувача, а також проблему з тестуванням пацієнтів у психолога. Запропоноване рішення дозволяє успішно інтегрувати стек сучасних WEB-технологій та застосувати його при розробці програмного комплексу для можливого тестування пацієнтів. Розроблений WEB-додаток показує в режимі реального часу можливість пройти тестування у психолога.

Метою дослідження є визначення способу створення інформаційної системи «тестування психолога», що містить інтерфейс користувача WEB-додатка, який оновлюється в режимі реального часу.

Проведено аналіз проблем сучасного WEB-розробки та аналіз шляхів їх вирішення, аналіз ринку існуючих автоматизованих систем управління, визначено їх переваги та недоліки. Пропонується вивчити сучасні WEB-технології та на їх основі розробити систему, яка спрощує процес взаємодії клієнтів та психологів.

Для розробки використовували VS Code, MySQL Workbench (графічний інтерфейс для MySQL), різні бібліотеки JavaScript.

Інтерфейс користувача реалізований за технологіями React.JS і D3. Для серверної частини було використано технологію Node.JS.

Були змодельовані діаграми для повноцінного розуміння усіх аспектів додатку. В рамках випускної кваліфікаційної роботи розроблено програмний комплекс, що включає WEB-інтерфейс користувача. Додаток був протестований за допомогою методики manual testing.

Додаток був визначений актуальним для використання психологами у особистих цілях, для автоматизації проходження психологічних тестів у навчальних закладах та для автоматизації збору та аналізу результатів.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПК – персональний комп'ютер

MVC – Model, View, Controller; програмна архітектура, структура системи, яка відокремлює бізнес-логіку від решти програми, логічно поділяючи додаток на 3 частини: Модель, Подання та Контролер, відповідно до скорочення

JS – JavaScript

DOM – Document Object Model, об'єктна модель документа

CSS – Cascading Style Sheets, каскадні таблиці стилів

Роутінг – функціонал, який зіставляє набраний користувачем URL-адресу з дією програмної логіки WEB-інтерфейсу

API – Application Programming Interface, програмний інтерфейс програми

JSON – JavaScript Object Notation, текстовий формат даних, використовується для представлення об'єктів у JavaScript

БД – база даних

СУБД - Система управління базами даних

ВСТУП

Актуальність теми дослідження. Жителі великих мегаполісів постійно стикаються з життєвими проблемами які потрібно вирішувати з психологами але бояться до них прийти. Навіть якщо є багатовіковий досвід, пацієнту може бути важко знайти необхідну йому інформацію та поділитися своєю проблемою з іншими.

Нинішнє покоління психологів має достатній рівень в користуванні веб-технологіями, які дозволяють орієнтуватися в просторі, але тестування все ще залишаються сліпою зоною.

Використання мобільних пристроїв є кращим варіантом надання інформації користувачам, однак не всі пацієнти є активними користувачами смартфонів. Для таких клієнтів необхідно надати інформацію в іншій доступній формі.

Усі ці фактори свідчать про те, що використання невеликих візуальних WEB-додатків у поєднанні з технологіями є перспективним напрямком у створенні різноманітних компонентів у роботі психолога.

Аналіз актуальності визначив вибір теми дослідження: «Розробка програмного забезпечення з використанням технологій Node.js та React для можливого тестування пацієнтів у психолога».

Гіпотеза дослідження: Використання сучасних технологій WEB-програмування та розроблених підходів до їх інтеграції в систему «тестування у психолога» дозволить значно пришвидшити процес розробки WEB-додатку «тестування у психолога», підвищить зручність використання послуг психолога та зменшить вартість обчислювальних ресурсів.

Метою дослідження є пошук способу створення інформаційної системи «тестування у психолога», що має можливість тестувати пацієнтів, підказки для клієнтів під час тестування та інтерфейс користувача WEB-додатка.

Для досягнення цієї мети необхідно вирішити наступні завдання:

- аналіз існуючих рішень у сфері інформаційних систем для організації «тестування у психолога»;
- складання схеми інформаційної системи та обґрунтування її ефективної застосовності;
- вибір засобів розробки з урахуванням існуючих критеріїв;
- розробка, тестування WEB-додатку для тестування пацієнтів у психолога;
- оцінити шляхи подальшої оптимізації споживаних ресурсів та способи подальшого масштабування системи.

Об’єктом дослідження є процес автоматизації роботи психологів. Предметом дослідження є розробка програмного забезпечення для тестування пацієнтів у психолога.

Методи дослідження включають:

- аналіз, порівняння, систематизація та узагальнення даних про існуючі та розроблені методики автоматизації роботи психологів;
- апробація сучасних WEB-технологій при створенні веб-додатку;
- тестування інтерфейсу на ПК;
- аналіз технологій та підходів, що дозволяють масштабувати та оптимізувати WEB-додаток;

Теоретичною основою дослідження було:

- зарубіжні дослідження та рішення щодо організації автоматизації роботи психологів;
- сучасні концепції та технології розробки веб-додатків;
- документація для різних фреймворків і бібліотек, що використовуються в сучасному WEB-програмуванні.

Наукова новизна та теоретичне значення дослідження.

Робота відкриває напрямок досліджень у розвитку сучасних інформаційних та WEB-технологій, використання інформаційних та WEB-технологій для можливого тестування пацієнтів у психолога.

Виявлено, обґрунтовано та описано переваги інформаційних технологій як інструменту розвитку тестувальних систем. Показано, що за допомогою цього інструменту стає можливим підвищити ефективність та зручність для пацієнтів, не витрачаючи на це значних ресурсів.

Дослідження показує, що використання інформаційної системи пацієнтами не порушує її цілісності, та одночасно розкриваючи її потенціал у вирішенні проблем, переповненістю та безпекою; а також у підвищенні задоволеності користувачів.

Практичне значення дослідження. Проведено аналіз традиційної системи тестування пацієнтів, а також кількох подібних аналогів «тестування у психолога», створено схему пропонованого тестування, робочий макет психолога та прототип з робочим програмним забезпеченням.

1 АНАЛІЗ ПРЕДМЕТНОГО СЕРЕДОВИЩА. ІСНУЮЧІ ЗАСТОСУНКИ НА РИНКУ

1.1 Опис предметного середовища

Основна мета психологічного тестування – дізнатися щось про людину, яку тестують. Існує кілька методів психологічного тестування [1]:

- тестовий опитувальник ґрунтується на системі запитань, попередньо відібраних і перевірених з точки зору їх обґрунтованості та достовірності, за відповідями на які можна судити про психологічні якості обстежуваних;
- тестове завдання передбачає оцінку психологічного стану та поведінки людини не за тим, що вона говорить, а за тим, що вона робить. У тестах такого типу людині дається ряд спеціальних завдань, за результатами яких судять про якість, що вивчається;
- проективний тест здійснюється шляхом проектування учасником дій і ситуацій на іншу особу і зазвичай має на меті вивчення тих психологічних і поведінкових особливостей людини, які вони погано усвідомлюють або викликають вкрай негативне ставлення з їх боку.

Поширеним способом тестування є комплексне проведення тестів-анкетування та тестових завдань. Взаємодіючи безпосередньо з психологом, група людей відповідає на стандартні запитання, а також виконує загальні завдання. Ці дії дозволяють фахівцеві найбільш точно оцінити ситуацію в колективі, оскільки завдання спільної роботи показують клімат у групі, а опитування спрямовані на виявлення особистісних якостей кожного учасника. Досить поширеним є також проведення лише анкетних тестів, за результатами яких психолог може оцінити ситуацію і дати рекомендації не тільки щодо покращення стосунків між учасниками команди, а й щодо усунення особистих проблем учня.

На даний момент у сфері розробки веб-додатків існує багато проблем, пов'язаних з недостатньою продуктивністю.

Дані у великих веб-додатках на стороні сервера (або бази даних) можуть займати більше 900 ГБ (наприклад, дані про студентів у міжнародному масштабі) [21]. Тому однією з проблем розробки WEB-інтерфейсу великих WEB-додатків є низька швидкість обробки великого потоку даних, що надсилається від сервера до WEB-клієнта (найчастіше, WEB-браузер). Організацію оновлення візуальних даних у WEB-клієнті можна здійснювати різними способами: оновлювати (і перемалювати) дані лише після прямого запиту користувача, використовувати опитування (WEB-клієнт робить запити до сервера через певний інтервал) або використовувати протокол, відмінний від HTTP, для обміну повідомленнями між браузером і WEB-сервером реального часу (наприклад, WebSocket).

Психологічний тест — це стандартизоване завдання, за результатами якого судять про психофізіологічні та особистісні характеристики, знання, уміння та навички досліджуваного.

Для створення ефективного психологічного тесту необхідно більше 10 років роботи авторських колективів. Якість тесту забезпечується багатоетапною процедурою перевірки та стандартизації ваг. Тестів, адаптованих до російських реалій 1990-х років, небагато, тому вибрати хороші психологічні тести для оцінки персоналу важко. Звичайно, крім тестів, відомі різні методики вивчення особистості, і кожен вирішує свої завдання.

Хоча психологічні знання широко доступні, психологічна наука вимагає поступового засвоєння, як і інші науки. Ілюзію простоти застосування тестів та інтерпретації результатів створюють такі книги, як Енциклопедія популярних психологічних тестів, Практична психологія на кожен день.

Характеристика психологічних тестів

Стандартизація - методологія тестування проходить стандартизацію, в результаті якої отримані дані повинні відповідати закону нормального розподілу

або нормі соціокультурного характеру. Відповідно до норм формуються діапазони значень, що вказують на силу вираженості досліджуваної ознаки.

Надійність — це властивість тесту давати близькі результати при багаторазовому вимірі. Надійна методика дає подібні результати незалежно від пори року чи статі експериментатора, вплив таких фонових факторів слід мінімізувати самим методом, що визначає його надійність.

Надійність - це відповідність результатів випробувань характеристики, для якої вони призначені для вимірювання. Розрізняють внутрішню і зовнішню валідність. У випадку екстерна це відповідність можна перевірити за позитивною кореляцією, з об'єктивними досягненнями, результатами тесту на інтелект, порівнянними з успішністю. У випадку з внутрішнім все складніше, тут мова йде про теоретичний зв'язок, про те, наскільки побудована модель дійсно моделює заявлений аспект. Але все може бути простіше, якщо вже є такі «перевірені» методи, то можна обійтися співвіднесенням з відомим методом. Якщо вам вдалося бути новатором, то внутрішня обгрунтованість знайшлася завдяки багаторічній експериментальній та інтелектуальній роботі і певним чином залишається на совісті «творця».

Надійність є одним із критеріїв якості тесту, його стабільності щодо похибок вимірювань. Існує два види надійності - надійність як стабільність і надійність як внутрішня узгодженість.

Надійність як стабільність

Стабільність результатів тесту або retest reliability (англ. – test-retest reliability) – можливість отримання однакових результатів від досліджуваних у різних випадках.

1.2 Ознайомлення із середовищем для розробки програмного додатку

Технологічний стек, який використовується при розробці WEB-додатка, включає фреймворки та бібліотеки, необхідні для ефективної та швидкої розробки, що дозволяє з мінімальною кількістю даних та обмеженими

системними вимогами інтегрувати WEB-додаток з іншими службами Smart Parking для відображення даних у WEB інтерфейс в режимі реального часу.

Бібліотеки JavaScript

Розглянемо основні бібліотеки та фреймворки JavaScript, які використовуються в сучасному WEB-програмуванні. У роботі [1] порівнюється їх продуктивність (згідно з рисунком 1) у браузері Google Chrome 48. У цьому випадку ми тестуємо створення 1000 рядків відразу після завантаження сторінки («створити 1000 рядків»), оновлюючи 1000 рядків у таблиці після 5 ітерацій «розігрів» движка JavaScript («оновити 1000 рядків (гаряче)») часткове оновлення рядків у таблиці після 5 ітерацій «розігріву» движка JavaScript (додавання крапки в кінець кожного 10-го рядка, «часткове оновлення»), виділення рядка після 5 ітерацій «прогрівання» движка JavaScript (візуальне виділення рядка, «вибрати рядок»), видалення рядка після 5 ітерацій «розігріву» движка JavaScript («видалення рядка»). Як видно з гістограми на рисунок 1, бібліотека VueJS працює найкраще, за винятком оновлення рядків у таблиці. WEB-інтерфейс «тестування у психолога» передбачає візуальне оновлення даних в режимі реального часу, а створення візуальної «підкладки» для відображення тестових запитань створюється лише один раз, тому потрібно вибрати бібліотеку, продуктивну при оновленні даних і йде добре з реалізацією маршрутизації клієнтів. Такою бібліотекою є ReactJS.

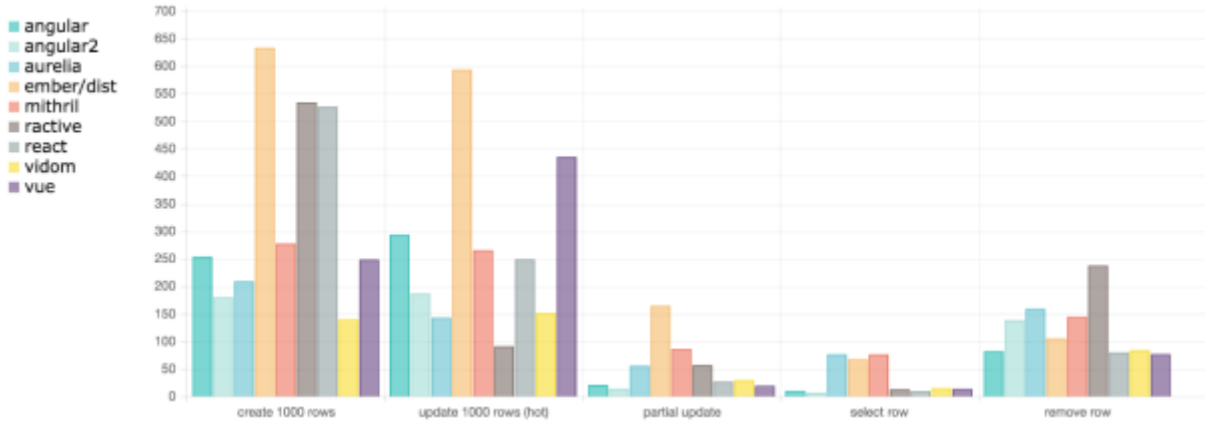


Рисунок 1.1 -Гістограма продуктивності фреймворку JS (мс)

React — це бібліотека JavaScript для створення інтерфейсів користувача. React дозволяє описувати елементи, тобто є декларативним. Використовуючи цю технологію, можна безболісно створювати інтерактивні інтерфейси користувача.

React побудований на концепції компонентів. Він відрізняється від таких фреймворків, як Angular або Ember, які використовують двостороннє прив'язування даних для оновлення HTML сторінки. На думку багатьох розробників front-end [3], React легше вивчити, ніж Angular або Ember — він набагато менший і добре працює з jQuery та іншими фреймворками. Він також надзвичайно швидкий, оскільки використовує віртуальний DOM і оновлює лише змінені частини сторінки (доступ до DOM все ще є повільною частиною сучасних WEB-додатків, тому бібліотека отримує перевагу в продуктивності, оптимізуючі її).

Маршрутизація на стороні клієнта в цьому випадку може бути реалізована за допомогою бібліотеки react-router [4]. Ця бібліотека дозволяє асоціювати клієнтські маршрути з компонентами React, тому всі можливі стани WEB-інтерфейсу будуть міститися у файлі JavaScript проекту, де оголошуються маршрути (у точці входу програми, згідно з рисунком 1.2).

```
const routes = (
    <Router history={browserHistory}>
        <Redirect from="/" to="/login" />
        <Route path="/" component={MainComponent}>
            <Route path="/login" component={LoginComponent} />
            <Route path="/registration" component={RegistrationComponent} />
            <Route path="/reservation" component={ReservationComponent} onEnter={requireAuth} />
            <Route path="/parking" onEnter={requireAuth}>
                <Route path="auto" component={ParkingComponent} onEnter={autoReserve} />
                <Route path="manual" component={ParkingComponent} />
            </Route>
        </Route>
    </Router>
);

ReactDOM.render(routes, document.getElementById('root'));
```

Рисунок 1.2 - Код JS-файлу точки входу програми, що містить маршрутизацію клієнта

Для малювання діаграм або складних креслень у WEB-інтерфейсі може знадобитися бібліотека D3. D3.js (або просто D3) — це бібліотека JavaScript для маніпуляції та візуалізації даних. Він надає зручні утиліти для обробки та завантаження масивів даних та створення елементів DOM. Назва d3 означає документ, керований даними.

Це бібліотека JavaScript, орієнтована на роботу з даними та їх візуальне представлення для WEB-додатків, включаючи завантаження даних, візуалізацію в реальному часі та багато інших функцій [5, 6].

Бібліотеки CSS

CSS є примітивною і неповною мовою програмування. Дуже важко створити функцію, повторно використати визначення або використати в ній успадкування. Для великих проектів або складних систем підтримання коду CSS стає великою проблемою. Однак WEB-технології швидко розвиваються, впроваджуються нові специфікації як в HTML, так і в CSS. Веб-браузери використовують ці специфікації, але залишають свої префікси, специфічні для виробника. У деяких випадках (наприклад, фонові градієнти) програмування за допомогою спеціальних префіксів постачальників стає тягарем. Щоб досягти однакового результату, потрібно додати всілякі префікси різних версій браузера.

Щоб покращити код CSS, програмісти часто використовують різні підходи, наприклад, розбивають визначення стилів на невеликі файли та імпортують їх в один великий головний файл.

Такий підхід допоміг розділити стилі на компоненти, але не вирішив проблеми повторення коду і не полегшив його обслуговування. Інший підхід полягав у спробі ввести об'єктно-орієнтоване програмування в CSS. У цьому випадку до елемента були застосовані два або більше визначення класів. Кожен клас додав до цього елемента один тип стилю. Створення кількох класів збільшило можливість успадкування коду, але зменшило здатність ефективно підтримувати код.

Препроцесори, у свою чергу, допомагають вам писати масштабований і підтримуваний код CSS. Використовуючи препроцесор, програміст може легко підвищити свою продуктивність і зменшити кількість коду в своєму проекті.

Таким чином, перш за все, для комфортного програмування стилів для WEB клієнта розробнику потрібен препроцесор CSS. Препроцесори CSS розширюють використання CSS за допомогою плагінів, операторів, інтерполяцій, функцій, міксинів та інших корисних інструментів. Найвідомішими препроцесорами CSS є SASS [11], LESS [12] та Stylus [13].

Як і будь-яка мова програмування, препроцесори мають різний синтаксис, але вони дуже схожі один на одного. Усі препроцесори підтримують «класичне» програмування CSS, а їх синтаксис загалом схожий на CSS.

Давайте подивимося, як синтаксис препроцесора CSS відрізняється на прикладі міксинів (як показано на малюнку 3). Міксини – це набір умов, які формуються за деякими параметрами або статичними правилами. За допомогою них ви можете створювати щось на зразок функцій і повторно використовувати правила стилю в інших місцях вашого коду. Як ви можете побачити на малюнку 3, найбільш знайомий синтаксис, схожий на CSS, - це SASS і LESS, але SASS має більш конкретний, детальний опис використання міксинів (@mixin, @include), ніж дубльований LESS (символ "".). клас селектора, який починається з символу «.», тому розробка буде використовувати препроцесор SASS CSS [14].

Sass

```
@mixin bordered($width) {
    border: $width solid #ddd;

    &:hover {
        border-color: #999;
    }
}

h1 {
    @include bordered(5px);
}
```

Less

```
.bordered (@width) {
    border: @width solid #ddd;

    &:hover {
        border-color: #999;
    }
}

h1 {
    .bordered(5px);
}
```

Stylus

```
bordered(w)
    border: n solid #ddd
    &:hover
        border-color: #999

h1
    bordered(5px)
```

Css

```
h1 { border: 5px solid #ddd; }
h1:hover { border-color: #999; }
```

Рисунок 1.3 - Опис міксинів у різних препроцесорах та аналог в CSS

Далі розробнику потрібно написати крос-браузерний CSS-код більш ефективно. Для цього потрібна утиліта, яка автоматично додає префікси до правил стилів. Наприклад, Google рекомендує автопрефікс [15, 16] для обробки postCSS. Ця бібліотека дозволяє писати стилі без специфічних для виробника префіксів (згідно з рисунками 4 і 5), тим самим вирішуючи проблему написання кросбраузерних стилів.

```
:fullscreen a {
    display: flex
}
```

Рисунок 1.4 – Опис правила CSS без префіксів постачальника

```
:-webkit-full-screen a {
    display: -webkit-box;
    display: flex
}
:-moz-full-screen a {
    display: flex
}
:-ms-fullscreen a {
    display: -ms-flexbox;
    display: flex
}
:fullscreen a {
    display: -webkit-box;
    display: -ms-flexbox;
    display: flex
}
```

Рисунок 1.5 - Опис правил CSS з префіксами постачальника, після запуску autoprefixer

Інтеграція стека технологій

Для ефективної інтеграції великої кількості залежностей у проект потрібен менеджер пакетів. Без менеджера пакетів залежності потрібно завантажувати вручну з джерел Інтернету або залежності потрібно завантажувати в систему управління кодом, що збільшить розмір проекту розробки в десять разів і негативно вплине як на швидкість завантаження, встановлення проекту та ресурси сервера контролю джерел. . Найвідомішим менеджером пакетів для бібліотек JavaScript є npm, який, у свою чергу, є найбільшим у світі реєстром програмного забезпечення [17]. Від npm немає необхідності додавати файли залежностей до проекту, а просто додайте файл під назвою "package.json" (Додаток А) до кореня проекту, який містить зіставлення між іменами бібліотек JavaScript та їх версіями. , тоді як версії можна «заморозити». Таким чином, проект завжди матиме необхідні версії пакетів, і ви можете завантажити їх перед запуском або створенням проекту за допомогою простої команди в консолі встановлення npm (розташованої в корені проекту).

Залежності проекту, у свою чергу, можуть містити власні залежності, які будуть завантажені під час команди «npm install». Таким чином, кодова база проекту буде містити значну кількість коду, більшість з якого не стискається, а деякі частини коду можуть не використовуватися в проекті та займати додатковий простір, що спричиняє завантаження додаткових даних у браузер під час завантажити. сторінки та розміщення даних в оперативній пам'яті, що може уповільнити WEB-інтерфейс. Щоб уникнути цих проблем, вам слід видалити невикористаний код і стиснути його під час складання, але краще робити це автоматично. Це можна зробити автоматично, якщо ви правильно налаштувати збірку webpack [10], а саме, додавши UglifyJsPlugin [18] до файлу конфігурації webpack, який мінімізує та вимикає JavaScript. Вам також потрібно додати DedupePlugin [19], який шукає ідентичні файли в проекті та виключає їх з остаточної збірки. Невикористані файли та бібліотеки webpack не включаються у вихідні дані збірки за замовчуванням.

Щоб автоматично використовувати такі технології, як препроцесори CSS, обробка постCSS (автопрефікс), сучасні стандарти EcmaScript [7], необхідно також додати так звані «завантажувачі» до конфігураційного файлу системи збірки webpack. [20]

Наприклад, залежності babel-loader, sass-loader, postcss-loader і autoprefixer, описані у файлі "package.json" (Додаток А), дозволяють розробнику писати сучасний код JavaScript, використовувати синтаксис препроцесора SASS CSS, а не написання специфічних для постачальників префіксів, тому що переклад мов і додавання префіксів відбуватиметься автоматично при компіляції

1.3 Знайомство з програмними аналогами на ринку

Веб-сайт psytests.org

Веб-сайт psytests.org містить велику колекцію психологічних тестів [2]. Тестування безкоштовне і не вимагає реєстрації.

Після проходження тесту генерується унікальне посилання на сторінку з аналізом результатів тестування. Доступ до результатів тесту може отримати кожен, хто має посилання на цей результат.

Користувачі сайту відзначають наступні переваги даного тестового сервісу: безкоштовність, анонімність, зручний інтерфейс та високу якість тесту. Кожен тест на сайті супроводжується детальним описом та списком літератури, використаної при його цифровізації.

Сайт оформлений зовні дуже просто без дизайну, тепер він не міг залучити молоду аудиторію. На рисунку 1.6 показано скріншот форми для вибору відповіді на запитання під час проходження тесту на сайті psytests.org. З малюнка видно, що елементи керування для 21 варіанту відповіді великі, що покращує зручність проходження тесту. Крім того, на сторінці опитування немає зайвих елементів.

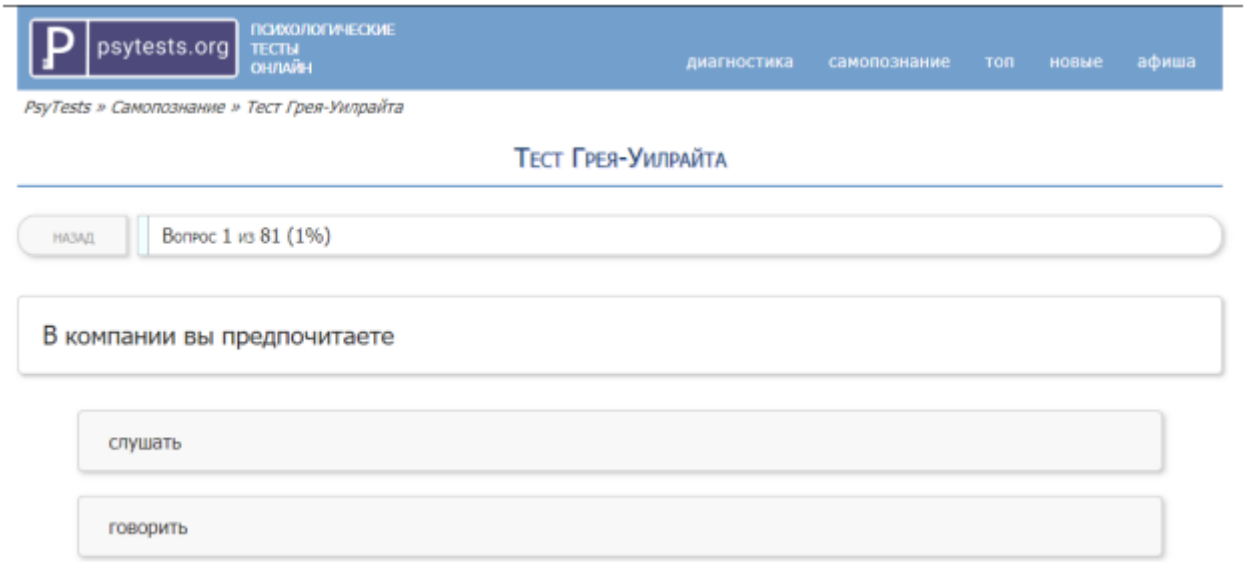


Рисунок 1.6 - Форма для вибору відповіді на питання при проходженні тесту на сайті psytests.org

Сайт psytests.org не є заміною розробляється рішення, оскільки він спрямований на проведення анонімного тестування і не містить інструментів для організації групового тестування.

Система тестування INDIGO

Система тестування INDIGO – це професійний інструмент для автоматизації процесу тестування та обробки результатів, призначений для вирішення широкого кола завдань[3]:

1) перевірка та контроль знань учнів;

2) визначення професійного рівня працівників;

3) проведення психологічного тестування;

4) проведення опитувань;

5) організація олімпіад та конкурсів.

Система INDIGO — це потужний і гнучкий інструмент тестування. Системою можна управляти як через локальну мережу, так і через Інтернет. Є вбудований редактор тестів, система правил проходження тестів, розроблені інструменти для підвищення безпеки даних.

На рисунку 1.7 показана форма вибору відповіді на запитання при проходженні тесту на платформі INDIGO. На рисунку видно, що елементи керування виконані в класичному стилі, внаслідок чого інтерфейс для відповідей на запитання на INDIGO менш зручний, ніж на сайті www.psytests.org.

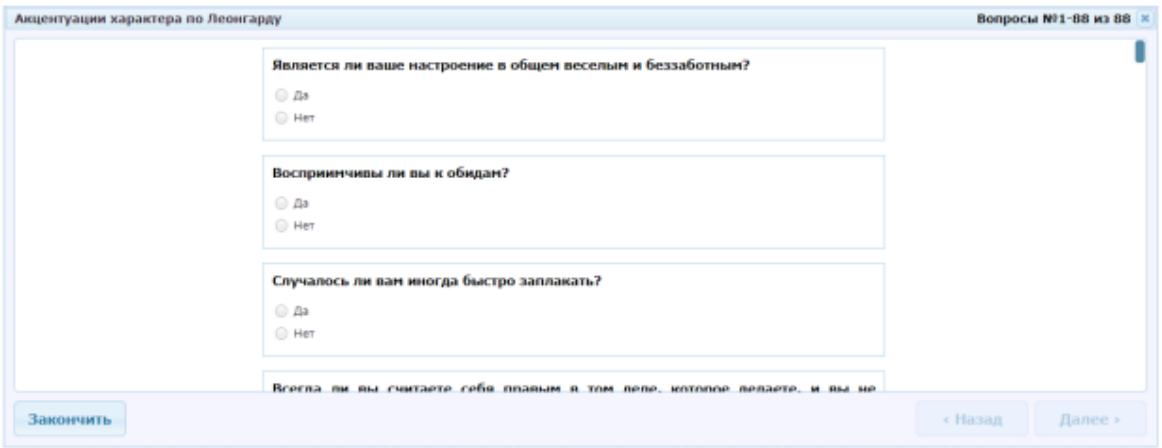


Рисунок 1.7 – Форма для вибору відповіді на запитання при проходженні тесту на платформі INDIGO

Основним недоліком тест-системи INDIGO є її висока вартість. Система тестування INDIGO ліцензована за принципом одна ліцензія на один тестовий сервер, ліцензія не обмежена в часі та оплата здійснюється лише один раз.

У той же час для використання цієї програми необхідно встановити певне програмне забезпечення, що не робить цю систему тестування мобільною, а також є великим недоліком.

Тестування системи StartExam

Система тестування співробітників StartExam є хмарним рішенням[4]. Доступ як до особистого кабінету адміністратора, так і до сторінки проходження тесту здійснюється через веб-інтерфейс. Метою сервісу є проведення зрізів знань серед співробітників.

Особистий кабінет адміністратора забезпечує доступ до управління списком учасників тестування, редагування та управління центрами тестування. Тестові центри — це група учасників тестування, яким необхідно скласти один і той же набір тестів.

Учасники тестування також мають особистий кабінет і для того, щоб пройти тест, вони повинні ввести свої облікові дані.

На рисунку 1.8 показана форма вибору відповіді на запитання при проходженні тесту на платформі StartExam. Оскільки StartExam не спеціалізується на психологічних тестах, картинка взята з тесту в іншій області.

Колірна палітра, яка використовується в інтерфейсі, має низьку контрастність, тому проходження тестів може бути складним для людей з вадами зору.

Основною відмінністю розробленого рішення від сервісу StartExam є його спеціалізація. Сервіс StartExam спеціалізується на тестах для перевірки знань, а рішення, яке розробляється, — на психологічному тестуванні. Крім усього іншого, ця система тестування є платною, якщо у вас є безкоштовні аналоги, це мінус послуги.

Рисунок 1.8 – Форма для вибору відповіді на запитання при проходженні тесту на платформі StartExam

1.4 Структура тестування пацієнтів у психолога

Постановка проблеми.

Для підвищення ефективності тестування логічним було б розширити коло перевіряються психічних процесів і, відповідно, діапазон вимірюваних психологічних характеристик. Звичайно, для цього потрібні принципово нові, особливо чутливі психологічні тести, адаптовані до конкретних видів пацієнтів. Оцінювати пацієнтів потрібно з точки зору споживчого мислення. При цьому необхідно враховувати цілий комплекс механізмів розуміння, асоціацій, прийняття рішень за умов вибору, наявність стереотипів, міфів, упереджень, соціальних уявлень, психологічних установок, орієнтації на соціальні оцінки референтні групи та багато іншого. Визначення того, що і за яких умов необхідно

перевірити, є важливою теоретичною проблемою, вирішення якої дає змогу психологу отримати значні переваги перед конкурентами.

Які методи зазвичай використовуються для тестування

Сьогодні рекламні та маркетингові видання часто критикують за тестування якісної реклами.

Більше за інших - метод фокус-груп. Однак не варто вимагати відповідей на складні питання від методу, який спочатку для цього не був призначений. Що робити, якщо нас все ще цікавлять складні психологічні питання? Звичайно, можна спробувати ускладнити метод фокус-груп, ввівши його в сферу потужних психологічних технологій. Наприклад, шляхом поєднання дискусій з проективними тестами, візуалізації рекламних матеріалів, використання інструментальних методів збору й обробки інформації та інших нетрадиційних форм організації групової дискусії. Однак чи буде така технологія дійсна? Чи буде це відповідати теорії та методології загальної та соціальної психології, які лежать в основі будь-якої ефективної групової взаємодії? Це питання залишається відкритим.

У свою чергу, цілком очевидно, що введення елементів класичного експерименту в групові дискусії збільшує кількість артефактів, знижує рівень контролю над змінними і в кінцевому підсумку створює лише ілюзію нових і глибших знань. Наприклад, учасники фокус-груп не зможуть відповісти на питання, що згадає пасажир (а може, водій) автомобіля, побачивши на трасі рекламний щит, що рухається зі швидкістю близько 120 км на годину. Це завдання важливе для тестування конкретної реклами, розміщеної на придорожньому білборді, але не вирішується методом фокус-груп, а також іншими методами дослідження ринку, широко використовуваними сьогодні.

Відразу слід сказати, що існує багато завдань для тестування пацієнтів, які, в принципі, неможливо вирішити іншими методами, крім експерименту. І експеримент не можна звести лише до пробних тестів (як це іноді трактують у літературі).

На сьогоднішній день є досвід впровадження проектної форми роботи в систему вищої освіти. Процес впровадження відбувався поетапно.

Модуль 1.

Знайомство з групою, отримання зворотного зв’язку (цілі, очікування від курсу), постановка питань, що цікавлять. Метою модуля є формування у студентів цілісного бачення та розуміння майбутнього процесу викладання дисципліни.

Модуль 2.

Психодіагностика. За допомогою спеціальних тестів і методик вивчаються особистісні особливості учнів (опитувальник «Ролі майстра групової роботи», вправа «Машина», методика розподілу ролей у колективі за М. Бельбіним та ін.). Процедура психологічної діагностики дозволяє сформувати уявлення про студента як про члена команди проекту. Для отримання повної оцінки ситуації студенти поділяються на групи. Завдання – розв’язувати випадки та вивчати відповідний матеріал. Мета – створити умови для індивідуальної та командної роботи. Тут вибудовується єдиний комунікативний простір всередині команд.

Модуль 3.

Презентація проектів. Представлені теми проектів, створені умови для самовизначення. Розподіл по командах здійснюється з урахуванням результатів психодіагностики та особистого вибору. Команди працюють у рамках власного проекту, який є частиною спільного проекту. У системі освіти це курс; в організації – це діяльність організації в цілому. Експерти-менеджери, які керують процесом, залучаються до роботи, дають зворотний зв'язок, діляться досвідом та допомагають у вирішенні виникаючих питань. Робота над проектом – це контакт з командою, рефлексія, моніторинг та аналіз процесу, прогнозування результатів, нових знань і компетенцій, практика, зворотний зв’язок у формі діалогу з експертами, новий досвід і можливості, а також цікавий, захоплюючий процес.

1.5 Висновки до глави 1

У цьому розділі розглядалися основні поняття та концепції побудови веб-сайтів.

Зі сказаного вище, можна зробити такі висновки: Розробка сайту йде поетапно.

Головні етапи:

- розробка структури сайту: визначення вихідних даних для сайту; визначення вимог до зовнішнього вигляду та функціональності; формування структури сайту – розділів меню;
- розробка концепції дизайну: створення дизайн-макету головної сторінки сайту; затвердження концепції дизайну – макету головної сторінки; створення внутрішніх сторінок сайту та визначення змін у дизайні до внутрішніх сторінок;
- html-верстка, дизайн та створення внутрішніх сторінок: розробка наповнення внутрішніх сторінок; розробка додаткових сторінок (карта сайту, результати пошуку тощо); оптимізація зображень;
- програмування: визначення задач програмування; розробка структури баз даних; написання адміністрування скриптів.

Вимоги до контенту сайту досить прості: контент сайту повинен бути написаний простою, зрозумілою мовою, орієнтованою на Вашу аудиторію сайту.

Також було розглянуто аналогічні сайти конкурентів.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Технічне завдання

Розробити інформаційну систему для можливого тестування пацієнтів у психолога у вигляді веб-сайту. Використання HTML, CSS, JS(Node.js та React) для створення веб-сайту, використання MySQL та PHP для роботи з базою даних, розробка форми та структури форми для проектування системи тестування для отримання замовлень на продукцію через web-представництво. Сайт має бути адаптивним і крос-браузерним. Необхідно передбачити перевірку правильності введених даних. Також необхідно реалізувати і надати всі види полів для реалізації форми, вказавши необхідні поля для заповнення форми. Додайте на сайт допоміжні сторінки, створивши таким чином структуру (наприклад, сторінку привітання, сторінку з формою реєстрації, сторінку, яка відкривається після успішного заповнення форми, сторінку контактів) і зв’яжіть ці сторінки одна з одною за допомогою посилань або кнопки.

Використання HTML (мова розмітки гіпертексту), основного будівельного блоку веб-програмування, і CSS (каскадні таблиці стилів) описують зовнішній вигляд сайту, створюючи форму, описану раніше. Як середовище розробки можна використовувати будь-який інструмент, Notepad++, VS Code або інші. Переконайтеся в правильності введених даних. Правильне управління базою даних.

2.2 Засоби розробки

Опис та обґрунтування вибору мови, середовища програмування та засобів розробки програмного забезпечення:

Вибір мови програмування

В якості мови програмування було обрано JavaScript.

Мова програмування JavaScript (JS) — це в першу чергу мова програмування, яка надає веб-сторінкам можливість реагувати на дії користувача та перетворювати статичні сторінки в динамічні. Таким чином, сторінки стають інтерактивними та зручнішими у використанні.

Що насправді може робити JavaScript? JavaScript, який існує сьогодні, є «безпечною» мовою програмування, призначеною для загальних цілей. Він не надає можливості для низькорівневих засобів роботи з пам'яттю, процесором, оскільки в першу чергу був орієнтований на браузери, в яких ця здатність не потрібна.

Що стосується можливостей мови JavaScript, то вони залежать від середовища, в якому, власне, і працює сам JavaScript. У браузері JavaScript може робити все, що пов’язано з будь-якими діями зі сторінкою, взаємодією з клієнтом сайту і, певною мірою, із сервером, наприклад:

- змінюйте стилі елементів, ховайте, показуйте елементи, видаляйте існуючі теги HTML, створюйте нові теги HTML тощо.

- реагувати на всілякі дії відвідувача, обробляти натискання користувачем сторінки на клавіатурі, клацання його мишею, переміщення курсору, помахи тощо.

- відправляти запити на сервер і завантажувати дані без перезавантаження сторінки (ця технологія називається «AJAX»);

- отримувати та встановлювати файли cookie, запитувати дані, відображати повідомлення.

Чого не може робити JavaScript? JavaScript: це швидка та потужна мова, але браузер накладає деякі обмеження на його виконання.

Обмеження встановлені для безпеки користувачів, щоб зловмисник не міг використовувати JavaScript для отримання особистих даних або якимось чином завдати шкоди комп’ютеру користувача.

Ці обмеження не існують, якщо JavaScript: використовується поза браузером, наприклад на сервері. Крім того, сучасні браузери надають власні

механізми встановлення плагінів і розширень, які мають розширені можливості, але вимагають від користувача спеціальних кроків встановлення.

Більшість можливостей JavaScript у браузері обмежені поточним вікном і сторінкою.

Враховуйте такі обмеження:

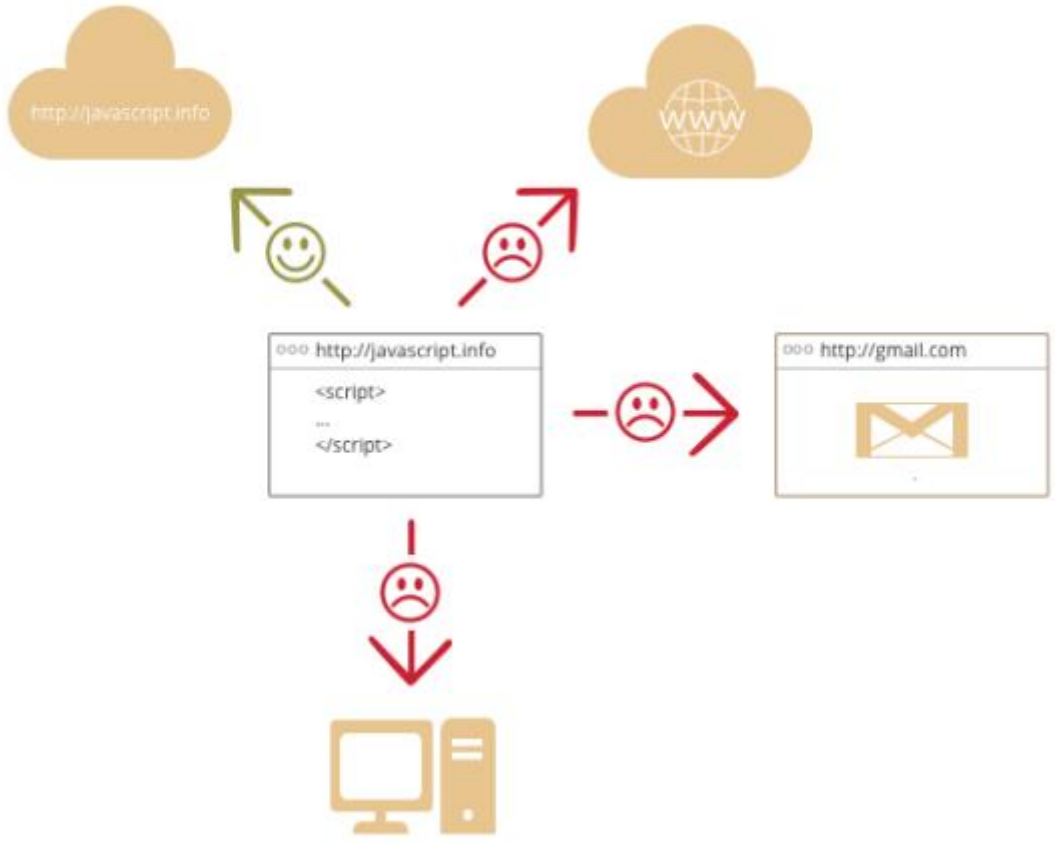


Рисунок 2.1 – Обмеження мови JavaScript (JS)

- JavaScript не може читати записувати довільні файли на жорсткий диск, копіювати їх або викликати програми. Він не має прямого доступу до операційної системи.

- сучасні браузери можуть працювати з файлами, але ця можливість обмежена спеціально виділеним каталогом — «пісочниця». Можливість доступу до пристроїв також розвивається в сучасних стандартах і частково доступна в деяких браузерах.

- JavaScript, запущений на одній вкладці, не може взаємодіяти з іншими вкладками та вікнами, за винятком випадків, коли він відкривав це вікно або декілька вкладок з одного джерела (одного домену, порту, протоколу).

- є способи обійти це, і вони розглянуті в підручнику, але вони вимагають спеціального коду для обох документів, які знаходяться на різних вкладках або вікнах. Без нього з міркувань безпеки неможливо перейти з однієї вкладки на іншу за допомогою JavaScript: це неможливо.

- з JavaScript ви можете легко надсилати запити на сервер, з якого прийшла сторінка. Запит на інший домен також можливий, але менш зручний, оскільки тут теж є обмеження безпеки.

Нижче наведено причини вибору мови програмування JavaScript.

Простота: в основному, бібліотеки в Node мають прості API, які легко зрозуміти та працюють інтуїтивно зрозумілим способом. Якщо одна з бібліотек з якихось причин не підходить для розробки, то їй можна знайти заміну, оскільки їх величезна кількість.

Контроль: програміст сам вибудовує інфраструктуру проекту, вибираючи та комбінуючи невеликі модулі для конкретних завдань. Це займає більше часу, але результат того вартий. Після розуміння отриманий досвід легко застосувати в майбутньому.

Універсальність: JavaScript спочатку працював лише на клієнті.

Спочатку разом з Node він перейшов на сервер, а останнім часом на ньому з'явилася можливість успішно писати настільні (Electron) і мобільні додатки. Крім того, для мобільних додатків є варіант гібридних програм (обгортка використовується над браузером (Cordova)) або додатків з нативним інтерфейсом (ReactNative, NativeScript). Для Node бібліотек величезна кількість і їх легко інтегрувати з іншими технологіями, базами даних, хмарними технологіями, різними форматами і протоколами, є все.

Легке розгортання: Node дуже легко розгорнути на сервері, як Linux, так і Windows. Після багатьох років роботи з NET для мене щоразу розгортання було

неприємним досвідом, на Node цей процес навіть приносить задоволення. Це просто потрібно спробувати.

Продуктивність: Node є асинхронним і не блокує виконання під час тривалих операцій, таких як читання файлу або доступ до бази даних. Це дозволяє досягти високого рівня продуктивності при використанні одного потоку (single thread environment). З іншого боку, обчислення в JavaScript повільніше, ніж у мовах зі статично типізованим. Для більшості проектів це не проблема. Якщо вам потрібні обчислення, а не просто перетворення даних, то краще написати окремий сервіс на чомусь іншому.

Одна мова на сервері і клієнті: це зручно, оскільки дозволяє без зусиль передавати код між клієнтом і сервером, його легше розробляти та підтримувати [2].

JavaScript дуже гнучкий і простий у використанні, багато недоліків мови було виправлено в останній версії ES6, а введення тексту можна додати за допомогою JavaScript.

Також, розглядаючи мову програмування JavaScript, необхідно знати, що таке HTML і CSS.

HTML, (мова гіпертекстової розмітки, "мова гіпертекстової розмітки") є стандартизованою мовою розмітки для документів у всесвітній мережі.

Більшість веб-сторінок описують розмітку в HTML (або XHTML).

При розробці цього програмного продукту буде використаний стандарт HTML5.

CSS ((/siːɛsɛs/ англійські каскадні таблиці стилів - каскадні таблиці стилів) - формальна мова для опису зовнішнього вигляду документа, написаного за допомогою мови розмітки.

Що ви можете зробити за допомогою CSS? CSS — це мова стилів, яка визначає відтворення документів HTML. Наприклад, CSS має справу зі шрифтами, кольором, полями, лініями, висотою, шириною, фоновими зображеннями, розташуванням елементів і багатьма іншими речами.

HTML можна (неправильно) використовувати для дизайну веб-сайтів. Але CSS надає більше можливостей і є більш точним і складним. CSS наразі підтримується всіма браузерами.

Поява CSS зробила революцію у світі веб-дизайну. Особливі переваги CSS:

- керування відображенням декількох документів за допомогою однієї таблиці стилів;
- більш точний контроль за зовнішнім виглядом сторінок;
- різні види для різних носіїв (екран, друк тощо);
- складна і витончена техніка проектування.

Враховуючи мову JavaScript! Варто сказати кілька слів про новий офіційний стандарт цієї мови програмування - ECMAScript 6.

Сама мова JavaScript вдосконалюється. Поточний стандарт ECMAScript 5 включає нові можливості для розробки, ECMAScript 6 стане кроком вперед у покращенні синтаксису мови.

Сучасні браузери вдосконалюють свої механізми, щоб прискорити виконання JavaScript, виправити помилки та намагатися дотримуватися стандартів.

Дуже важливо, щоб нові стандарти HTML5 і ECMAScript зберігали максимальну сумісність з попередніми версіями. Це дозволяє уникнути проблем із вже існуючими програмами.

Втім, із «суперсучасними речами» залишається невелика проблема. Іноді браузери намагаються включити нові функції, які ще не повністю описано в стандарті, але настільки цікаво, що розробники просто не можуть дочекатися.

Однак з часом стандартні зміни і браузери повинні адаптуватися до них, що може призвести до помилок у вже написаному JavaScript-код на основі старої реалізації. Тому варто добре подумати, перш ніж втілювати в життя такі «супернові» рішення.

При цьому всі браузери збігаються до стандарту, і відмінності між ними вже набагато менше, ніж кілька років тому.

Існують також компілятори, які беруть код, який використовує функції майбутніх стандартів JavaScript, і перетворюють його на старішу версію, зрозумілу всім браузерам.

Node.js - це програмна платформа, яка переводить JavaScript в машинний код і перетворює JavaScript з вузькоспеціалізованої мови на мову загального призначення з власним середовищем розробки. Node.js додає можливість підключення зовнішніх бібліотек, написаних різними мовами, JavaScript для взаємодії з пристроями введення-виводу через API. Node.js в основному використовується на сервері, але можна розробляти на Node.js та настільні віконні програми з використанням електрона.

Однією з головних рис Node.js є швидкість. Код JavaScript, що працює в Node.js, може бути в два або два рази швидше, ніж код, написаний такими мовами, як Java або C, і на кілька порядків швидше, ніж мови, що інтерпретуються, такі як Python або Ruby.

Причина в неблокуючій архітектурі платформи, яка орієнтована на роботу з асинхронними функціями. Ще однією незаперечною перевагою є використання однієї мови як для браузерної розробки, так і для серверної, що полегшує завдання створення ізоморфних програм з коду браузера ssr.

Оскільки розроблена програмна програма вимагає структурованої бази даних, а сервери зберігають дані користувача, тести та результати, кращим вибором стала MySQL, оскільки вона не вимагає багато часу та ресурсів для встановлення та налаштування, але задовольняє всі потреби.

React розроблений Facebook і показує свою високу ефективність усередині динамічних програм з високим трафіком. Він вважається найшвидшим JS-фреймворком, хоча його можна назвати фреймворком, тому що у MVC (Model-View-Controller) у моделі React.js він виступає як «V» і легко інтегрується в будь-яку архітектуру. Завдяки використанню дерева ShadowDOM забезпечується підвищена продуктивність у порівнянні з Angular 1. Крім того, компоненти React можна створювати та повторно використовувати в інших програмах.

2.3 Алгоритм тестування пацієнтів

Проектування алгоритмів які, зазвичай, використовуються для тестування.

На початковому етапі проектування інформаційної системи необхідно визначити ролі користувачів системи та їх функціональні можливості. Було визначено три ролі користувача:

Адміністратор (психолог) і користувач (студент), а також неавторизований користувач.

Адміністратор - авторизований користувач, який має доступ до панелі керування тестами, що містяться в програмі. Може додавати, видаляти та редагувати тести та студентів в системі.

Користувач - авторизований користувач, має функціонал неавторизованого користувача, крім того, додано можливість перегляду каталогу тестів, початку тестування, завершення тестування та перегляду результатів, коментарів психолога.

Неавторизований користувач може зареєструватись як студент та авторизуватись під своєю роллю.

На рисунку 2.7 показані варіанти використання системи.

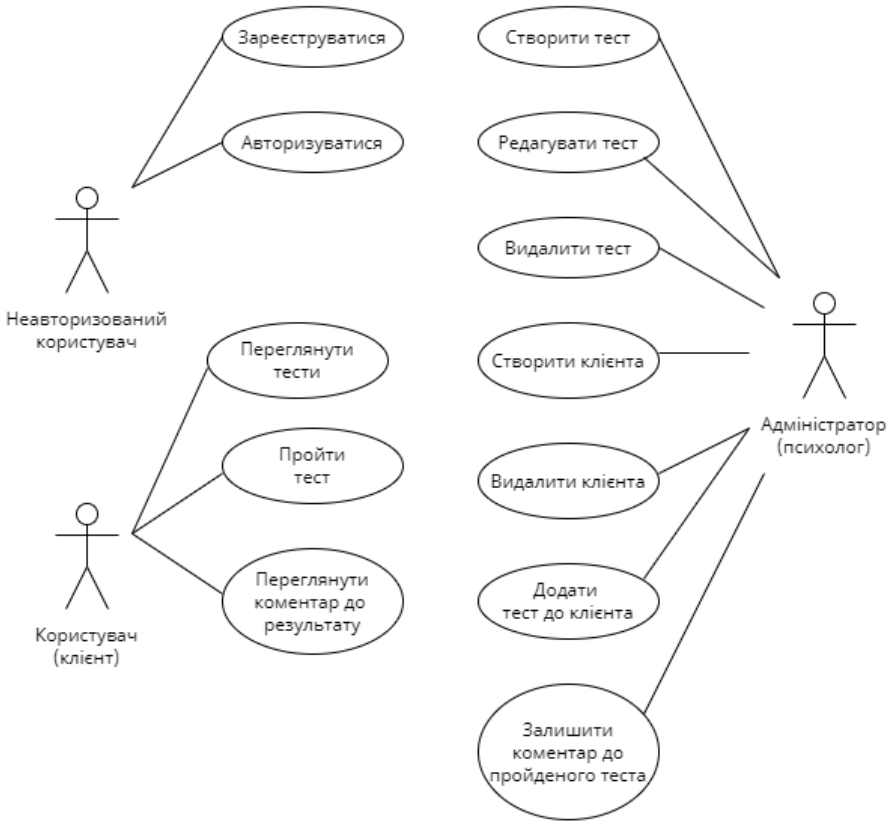


Рисунок 2.2 - Функціональні можливості користувачів

Якщо розглядати психологічне тестування всередині групи як процес, то початком або ініціатором цього процесу є розпорядження керівництва про проведення тестування, тобто обов’язкового тестування на 1 курсі. Потім учні виконують інструкції, підготовлені психологом. Психолог аналізує отриману під час тестування інформацію та формує підсумковий звіт для керівництва, при необхідності додатково запрошує студентів до індивідуальної роботи. На рисунку 2.8 наведена діаграма в нотації IDEF0, яка дозволяє зрозуміти входи, виходи, механізми та елементи керування, залучені в цей процес.

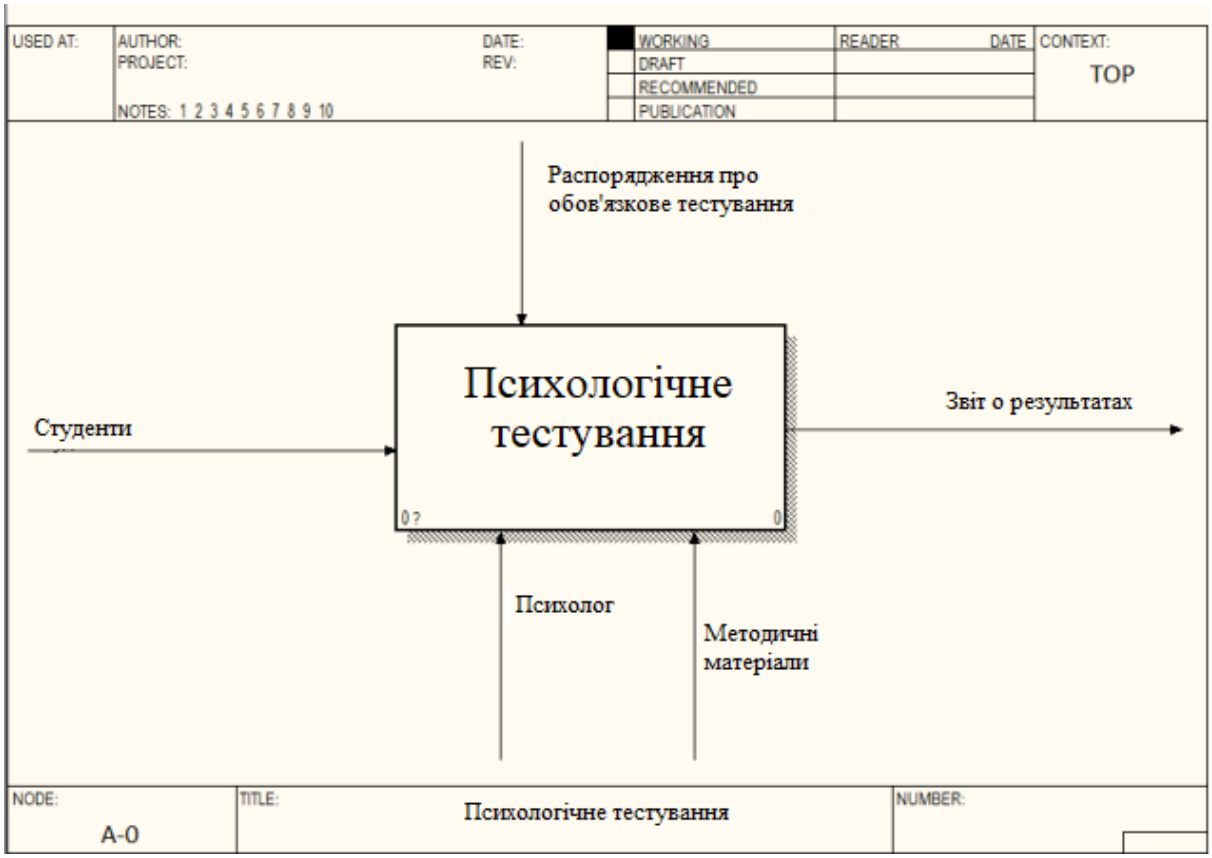


Рисунок 2.3 - Процес організації психологічного тестування як контекстної діаграми

Процес тестування не є атомарним і цікавий для більш детального розуміння процесу.

Автоматизація процесу психологічного тестування знизить навантаження на всіх учасників процесу, в тому числі студентів і психологів, що значно скоротить тимчасові та фінансові витрати. Розклад процесу тестування показано на рисунку 2.9.

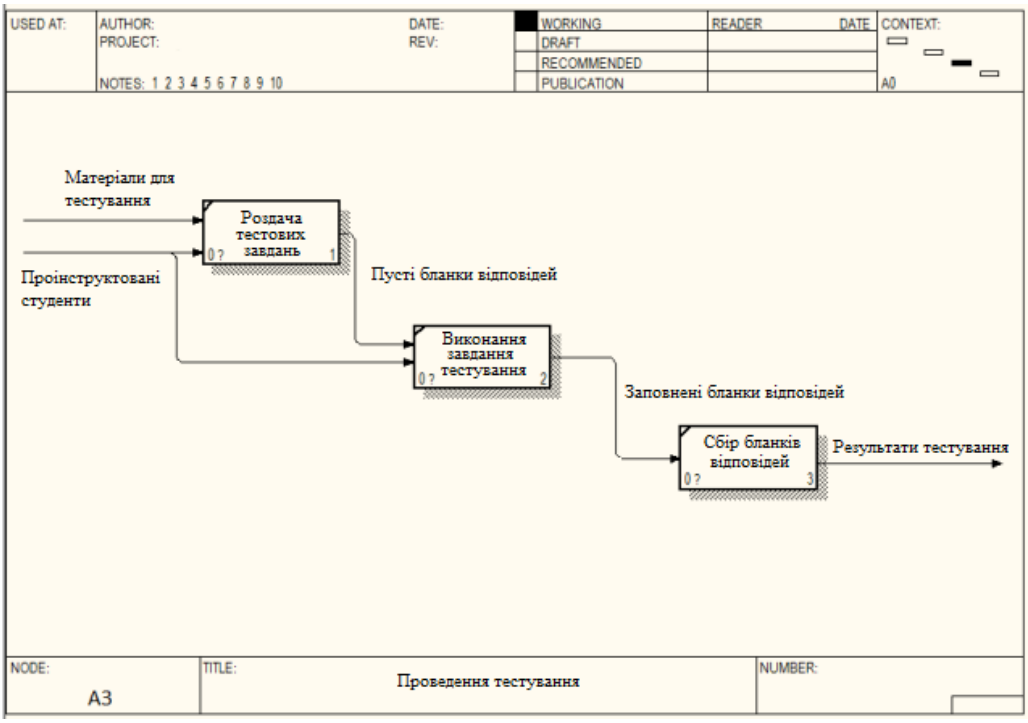


Рисунок 2.4 - Декомпозиція процесу «Провести тестування»

Проведення психологічного тестування дозволяє оцінити ситуацію в колективі і, як наслідок, сформувати сприятливий психологічний клімат у групі та сприяти зростанню інтересу учнів до навчання.

Відразу слід сказати, що існує багато завдань для тестування пацієнтів, які, в принципі, неможливо вирішити іншими методами, крім експерименту. І експеримент не можна звести лише до пробних тестів (як це іноді трактують у літературі).

На сьогоднішній день є досвід впровадження проектної форми роботи в систему вищої освіти.

Психодіагностика. За допомогою спеціальних тестів і методик вивчаються особистісні особливості учнів (опитувальник «Ролі майстра групової роботи», вправа «Машина», методика розподілу ролей у колективі за М. Бельбіним та ін.). Процедура психологічної діагностики дозволяє сформувати уявлення про студента як про члена команди проекту. Для отримання повної оцінки ситуації студенти поділяються на групи. Завдання – розв’язувати випадки та вивчати

відповідний матеріал. Мета – створити умови для індивідуальної та командної роботи. Тут вибудовується єдиний комунікативний простір всередині команд.

2.4 Вимоги до технічного забезпечення

Тестування проводиться зазвичай у кількох етапах життєвого циклу, коли розробляється програмне забезпечення. Технологія тестування програмного забезпечення містить такі етапи:

- визначення функціоналу, що підлягає і не підлягає тестуванню;
- формування підходів, що використовуються для даного продукту;
- написання тест кейсів;
- розробка критерію проходження тестів;
- визначення вимог середовища проведення тестування;
- проведення тестування та оцінка результатів;
- звітність результатів.

Системні вимоги:

- операційна система Windows 7 чи наступних версій, Mac OS, Linux OS;

Апаратні вимоги:

- досить вільної оперативної пам'яті комп'ютера для запуску браузеру;
- 32 або 64-розрядний процесор із тактовою частотою 1GHz;
- підключення до мережі інтернет.

Тестування проводилося на ноутбуці компанії DELL із процесором Intel Core i3-6060U, тактовою частотою 2GHz, операційною системою Windows 10 та за наявності програми Microsoft Word 2016 та програми Google Chrome v101.

2.5 Проектування архітектури програмного забезпечення

Для підвищення модульності, безпеки та масштабованості системи серверна частина програми поділена на 2 компоненти – бекенд-сервер і фронтенд-сервер. Крім того, поділ програми на модулі допомагає спростити процес розробки.

Бекенд-сервер виконує такі ролі:

1) робота з базою даних;

2) виконання бізнес-логіки.

Фронтенд-сервер відповідає за такі завдання:

1) вивантаження клієнтської програми для користувача;

2) обробка запитів користувачів;

3) керування сесіями користувачів.

Варто зазначити, що зв'язок між двома серверами є односпрямованим. Бекенд-сервер нічого не знає про існування зовнішнього сервера, тоді як інтерфейсний сервер не може повноцінно функціонувати без доступу до API серверного сервера.

Існує три основних підходи до розробки клієнтської програми:

1) односторінкова заявка (SPA);

2) багатосторінковий додаток (MPA);

3) гібридне застосування (HPA).

Односторінкові програми характеризуються більш чуйним інтерфейсом, тоді як багатосторінкові програми менше навантаження на комп’ютер користувача. Гібридні програми поєднують переваги та недоліки обох архітектур. У цій роботі був реалізований другий варіант – багатосторінковий додаток.

2.5.1 Проектування бази даних

У ході проектування архітектури веб-додатка була розроблена логічна схема бази даних. Логічна схема бази даних відрізняється від фізичної тим, що вона відображає модель даних разом із відносинами і не залежить від конкретної системи керування базою даних, яка використовується. На рисунок 2.10 показана модель предметної галузі (бази даних).

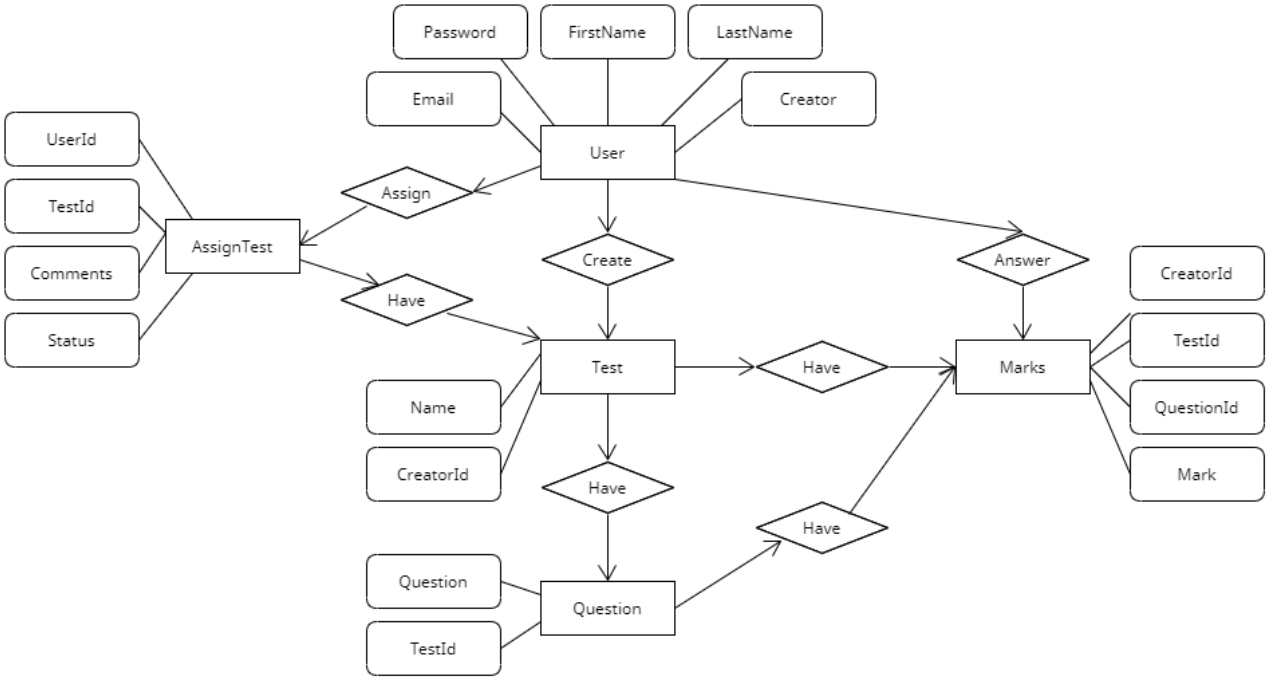


Рисунок 2.5 – Модель предметної області

В ході складання діаграми було визначено 9 сутностей для зберігання в базі даних:

- об’єкт «Тест» містить атрибути назви тесту та зовнішній ключ ідентифікатора психолога;
- сутність «Запитання» містить атрибути змісту запитання, зовнішній ключ ідентифікатора тесту;

– сутність «Відповідь» містить атрибути вмісту відповіді, зовнішні ключі: ідентифікатор запитання, ідентифікатор клієнта, ідентифікатор тесту;
– суть «Призначення» містить коментар психолога, статус та зовнішні ключі: ідентифікатор тесту, ідентифікатор користувача;
– сутність «Користувач» містить атрибути ім'я користувача, фамілію користувача, пароль у вигляді хешу, електронна пошта та ідентифікатор психолога, якщо це клієнт;

Склад сутностей «Тест» і «Запитання» описує тести, що зберігаються в системі.

Сутність «Вибір» є підсумковим файлом для аналізу пройденого тестування.

Об’єкт «Користувач» зберігає інформацію про користувачів

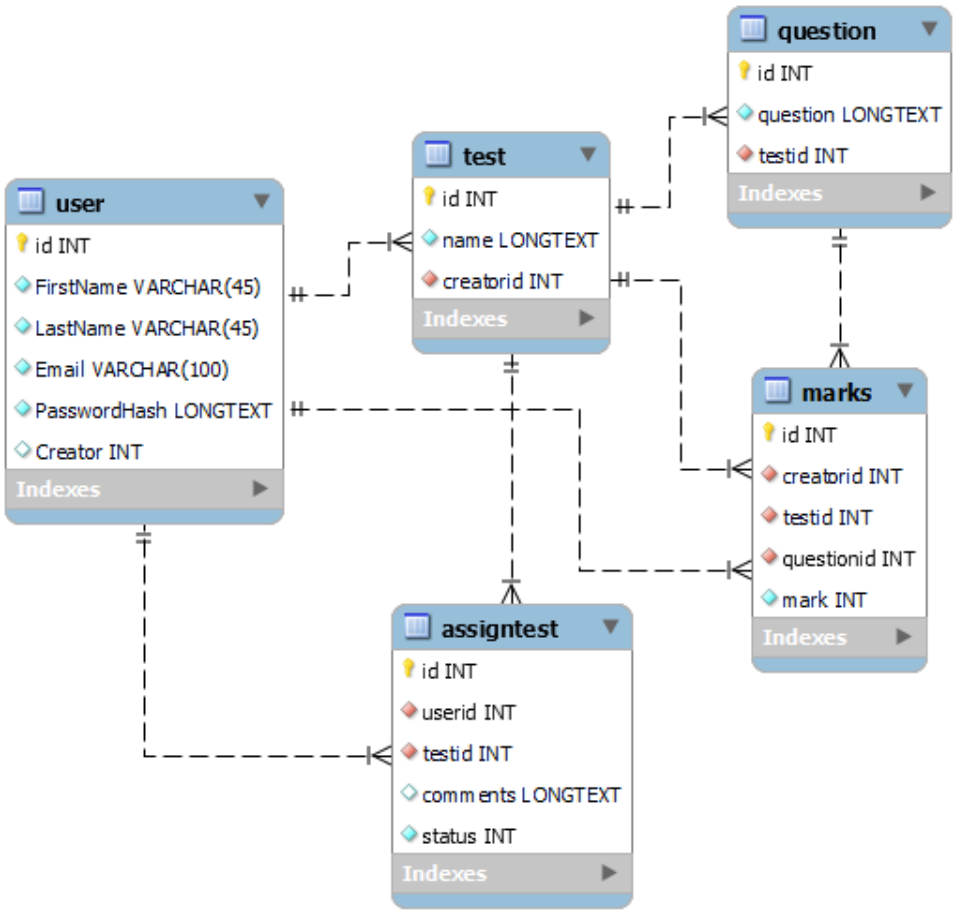


Рисунок 2.6 - Логічна схема бази даних

2.5.2 Моделювання

Розроблена автоматизована система психологічного тестування студентів загальноосвітніх та вищих навчальних закладів є онлайн-тестуванням і дозволяє виявити необхідні показники.

Призначення системи:

- комплексна оцінка показників;
- висока швидкість обробки результатів;
- онлайн-тестування;
- забезпечення стандартних умов тестування для всіх студентів;
- точний контроль процедури тестування (пропускати завдання не можна, фіксується час виконання кожного завдання, що особливо важливо в тестах для визначення рівня інтелектуального розвитку).

Ми проаналізуємо процес онлайн-скринінгу, а також розглянемо процес тестування та виділимо основні параметри, які впливають на результат тесту.

Для розуміння особливостей і закономірностей психічного розвитку було обрано лонгітюдний тип дослідження, що представляє собою багаторазове обстеження одних і тих самих осіб протягом тривалого періоду часу [2].

Вивчення та аналіз зазначених показників проводилися протягом навчального року з періодичністю 1 раз на 3 місяці. Загальна схема тестування показана на рисунок один.

Модель прецедентів описує концепт взаємодії клієнта та психолога, що допомагає зрозуміти порядок виконання програми для кожного. Клієнт та психолог мають змогу авторизуватися у системи та працювати з тестами з різних сторін. Клієнт може розпочати тестування та зберегти свої відповіді, а психолог аналізувати відповіді та створювати тести для клієнта.

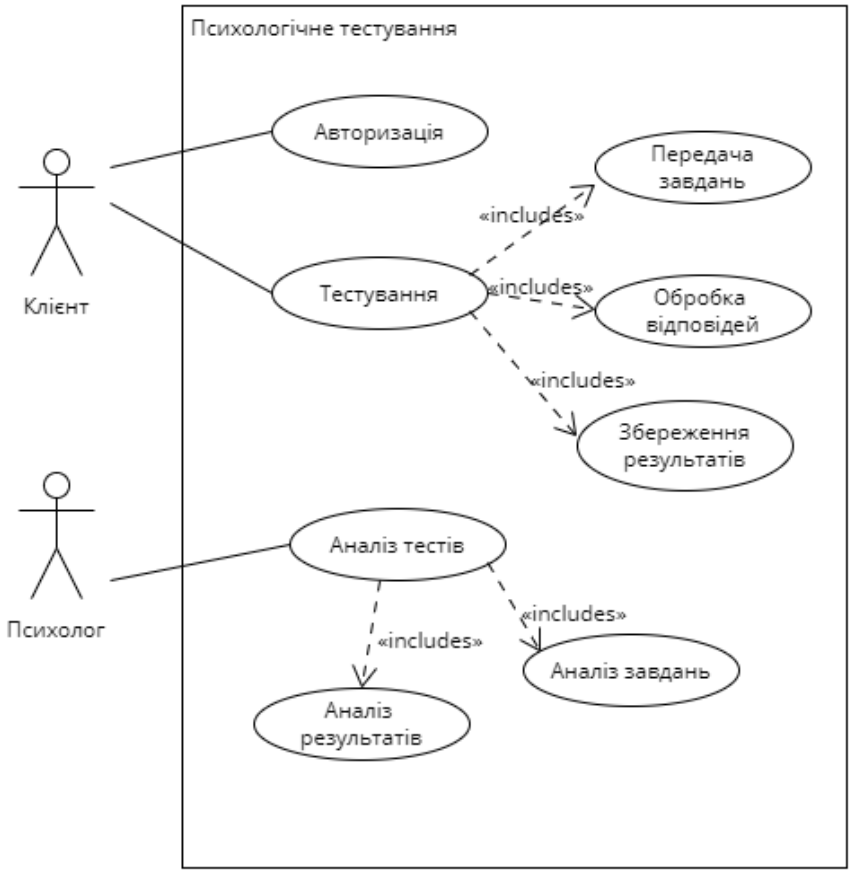


Рисунок 2.7 – Модель прецедентів

Діаграма послідовності відображає порядок тестування клієнтів за часом, де психолог спочатку дає користувачу логін та пароль для авторизації системи, клієнт проходить тест та зберігає відповіді у БД. Психолог по завершенню тестування перевіряє результати.

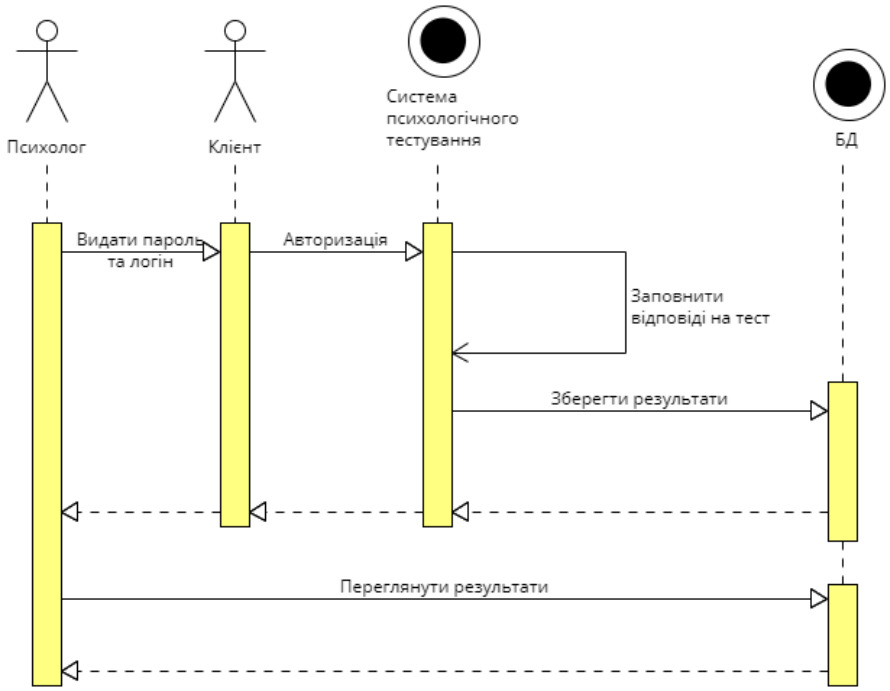


Рисунок 2.8 – Моделювання процесу тестування

У базі даних системи зберігається інформація про студентів, психологів, види тестів, запитання та відповіді студентів (рисунок 2.13).

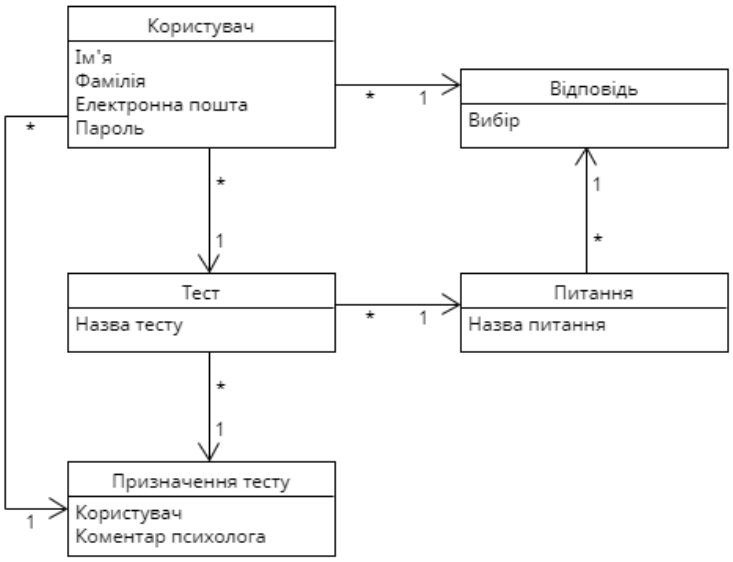


Рисунок 2.9 – Модель предметної галузі

Діаграма пакетів – відображає усі залежності між різними пакетами, з яких складається модель.

Основним пакетом являється Server, який включає в себе серверну частину, що складається з Routes, Controllers, Helpers та node_modules, деякі з яких підключені до DB. Client – клієнтська частина, що має пакети build, src, public та node_modules. Ця частина запитує серверну частину Routes з пакетів build та src (рисунок 2.14).

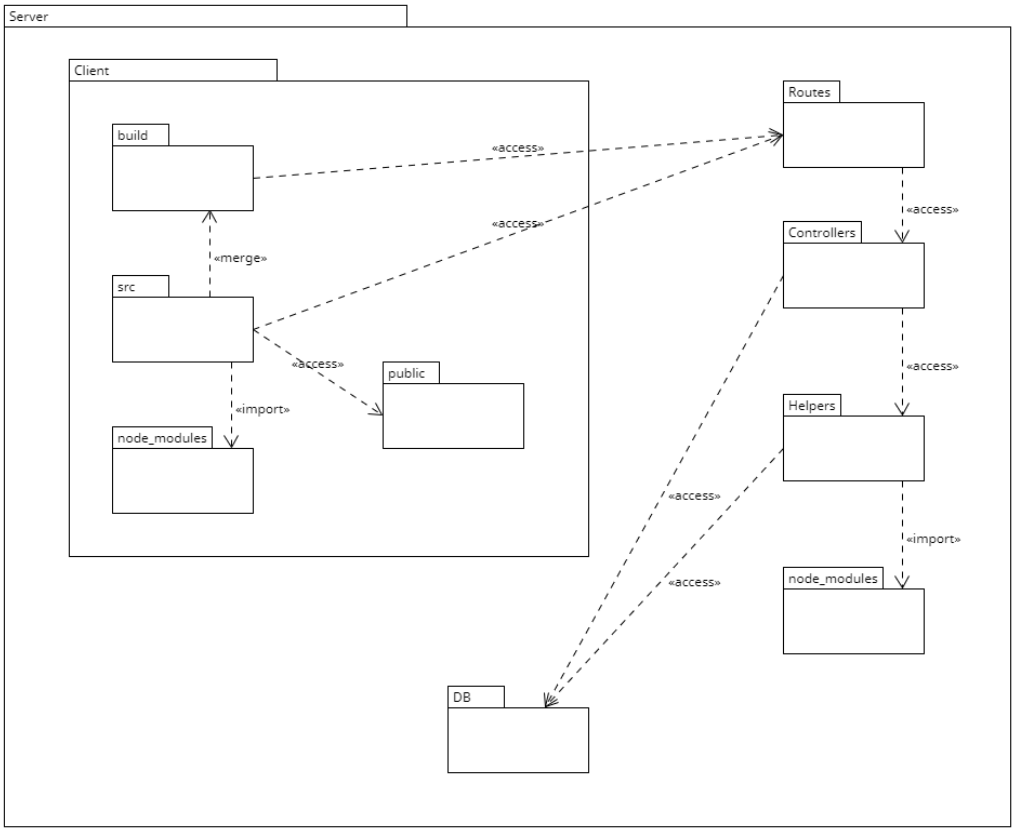


Рисунок 2.10 – Модель пакетів

Діаграма розгортання – відображає усі вузли, компоненти та об’єкти додатку, які будуть активні при використанні.

Для розгортання додатку будуть використовуватись сервіси AWS, бо це найбільш актуальний та ефективний сервіс для веб-додатків, який включає в себе усі потрібні компоненти розгортання та менеджменту. Elastic Beanstalk дозволяє:

– налаштовувати CI\CD за допомогою CodePipeline та GitHub;

– створювати підмережу для підключення до EC2 серверу та запуску створеного Node.JS;
– менеджменту логів за допомогою S3 bucket;

База даних на основі RDS створюється на тому ж VPS, що й EC2 instance.

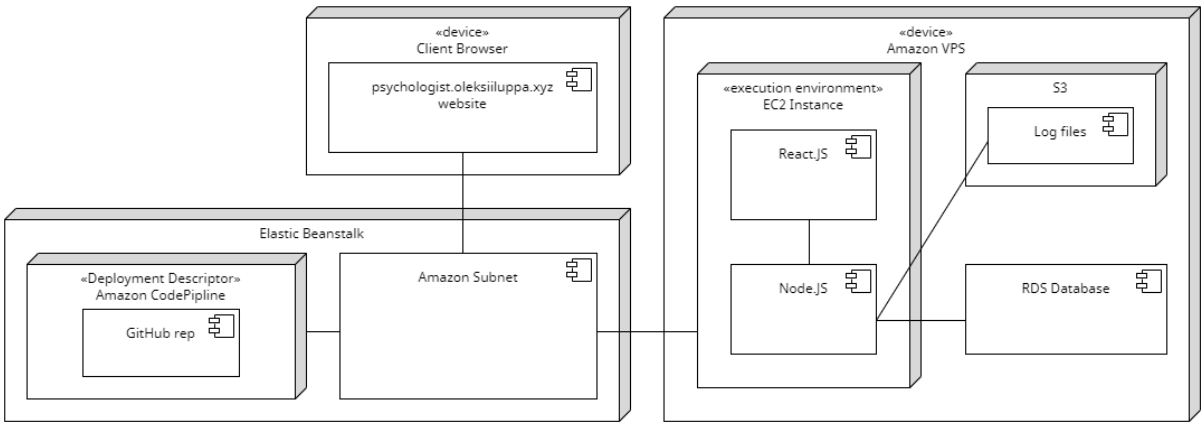


Рисунок 2.11 – Модель розгортання

Модель діяльності описує флоу користувача з різними ролями. Починаючи від реєстрації та авторизації, закінчуючи навігацією по сайту, діями на ньому та логаутом (рисунок 2.16).

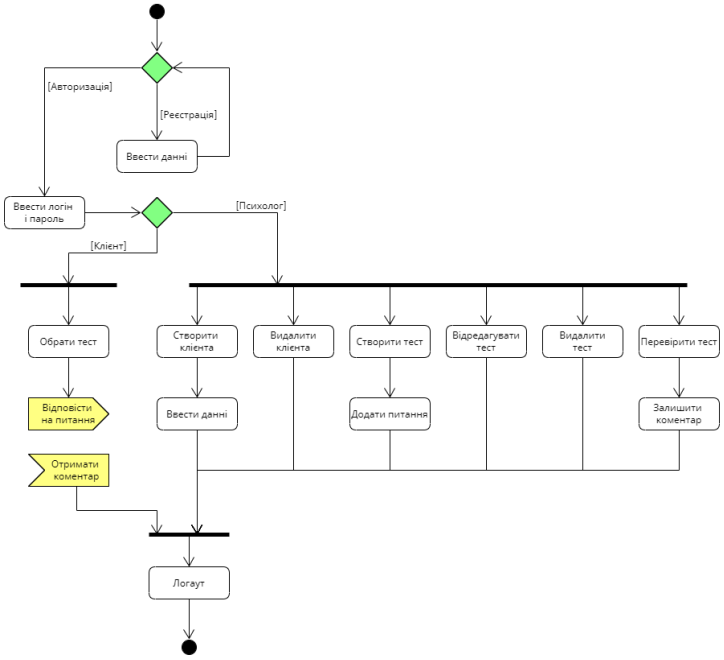


Рисунок 2.12 – Модель діяльності

2.6 Висновки до глави 2

У другому розділі було створено технічне завдання, описано середовище та системи, необхідні для створення власного додатка. Були обрані інструменти розробки, необхідні вимоги до технічної підтримки та програми, які будуть використовуватись для розробки, тип бази даних та інструменти для керування нею.

Запроектовано алгоритм тестування пацієнтів та структуру бази даних, до якої додано діаграму предметної області та логічну модель. Також, були змодельовані:

- діаграма функціональної можливості користувачів;
- діаграма прецедентів;
- модель процесу тестування;
- модель предметної галузі;
- модель пакетів;
- модель розгортання;
- модель діяльності.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Основні етапи розробки

Завдяки бурхливому розвитку науки інформатики і проникненню їх у різні галузі народного господарства слово " алгоритм " стало найпоширенішим і найбільш вживаним у життєвому плані поняттям для широкого кола фахівців. Більше того, з переходом до інформаційного суспільства алгоритми стають одним з найважливіших факторів цивілізації.

Відомо, що математична теорія алгоритмів склалася зовсім у зв'язку з бурхливим розвитком інформатики та обчислювальної техніки, а виникла надрах математичної логіки на вирішення її проблем. Вона, перш за все, дуже вплинула на світогляд математиків і їх науку.

Тим не менш, взаємовплив теоретичних областей, пов'язаних з обчислювальною технікою, та теорії алгоритмів також, безсумнівно.

Теорія алгоритмів вплинула на теоретичне програмування. Зокрема, велику роль теоретичному програмуванні відіграють моделі обчислювальних автоматів, які, за суттевістю, є обмеженнями тих представницьких обчислювальних моделей, створених раніше у теорії алгоритмів. Трактування програм, як об'єктів обчислення, оператори, що використовуються для складання структурованих програм (послідовне виконання, розгалуження, повторення), прийшли в програмування з теорії алгоритмів. Зворотний вплив виявилося, наприклад, у тому, що виникла потреба у створенні та розвитку теорії обчислювальної складності алгоритмів. Отже, можна сказати, що теорія алгоритмів застосовується у інформатиці, а й у інших галузях знань.

Поняття алгоритму та його властивості

Слово «алгоритм» означає «процес або набір правил, яких слід дотримуватися під час обчислень або інших операцій з вирішення проблем».

Тому Алгоритм відноситься до набору правил/інструкцій, які крок за кроком визначають, як має виконуватися робота, щоб отримати очікувані результати.

Це можна зрозуміти, взявши приклад приготування за новим рецептом. Щоб приготувати новий рецепт, потрібно читати інструкції та кроки та виконувати їх по черзі в заданій послідовності. Отриманий таким чином результат – нове блюдо, приготоване ідеально. Аналогічно, алгоритми допомагають виконати завдання в програмуванні, щоб отримати очікуваний результат.

Розроблений алгоритм не залежить від мови, тобто це прості інструкції, які можна реалізувати будь-якою мовою, але результат буде таким же, як і очікувалося.

Оскільки не слід дотримуватись жодних письмових інструкцій, щоб приготувати рецепт, а лише стандартний. Так само не всі письмові інструкції з програмування є алгоритмами. Для того, щоб деякі інструкції були алгоритмом, вони повинні мати такі характеристики:

Чітко і однозначно: алгоритм повинен бути зрозумілим і недвозначним. Кожен із його кроків має бути зрозумілим у всіх аспектах і вести лише до одного значення.

Добре визначені вхідні дані: якщо алгоритм каже приймати вхідні дані, це повинні бути чітко визначені вхідні дані.

Добре визначені результати: алгоритм повинен чітко визначити, який вихід буде отримано, і він також повинен бути чітко визначений.

Скінченність: Алгоритм повинен бути скінченним, тобто він не повинен опинитися в нескінченних циклах або тому подібних.

Можливість: Алгоритм повинен бути простим, загальним і практичним, щоб його можна було виконувати за наявності доступних ресурсів. Він не повинен містити якусь майбутню технологію чи щось таке.

Незалежний від мови: розроблений алгоритм повинен бути незалежним від мови, тобто це повинні бути просто прості інструкції, які можна реалізувати будь-якою мовою, але результат буде таким же, як і очікувалося.

Переваги алгоритмів:

Це легко зрозуміти.

Алгоритм — це поетапне представлення рішення заданої задачі.

В алгоритмі проблема розбивається на менші частини або кроки, отже, програмісту легше перетворити її на справжню програму.

Недоліки алгоритмів:

- написання алгоритму займає багато часу, тому займає багато часу.
- розгалуження та циклічні оператори важко показати в алгоритмах.
- як розробити алгоритм?
- для того, щоб написати алгоритм, необхідні такі речі як передумова:
- задача, яку потрібно вирішити за допомогою цього алгоритму.
- обмеження проблеми, які необхідно враховувати при розв’язуванні задачі.
- вхідні дані, які потрібно зробити для вирішення проблеми.
- результат, якого слід очікувати, коли проблема буде вирішена.
- рішення цієї задачі, в заданих обмеженнях.

Потім алгоритм записується за допомогою вищевказаних параметрів так, щоб він вирішував задачу.

Приклад: Розглянемо приклад, щоб додати три числа та надрукувати суму.

Крок 1: Виконання попередніх умов

Як обговорювалося вище, для того, щоб написати алгоритм, повинні бути виконані його передумови.

Задача, яку потрібно розв’язати за цим алгоритмом: Додайте 3 числа та виведіть їх суму.

Обмеження задачі, які необхідно враховувати при розв’язуванні задачі: числа повинні містити тільки цифри і ніяких інших символів.

Вхідні дані, які потрібно зробити для розв’язання задачі: три числа, які потрібно додати.

Вихід, якого слід очікувати, коли задачу буде розв’язано: сума трьох чисел, взятих як вхідні дані.

Розв’язання цієї задачі за наведених обмежень: Рішення складається з додавання 3 чисел. Це можна зробити за допомогою оператора «+», або порозрядно, або будь-яким іншим способом.

Крок 2: Розробка алгоритму

Тепер давайте розробимо алгоритм за допомогою наведених вище умов:

Алгоритм додавання 3 чисел і виведення їх суми:

СТАРТ

Оголосити 3 цілі змінні num1, num2 і num3.

Візьміть три числа, які потрібно додати, як вхідні дані у змінних num1, num2 та num3 відповідно.

Оголосіть цілу змінну sum, щоб зберегти підсумкову суму трьох чисел.

Додайте 3 числа та збережіть результат у сумі змінної.

Вивести значення змінної суми

КІНЕЦЬ

Крок 3: Тестування алгоритму шляхом його реалізації.

3.2 Розробка власного продукту

Архітектура програмного забезпечення дає пояснення того, як ваші системи поводяться на структурному рівні. Системи, які ви використовуєте, мають набір компонентів, розроблених для виконання певного завдання або набору завдань. Архітектура програмного забезпечення забезпечує фундамент, на якому все програмне забезпечення, яке є у компанії, можна змінити, створити або вилучити з експлуатації.

Архітектура програмного забезпечення впливає на якість, продуктивність, обслуговування та успіх системи на основі дизайну. Не розглядаючи архітектуру програмного забезпечення на регулярній основі, компанія відкриває себе для довгострокових наслідків, і проблеми, які можуть поставити їх системи під загрозу поломки, злому або низької продуктивності.

У сучасних системах існують загальні шаблони в архітектурі програмного забезпечення, які називаються архітектурними системами для програмного забезпечення. У більшості випадків для створення цілісної системи використовується кілька різних архітектурних систем, особливо для систем, які створювалися роками або працювали, або тих, які були побудовані різними розробниками.

Для роботи сайту було розроблено наступний алгоритм:

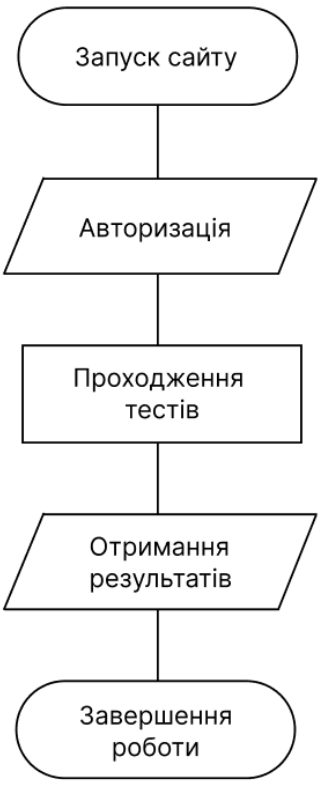


Рисунок 3.1 – Алгоритм сайту

3.3 Розробка архітектури програмного забезпечення

Інтерфейс був частково реалізований за допомогою фреймворку Bootstrap. Його відмінними рисами є висока швидкість створення, кросплатформеність і багато іншого, а інша частина написана від руки відповідно до макетів. Він також включає шаблони дизайну HTML і CSS для веб-форм, кнопок, міток, блоків

навігації та інших компонентів веб-інтерфейсу, включаючи розширення JavaScript.

На багатьох сторінках повторюється один і той же елемент - меню керування (верхній колонтитул) і нижній колонтитул (нижній колонтитул сторінки). Щоб не налаштовувати відображення цих елементів на кожній сторінці окремо, створюється загальний шаблон сторінки. Усі сторінки, які реалізують шаблон, що містить меню та нижній колонтитул, також міститимуть меню та нижній колонтитул, а також макет та макет сторінки.

Крім того, меню було реалізовано в двох форматах, для авторизованого та неавторизованого користувача, що дозволяє обмежити можливість переміщення по сайту людини, яка ще не зареєструвалася на цій платформі.

3.3.1 Розробка структури даних

Структура даних – це спосіб збору та організації даних таким чином, щоб ми могли ефективно виконувати операції з цими даними. Структури даних – це відтворення елементів даних у термінах певного зв’язку для кращої організації та зберігання.

Ми можемо організувати ці дані у вигляді запису на зразок Запису гравця, у якому буде вказано ім’я гравця та вік. Тепер ми можемо збирати та зберігати записи гравця у файлі чи базі даних як структуру даних.

Структура даних —спосіб організації та зберігання даних, щоб ефективно виконувати операції. Доступ, вставка, видалення, пошук і сортування даних є одними з основних операцій, які можна виконувати за допомогою структур даних. Не всі структури даних можуть ефективно виконувати ці операції, це призвело до розробки різних структур даних. Скажімо, вам потрібно знайти конкретну книгу в неорганізованій бібліотеці, це завдання займе величезну кількість часу. Подібно до того, як бібліотека організовує свої книги, нам потрібно організувати наші дані так, щоб операції можна було виконувати ефективно.

Для створення структури даних використовується попередньо визначений тип даних, наприклад Integer, Stings, Boolean. Використання типу даних залежить від вимог користувача та типу даних, які вони хочуть зберігати. Тепер давайте зануримося в структури даних, які використовуються в нашому повсякденному програмуванні, а також розглянемо підтримку структур даних для різних мов програмування.

Розгортання бази даних

Для коректної роботи бази даних і підключення до неї з будь-якого місця мережі інтернет, потрібен клауд сервіс, який би задовольнив потреби нашого додатку. Кращим варіантом виступає AWS RDS сервіс, який дозволяє легко та швидко розгортати MySQL БД безкоштовно та налаштовувати її під будь-які потреби.

Взагалі, Amazon Relational Database Service (або Amazon RDS) — це розподілена реляційна база даних (РБД), яка надається як сервіс від Amazon Web Services (AWS). Є вебсервісом, який виконується у «хмарі». Спеціально розроблений для спрощення установки, простоти обслуговування та легкого масштабування РБД в застосунках. Такі складні процеси, як оновлення програмного забезпечення бази даних, проведення резервного копіювання, повернення до раннього стану (відновлення) відбувається автоматично.

Почнемо зі створення RDS instance:

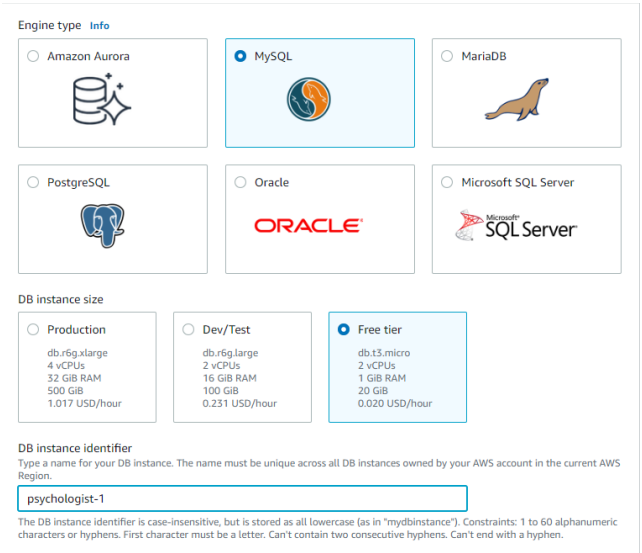


Рисунок 3.2 – Створення RDS instance

Після успішної ініціалізації бази даних, потрібно налаштувати доступ до неї з інших IP для подальшої розробки. Для цього треба налаштувати Security Group VPC, на якому вона розміщена.

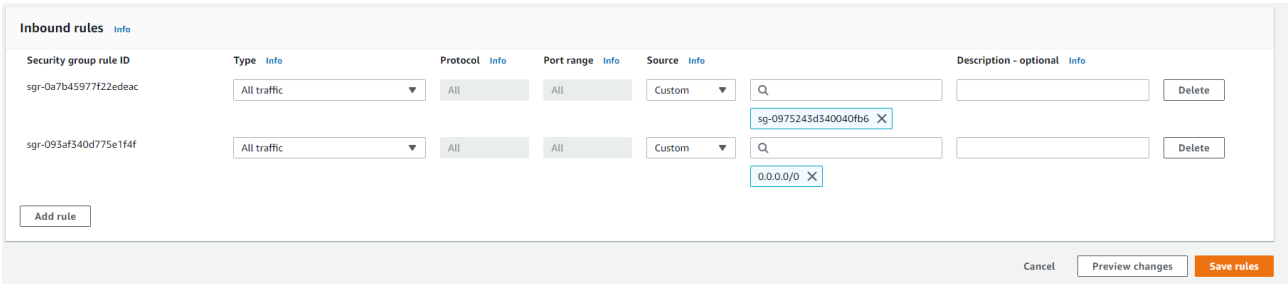


Рисунок 3.3 – Налаштування Security Group

За попередньо створеною моделлю даних відтворюємо усі таблиці та зв’язки між ними, підключившись до БД за допомогою MySQL Workbench. Використовуємо Primary Key та Foreign Key для чіткого налаштування зв’язку між полями.

Маємо таку структуру бази даних:

Table	Column	Type	Default Value	Nullable
assigntest	id	int		NO
assigntest	userid	int		NO
assigntest	testid	int		NO
assigntest	comments	longtext		YES
assigntest	status	int	1	NO
marks	id	int		NO
marks	creatorid	int		NO
marks	testid	int		NO
marks	questionid	int		NO
marks	mark	int		NO
question	id	int		NO
question	question	longtext		NO
question	testid	int		NO
test	id	int		NO
test	name	longtext		NO
test	creatorid	int		NO
user	id	int		NO
user	FirstName	varchar(45)		NO
user	LastName	varchar(45)		NO
user	Email	varchar(100)		NO
user	PasswordHash	longtext		NO
user	Creator	int	0	YES

Рисунок 3.4 – Структура БД

3.3.2 Написання програмного продукту

Для початку потрібно відтворити структуру модулів додатку для серверної та клієнтської частини. У корені кожної частини має бути файл package.json для менеджменту потрібних бібліотек та коректного запуску обох частин.

Рисунок 3.5 – Структура модулів серверу

За допомогою команд «npm i {module}» встановлюємо всі потрібні для роботи бібліотеки.

У файл «app.js» додаємо зовнішні модулі для запуску сервера та встановлюємо початкові налаштування.

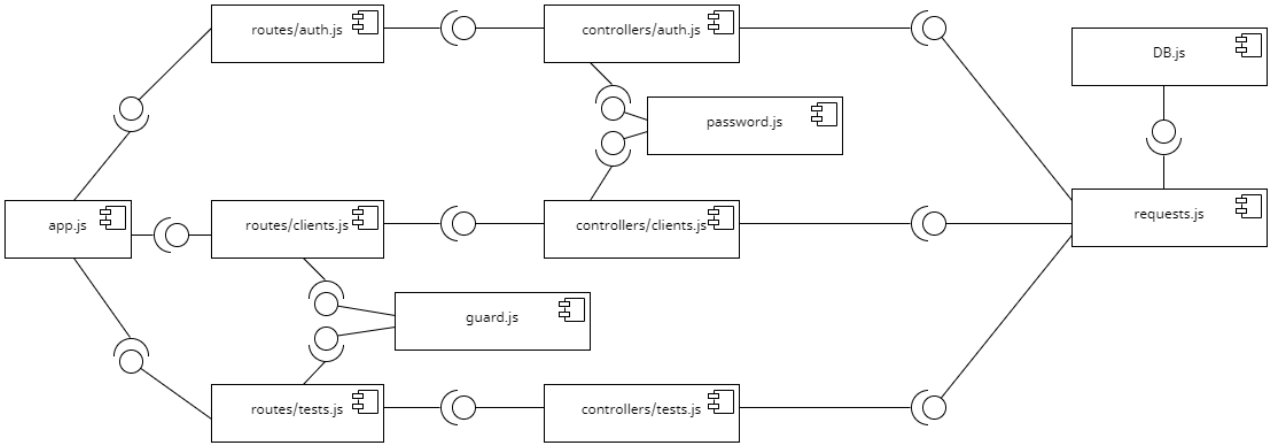


Рисунок 3.6 – Діаграма компонентів серверу

Так як сервер буде звертатись до локально створених при білді файлів клієнта, нам потрібно додати особливий get роут, що буде їх зчитувати:

```
app.use("/", express.static(path.join(__dirname, "client", "build")));
app.get("*", (req, res) => {
  res.sendFile(path.resolve(__dirname, "client", "build", "index.html"));
});
```

Рисунок 3.7 – Сервінг клієнтських файлів через сервер

Для зберігання сесії буде використана бібліотека «jsonwebtoken-refresh», яка дозволяє створювати JWT токени з лімітом у часі та даними у середині. Щоб підключити її до серверу потрібно додати наступний код:

```
app.use(
  cookieSession({
    name: "session",
    keys: ["token"],
  })
);
```

Рисунок 3.8 – Налаштування сесії на сервері

Для серверу у папці «helpers» створюємо наступні файли:

– controllerHandre.js – невеликий модуль, який перехоплює помилки при запитах на сервер:

```
const controllerHandler = (controller) => {
  return async (req, res, next) => {
    try {
      await controller(req, res, next);
    } catch (error) {
      next(error);
    }
  };
};

module.exports = controllerHandler;
```

Рисунок 3.9 – Файл controllerHandre.js

– DB.js – модуль, що повністю контролює підключення до БД та надсилає запити до неї;
– guard.js – контролює та валідує сесію, постійно оновлюючи її при запитах;
– password.js – невеликий хелпер, який хешує паролі для подальшого запису його в БД;
– constants.js – для менеджменту усіх константих значень, наприклад, HTTP коди відповідей з сервера, чи кольори для бека.

```
const guard = async (req, res, next) => {
  try {
    const token = req.session.token;
    if (!token) {
      return res.status(HttpCode.UNAUTHORIZED).json({
        message: "Session doesn't exist",
      });
    }

    jwt.verify(token, JWT_SECRET);
    const originalDecoded = jwt.decode(token, { complete: true });
    const refreshed = jwt.refresh(originalDecoded, "7d", JWT_SECRET);
    req.session.token = refreshed;
    req.user = originalDecoded.payload
    return next();
  } catch (error) {
    console.log(error);
    req.session.token = null;
    return res.status(HttpCode.UNAUTHORIZED).json({
      message: "Session expired",
    });
  }
};
```

Рисунок 3.10 – Файл guard.js та контроль сессії

У папці «routes» потрібно створити 3 файла, які будуть відповідати 3 блокам додатку для контролю запитів на сервер:

– auth.js – для модулю авторизації;
– clients.js – для модулю роботи з клієнтами;
– tests.js – для модулю роботи з тестами та їх результатами.

Ці модулі дуже схожі один на одного, тільки різняться шляхом запиту на контролером, який вони викликають. Важливо не забути імпортувати ці модулі до app.js та підключити до серверу.

Приклад auth.js:

```
const { Router } = require('express');
const router = Router();
const ctrl = require('../controllers/auth');
const ctrlHandler = require('../helpers/controllerHandler');

router.post('/signup', ctrlHandler(ctrl.signup));
router.post('/signin', ctrlHandler(ctrl.signin));
router.get('/refreshSession', ctrlHandler(ctrl.checkSession));
router.post('/logout', ctrlHandler(ctrl.logout));

module.exports = router;
```

Рисунок 3.11 – Роути для авторизації

Контролери також поділені на 3 блоки: auth.js, clients.js, tests.js та одни допоміжний блок – requests.js, який містить усі запити до бд.

Приклад одного з таких запитів:

```
const getTests = async (userId, creator) => {
      const results = await new DB().ExecuteQuery(`select t.* from test t where
                                                  t.creatorid=${userId}`);
      for(let r of results){
            const resultsQ = await new DB().ExecuteQuery(`select * from question
                                                  where testid=${r.id}`);
            const resultsA = await new DB().ExecuteQuery(`select count(*) as
                              assigns from assigntest where testid=${r.id}`);
            r['questions'] = resultsQ
            r['assigns'] = resultsA[0].assigns
      }
      return results.length > 0 ? results : null;
};
```

Рисунок 3.12 – Запит на отримання всіх тестів

У контролерах обробляються усі логічні помилки та повертаються коректні коди помилок та повідомлення для подальшого відображення на клієнтській частині.

Приклад функції контролеру авторизації:

```
const signup = async (req, res) => {
  try{
    const { email, password, firstName, lastName } = req.body;

    const user = await requests.getInfoByEmail(email);

    if (user) {
      return res.status(HttpCode.CONFLICT).json({
        success: false,
        message: 'User with this email already exist',
      });
    }

    const passwordHash = PasswordHelper.hashWithSalt(password,
process.env.SALT_SECRET);

    const addUserResult = await requests.addClient({ email, password: passwordHash,
firstName, lastName });

    return res.status(HttpCode.OK).json({
      message: 'Successfully registered',
      addUserResult
    });
  }
  catch(err){
    return res.status(HttpCode.SERVER_ERROR).json({
      message: err
    });
  }
};
```

Рисунок 3.13 – Контролер авторизації

Для клієнтської частини також потрібно встановити усі нод модулі, які будуть використовуватись та налаштувати початковий файл для рендеру усіх подальших сторінок за допомогою React.JS:

```
import React from "react";
import ReactDOM from "react-dom";
import { Provider } from "react-redux";
import { BrowserRouter } from "react-router-dom";
import "./index.css";
import App from "./App";
import { store } from "./redux/store";
ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </Provider>
  </React.StrictMode>,
  document.getElementById("root")
);
```

Рисунок 3.14 – Файл index.js

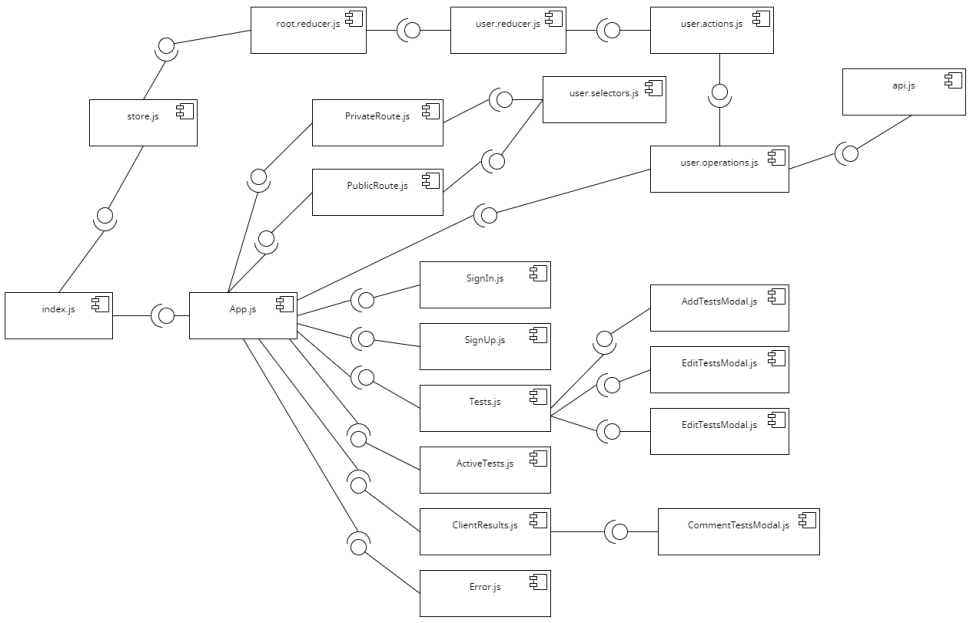


Рисунок 3.15 – Діаграма компонентів клієнту

Далі, для коректної навігації та переходів по різним сторінкам була створена модель станів:

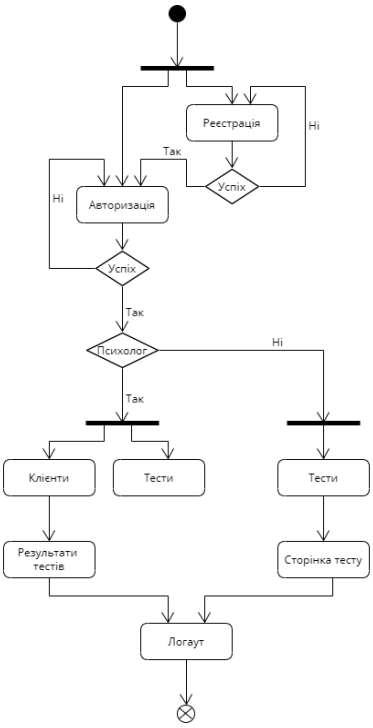


Рисунок 3.16 – Діаграма діяльності навігації

У проекті буде використана бібліотека «redux/toolkit» для зберігання та відтворення усіх відповідей з серверу, контролю станів різних компонентів. Для цього потрібно у папку redux імпортувати store та налаштувати всі частини: actions, operations, reducers, selectors.

Важливо налаштувати приватні та публічні роути для блокування доступу неавторизованим користувачам. Для цього створюємо два компоненти, які будуть це робити:

– PrivateRoute.js – якщо не існує активної сесії, перекидає користувача на /sign-in;

```
import { useSelector } from "react-redux";
import { getUser } from "../../redux/user/user.selectors";
import { Navigate, Outlet } from "react-router-dom";

const PrivateRoute = () => {
  const user = useSelector(getUser);
  return user ? <Outlet /> : <Navigate to="/sign-in" />;
};

export default PrivateRoute;
```

Рисунок 3.17 – Файл PrivateRoute.js

– PublicRoute.js – якщо існує активна сесія, перекидає користувача на головну сторінку.

```
import { useSelector } from "react-redux";
import { getUser } from "../../redux/user/user.selectors";
import { Navigate, Outlet } from "react-router-dom";

const PublicRoute = () => {
  const user = useSelector(getUser);
  return user ? <Navigate to="/"/> : <Outlet />;
};

export default PublicRoute;
```

Рисунок 3.18 – Файл PublicRoute.js

Підключаємо ці компоненти до App.js та налаштовуємо усі шляхи:

```
<Routes>
        <Route exat path="/sign-in" element={<PublicRoute />}>
          <Route path="/sign-in" element={<SingIn />} />
        </Route>

        <Route exat path="/sign-up" element={<PublicRoute />}>
          <Route path="/sign-up" element={<SingUp />} />
        </Route>

        <Route exat path="/" element={<PrivateRoute />}>
          <Route path="/" element={<Clients />} />
        </Route>

        <Route exat path="/tests" element={<PrivateRoute />}>
          <Route path="/tests" element={<Tests />} />
        </Route>

        <Route exat path="/tests/test" element={<PrivateRoute />}>
          <Route path="/tests/test" element={<ActiveTest />} />
        </Route>

        <Route exat path="/clients/test" element={<PrivateRoute />}>
          <Route path="/clients/test" element={<ClientResults />} />
        </Route>
</Routes>
```

Рисунок 3.19 – Підключеня роутів до App.js

Для запитів на сервер потрібно підключити бібліотеку Axios та створити спеціальний сервіс api.js, де будуть виконуватись та експортуватись усі запити.

```
import axios from "axios";

const clientApi = axios.create({
  withCredentials: true
})

export const signin = async (data) =>
  clientApi
    .post("/api/auth/signin", data)
    .then((result) => result)
    .catch((reason) => reason.response)

export const signup = async (data) =>
    clientApi
      .post("/api/auth/signup", data)
      .then((result) => result)
      .catch((reason) => reason.response)
```

Рисунок 3.20 – Приклад запитів авторизації

Для візуально приємного відображення повідомлень потрібно додати бібліотеку toastify та додати компонент Error.js, який буде отримувати та відображати усі повідомлення з сервера та клієнта.

Компонент використовує селектор errorMsg для постійного оновлення та показу помилок.

```
function Error() {
    const authError = useSelector(errorMsg)
    const dispatch = useDispatch()
    useEffect(() => {
        if(authError)
            toast.error(authError, notifyErrorOptions)
        else
            dispatch(resetError())
    }, [authError, dispatch])
    return (
        <div>
            <ToastContainer newestOnTop rtl={false} pauseOnFocusLoss={false} />
        </div>
    )
}
```

Рисунок 3.21 – Компонент виводу помилки

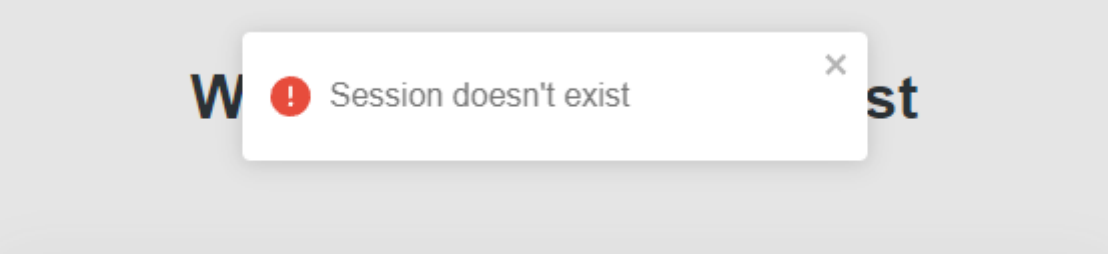


Рисунок 3.22 – Приклад помилки з використанням toastify

Далі створюємо всі інші компоненти і отримуємо таку структуру файлів:

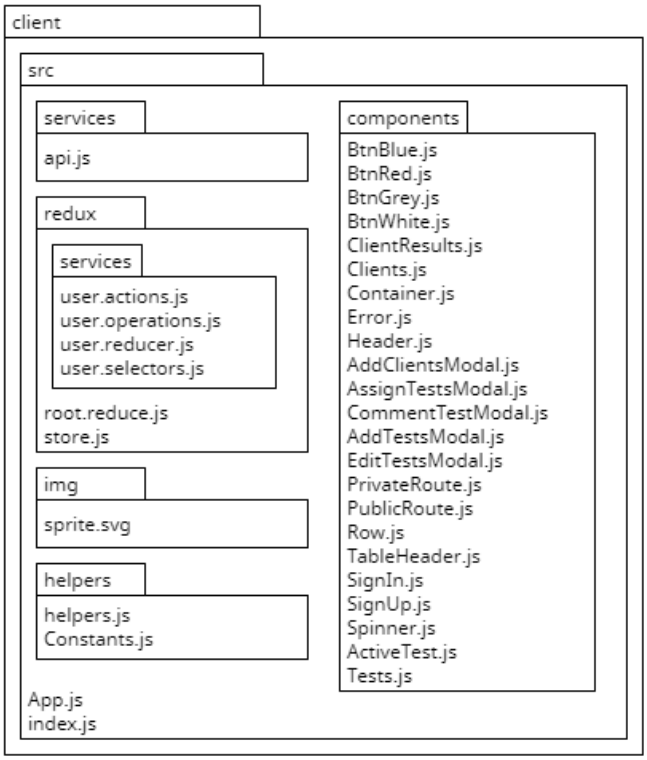


Рисунок 3.23 – Структура файлів компонентів

3.4 Розгортання серверу

Для стабільної та автономної роботи сервера будемо використовувати Amazon сервіс Elastic Beanstalk, який пропонує просту структуру налаштування, деплою та перегляду логів сервера.

AWS Elastic Beanstalk — це служба оркестрування, яку пропонує Amazon Web Services для розгортання програм, які оркеструють різні служби AWS, зокрема EC2, S3, Simple Notification Service (SNS), CloudWatch, автоматичне масштабування та Elastic Load Balancers. Elastic Beanstalk забезпечує додатковий рівень абстракції над відкритим сервером та ОС; замість цього користувачі бачать попередньо створену комбінацію ОС і платформи, наприклад «64-розрядна версія Amazon Linux 2014.03 v1.1.0 під керуванням Ruby 2.0 (Puma)» або «64-розрядна версія Debian jessie v2.0.7 під керуванням Python 3.4 (попередньо налаштована – Docker)». Для розгортання потрібно визначити ряд компонентів: «додаток» як логічний контейнер для проекту,

«версія», яка є розгорнутою збіркою програми, що виконується, «шаблон конфігурації», який містить інформацію про конфігурацію для середовища Beanstalk. і для продукту.

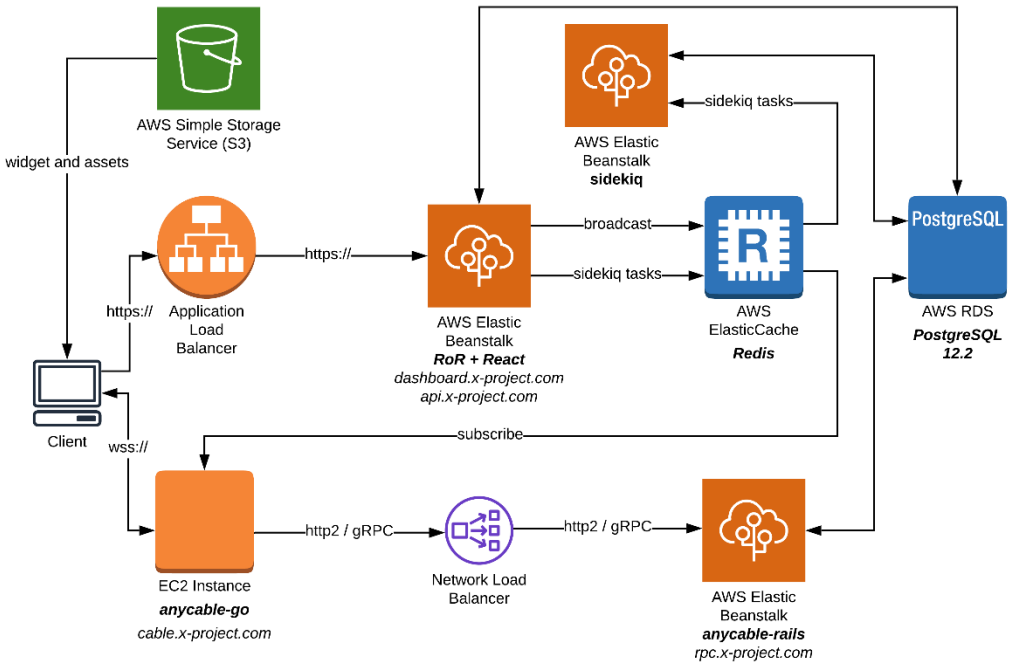


Рисунок 3.24 – Приклад зв’язку між компонентами AWS та Elastic Beanstalk

Спочатку створюємо новий application на AWS та називаємо його Psychologist, обираємо тип Node.JS та обираємо t3.micro инстанс, який входить у Free Tier для безкоштовного користування.

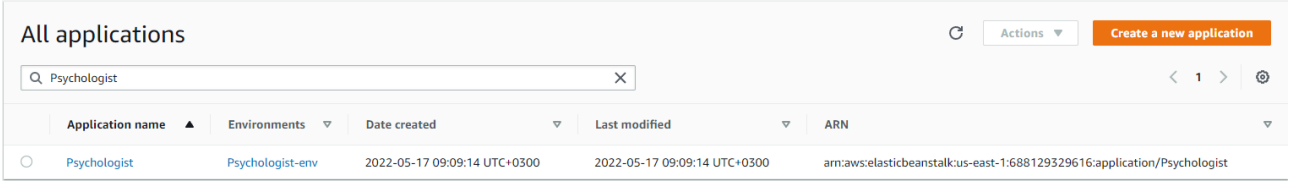


Рисунок 3.25 – Розгорнутий Elastic Beanstalk

У package.json нашого сервера додаэмо команду "deploy": "npm ci && npm run start", яка перевстановлює пакети та запускає сервер. Також додаємо Procfile у корінь проекту з однією командою web: npm run deploy.

Архівуємо проект та завантажуємо його у Elastic Beanstalk:

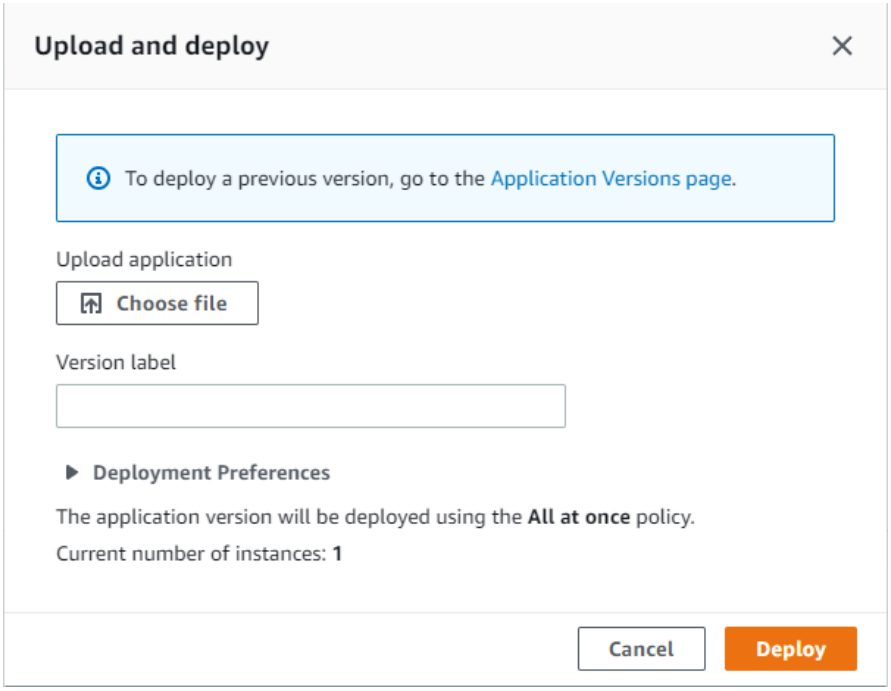


Рисунок 3.26 – Деплой додатку

Для отримання SSL сертифікату потрібно створити домен та направити його на Elastic Beanstalk сервер. Для цього у DNS налаштуваннях треба створити CNAME record зі значення нашого проекту. Замовляємо SSL сертифікат на AWS та теж додаємо CNAME record для нього.

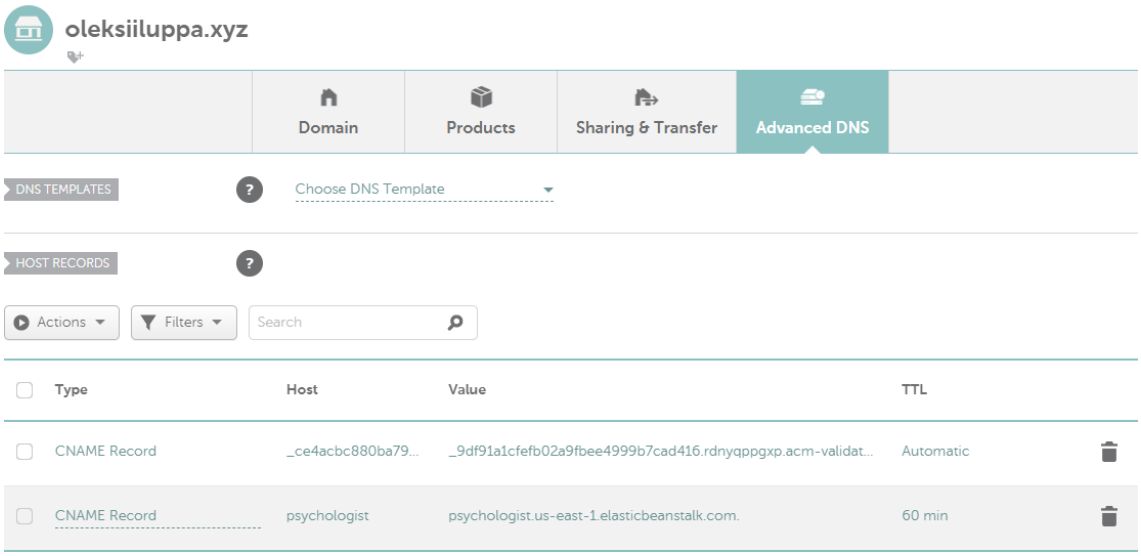


Рисунок 3.27 – Створення CNAME records

Ще потрібно створити автоматичну переадресацію з http на https. Це можна зробити в Load Balancer на AWS. Для цього створюємо два лісенери: один на порт 80 з переадресацією на порт 443, другий з портом 443, замовленим сертифікатом та прями доступом.

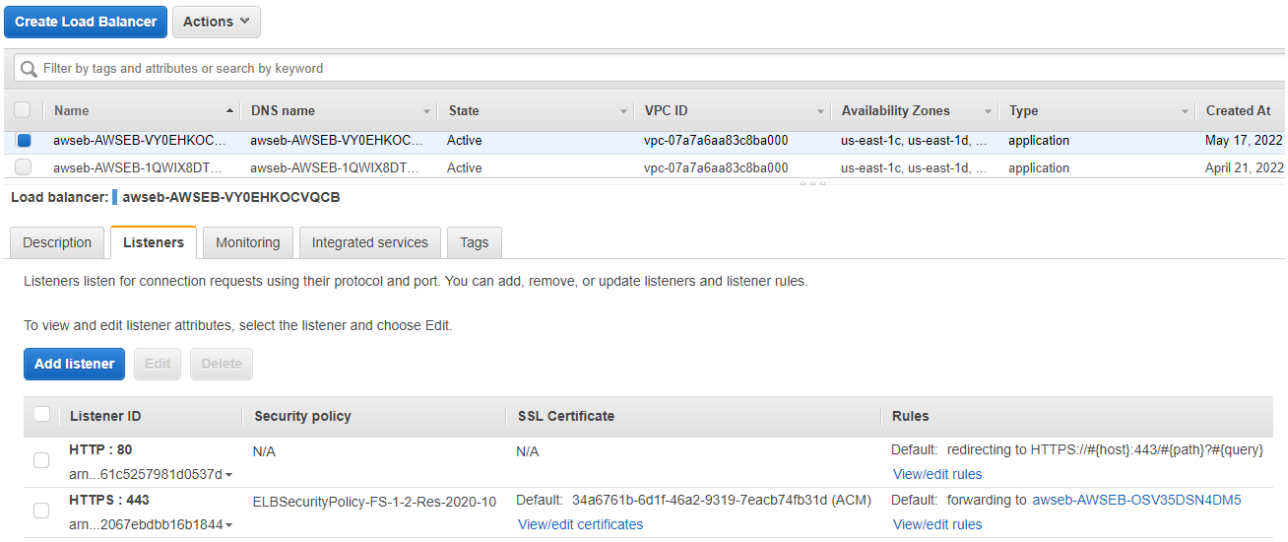


Рисунок 3.28 – Створення лісенерів в Load Balancer

3.5 Відтворення дій користувача

Переходимо за посиланням psychologist.oleksiiluppa.xyz

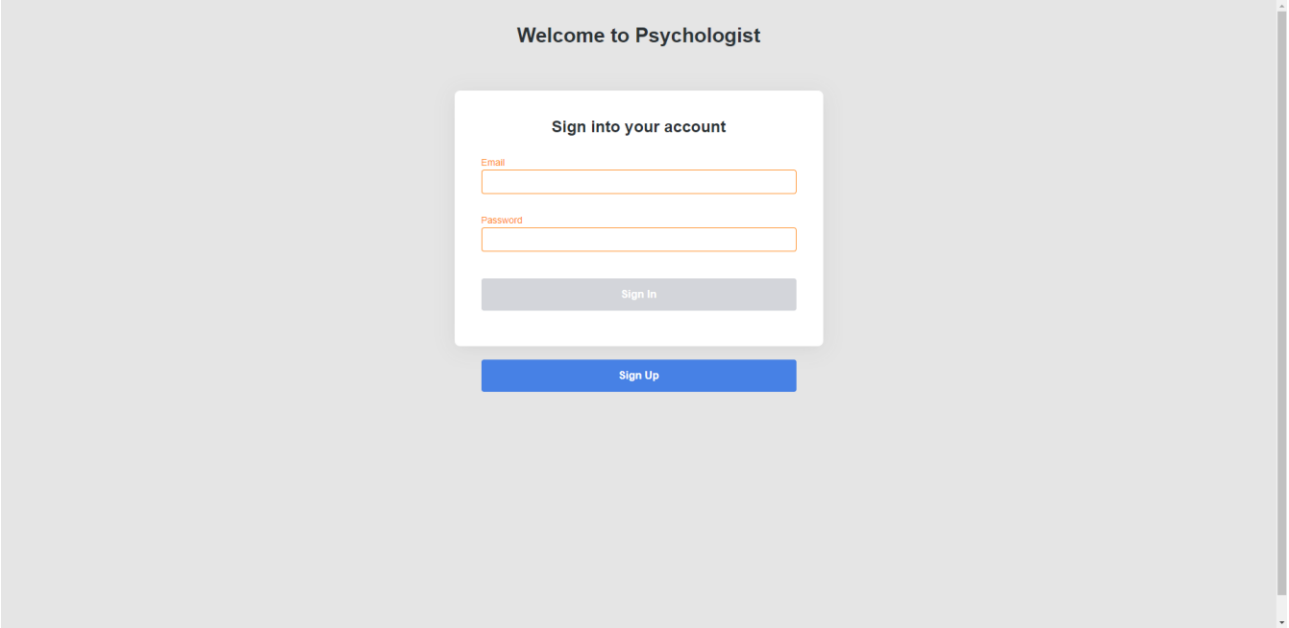


Рисунок 3.29 – Сторінка авторизації

Можемо авторизуватись, або перейти на сторінку реєстрації. Зареєструємо новий акаунт психолога:

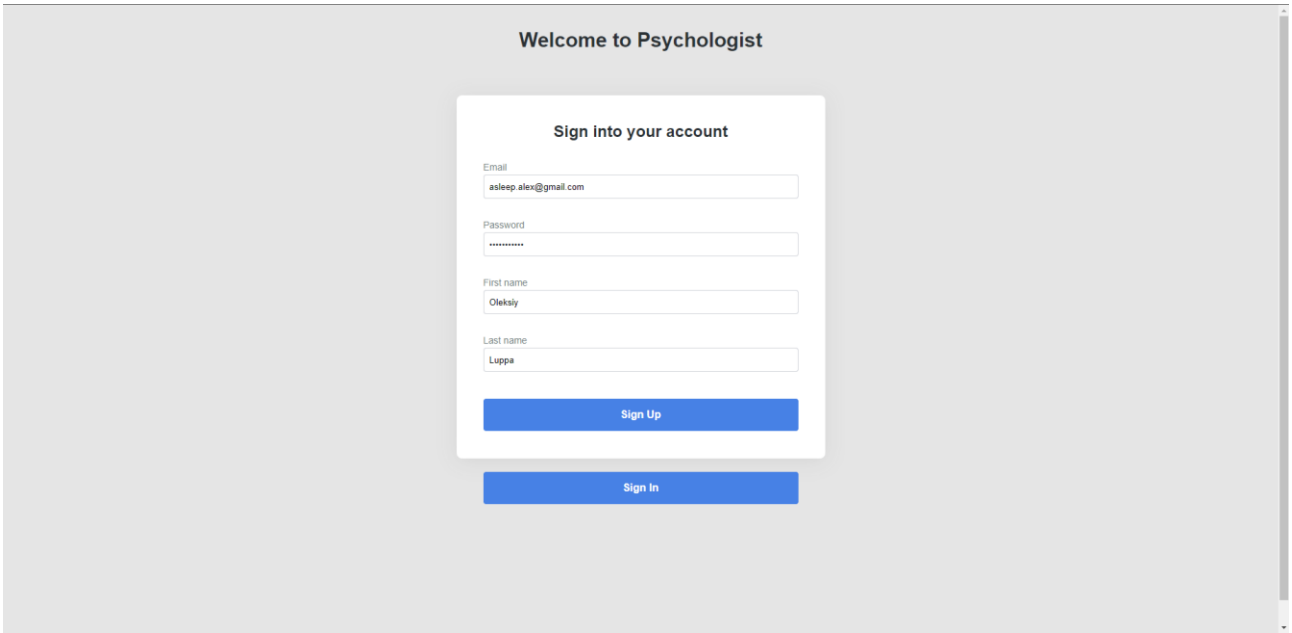


Рисунок 3.30 – Сторінка реєстрації

Створимо нового клієнта:

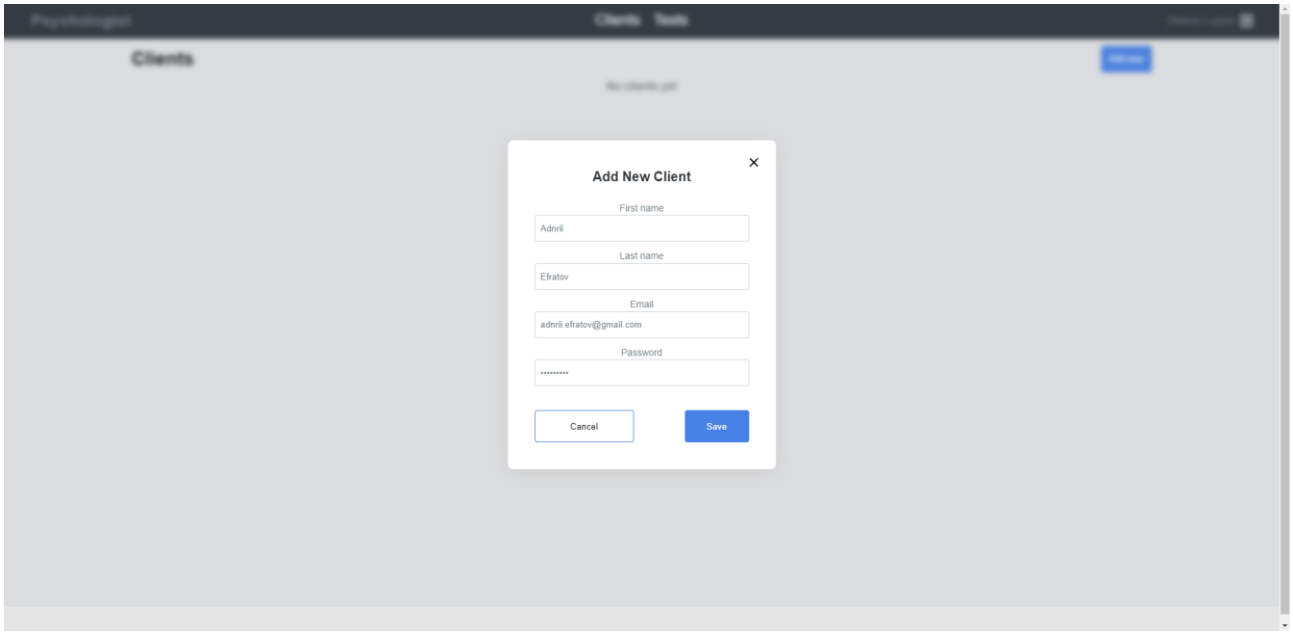


Рисунок 3.31 – Створення нового клієнта

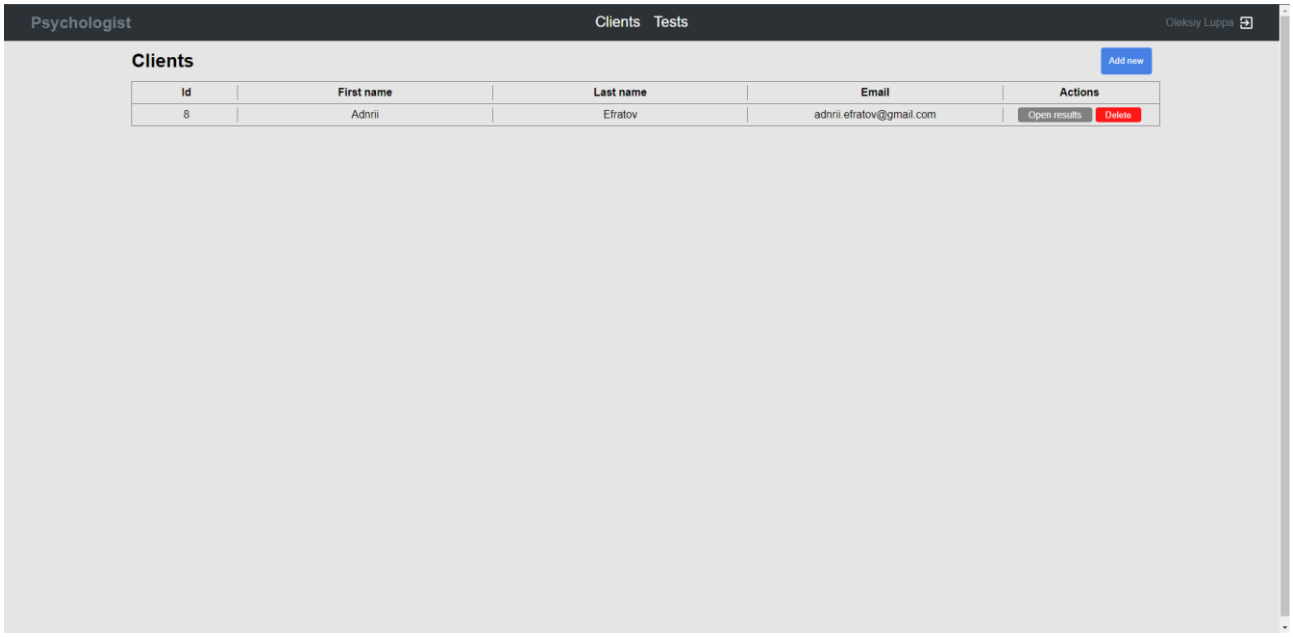


Рисунок 3.32 – Таблиця клієнтів

Створимо новий тест для проходження, введемо назву та декілька питань на які дасть відповідь клієнт.

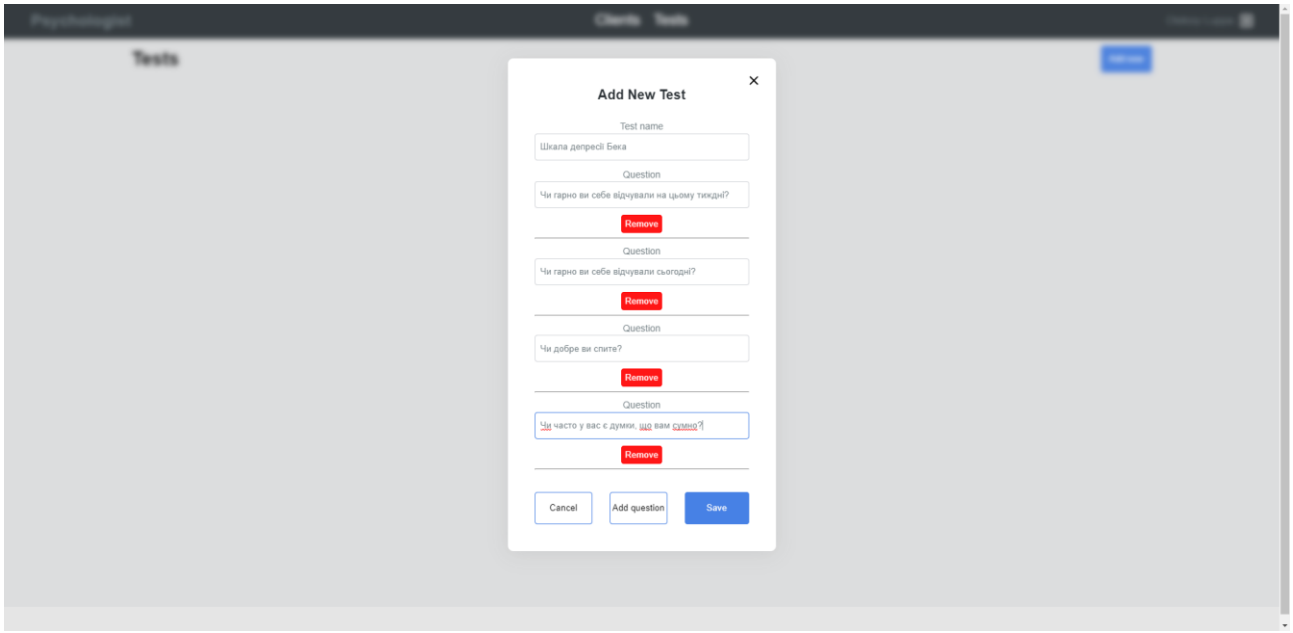


Рисунок 3.33 – Створення тесту

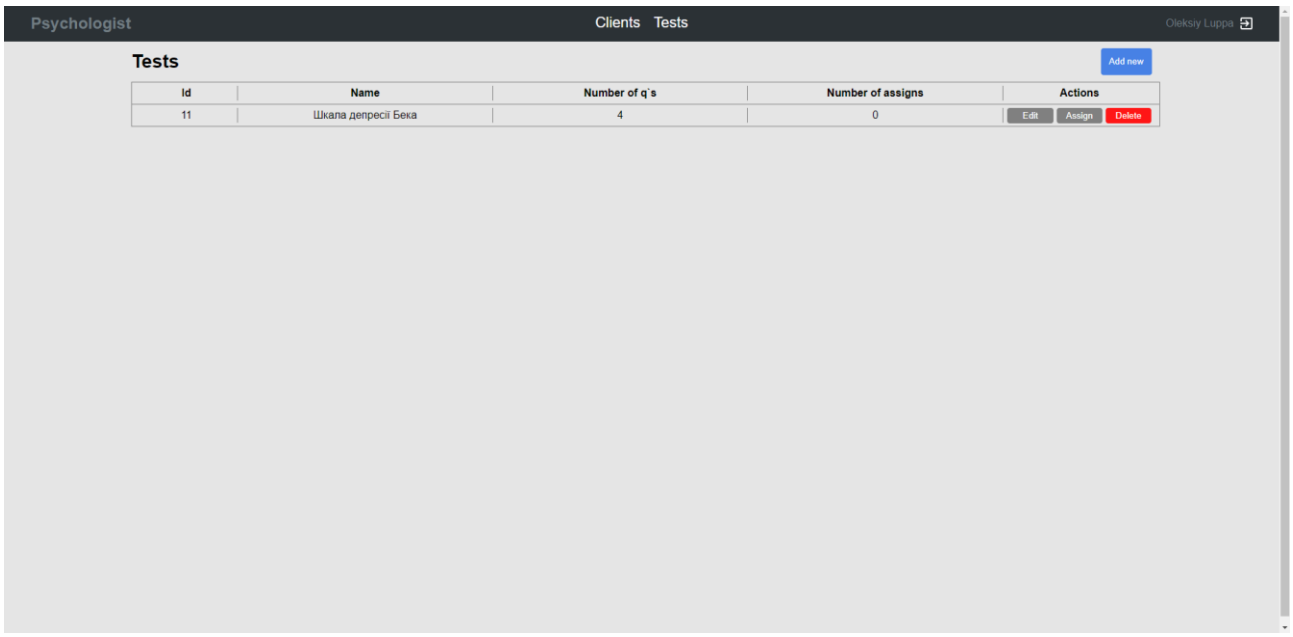


Рисунок 3.34 – Таблиця тестів

Присвоїмо тест нашому пацієнту, щоб він міг його пройти.

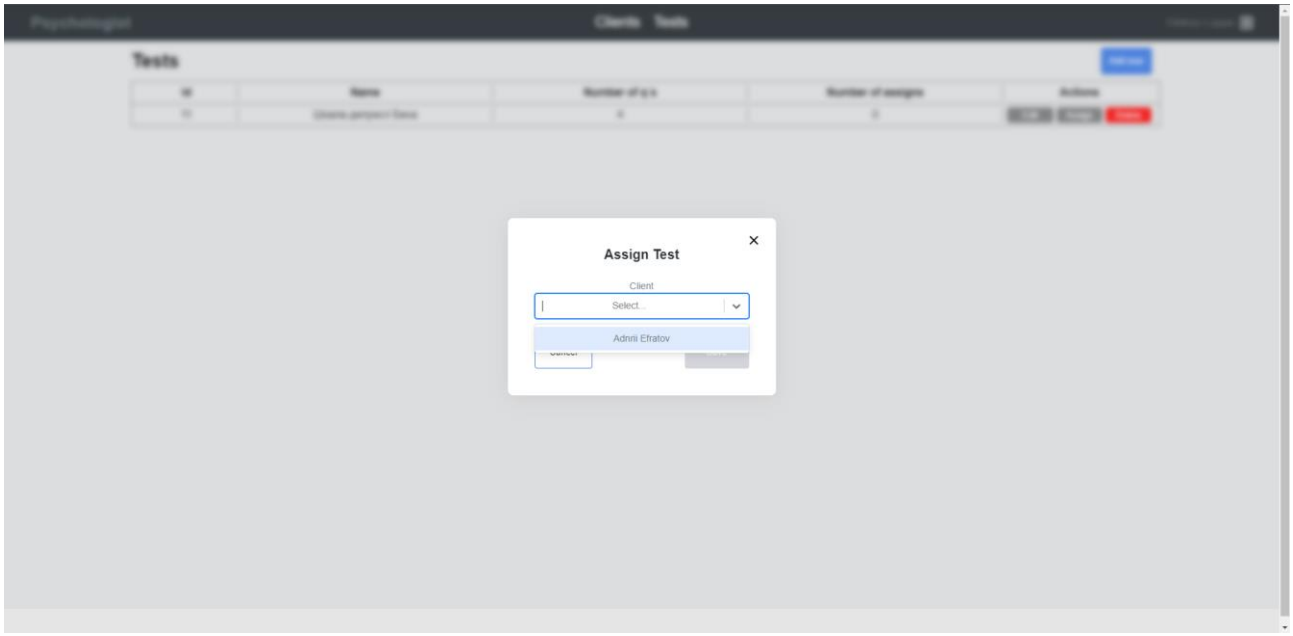


Рисунок 3.35 – Присвоєння тесту клієнту

Авторизуємось за клієнта та пройдемо новий тест.

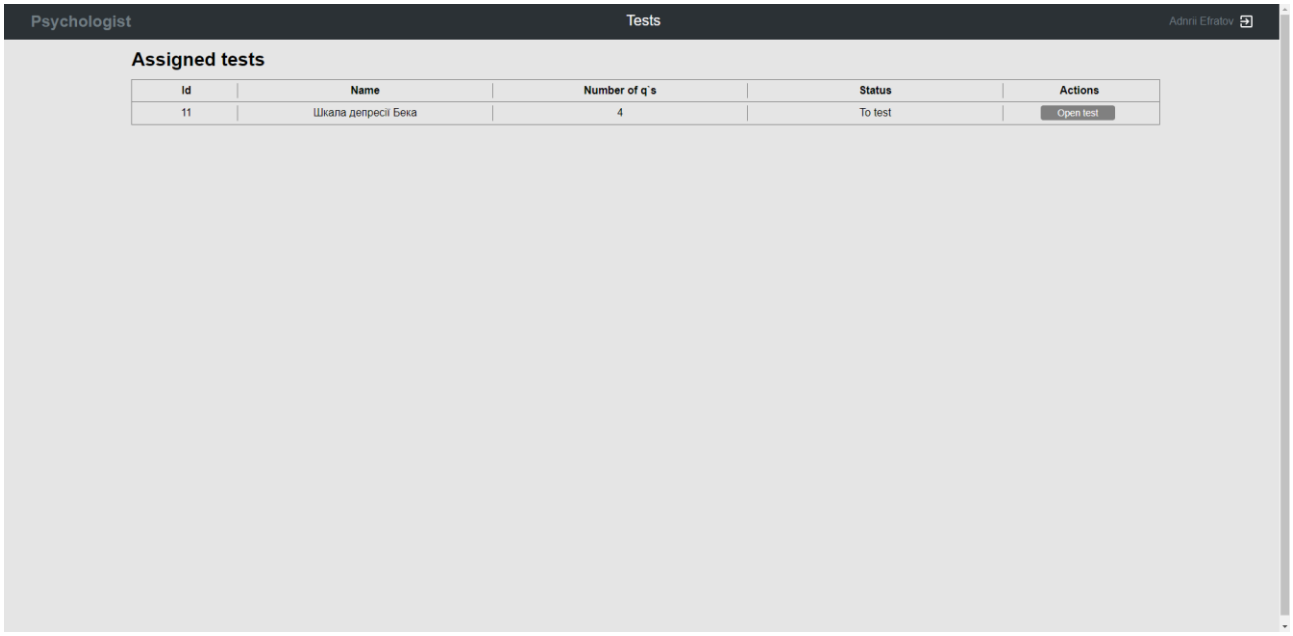


Рисунок 3.36 – Таблиця тестів клієнта

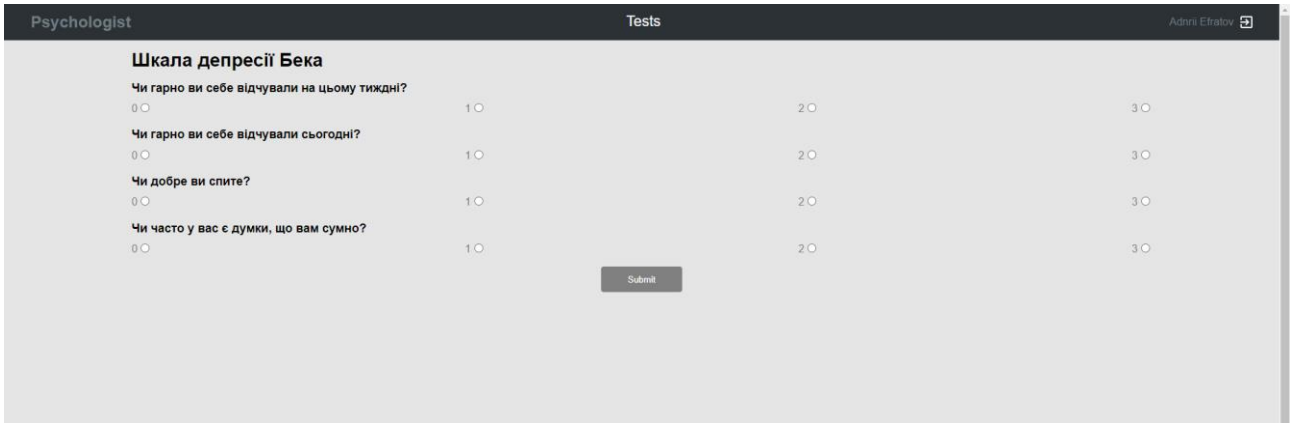


Рисунок 3.37 – Проходження тесту

Тест змінив свій статус, психолог тепер може перевірити його та залишити розгорнутий коментар.

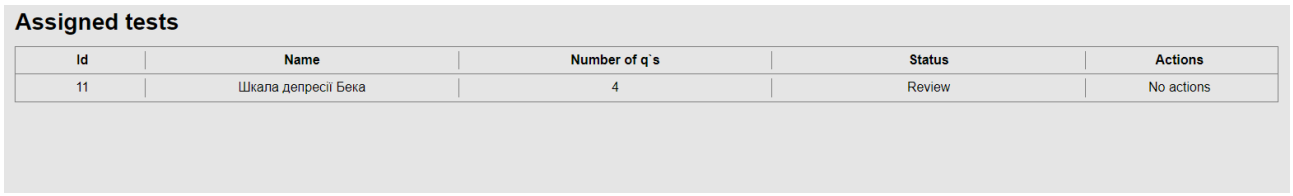


Рисунок 3.38 – Статус тесту змінився

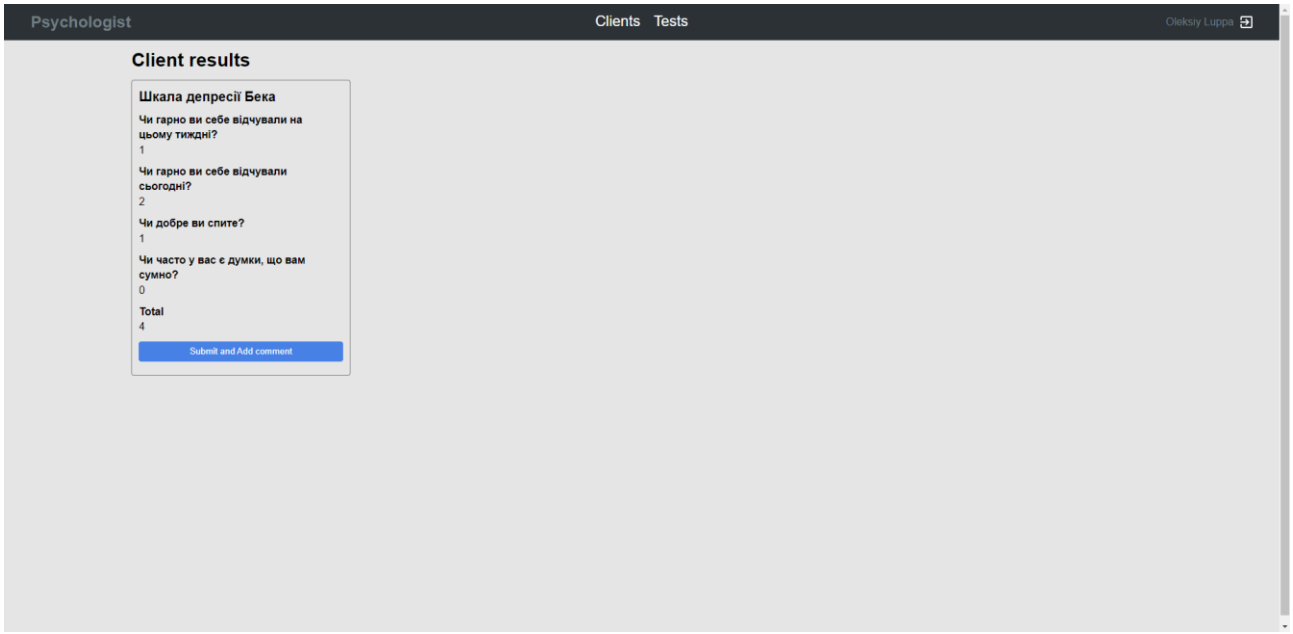


Рисунок 3.39 – Результат тесту

Рисунок 3.40 – Коментар психолога

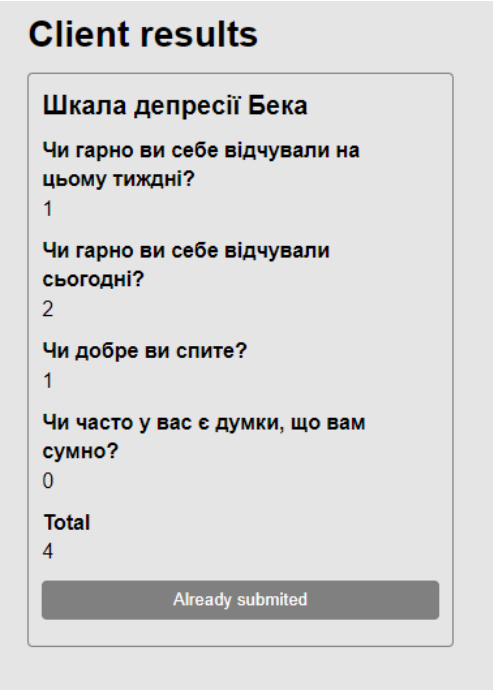


Рисунок 3.41 – Статус перевірки тесту

Тепер клієнт зможе побачити коментар психолога на зробити певні дії для поліпшення свого стану.

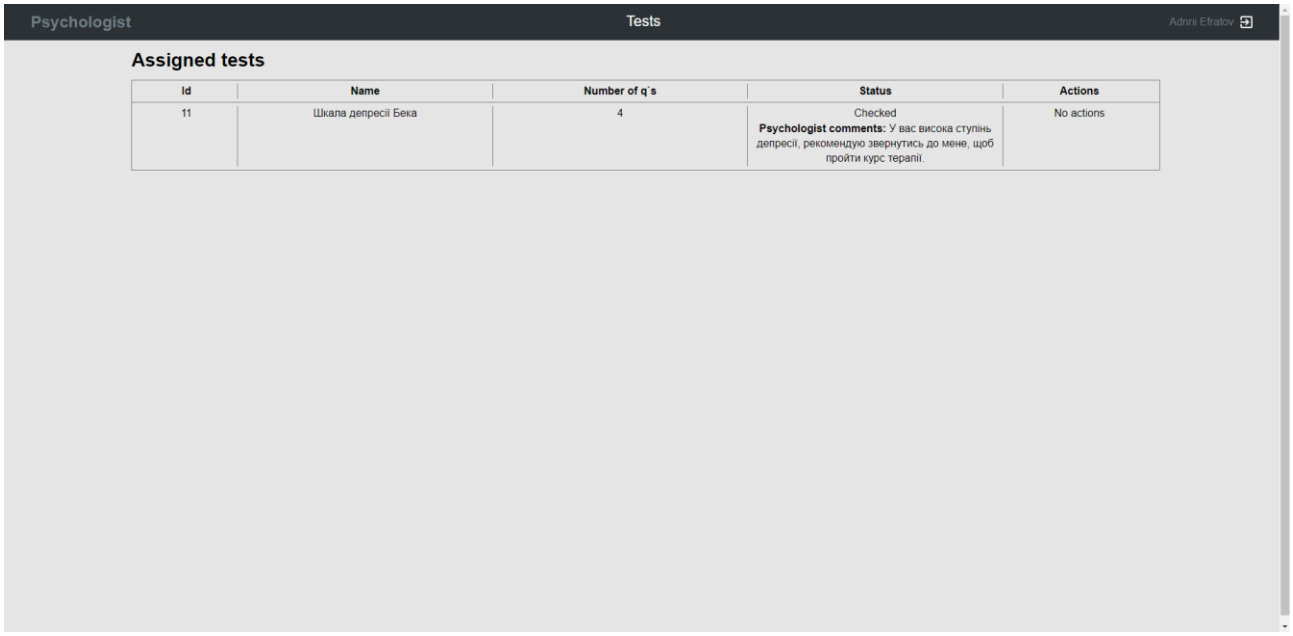


Рисунок 3.42 – Коментар психолога у клієнта

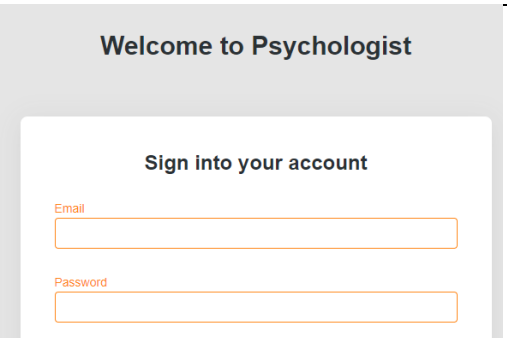
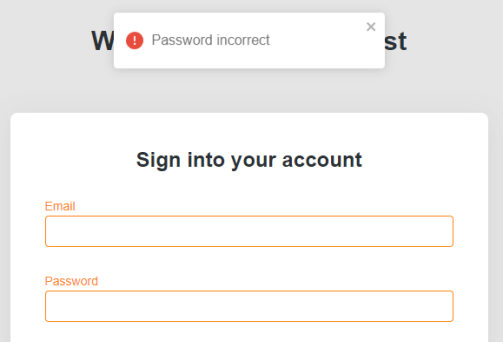
3.6 Тестування

Тестування додатку буде виконано у манері Manual testing. Ручне тестування (Manual testing) — це процес ручної перевірки програмного забезпечення на помилки.

Тестувальник має відігравати роль користувача програми й використовувати властивості програми для знаходження помилок у роботі програми. Для професійного тестування тестувальник часто користується написаним планом тестування з варіантами тестування (test cases).

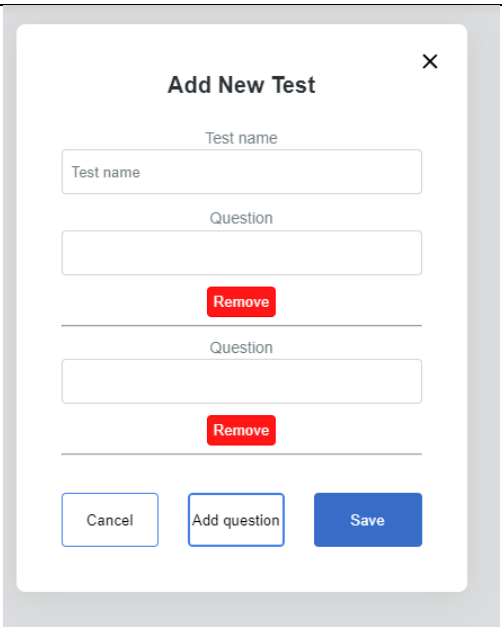
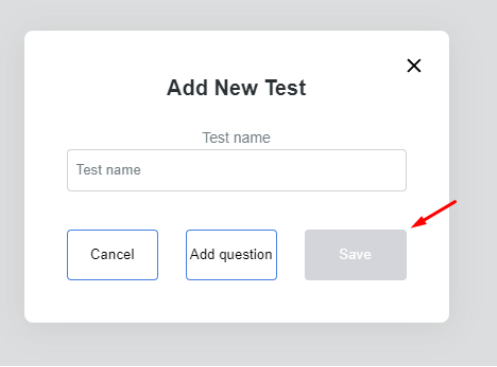
Таблиця 3.1 – Test case 1

Компонент системи (розділ)	Авторизація
Категорія (Category) (тип багу)	Повідомлення
Оточення	
операційна система	**Усе**
браузер	**Будь-який**

Передумови *(необов'язково)*	
Відтворення (Reproducibility)	Виникає у всіх завжди
Короткий опис (Summary)	
Що? (опис помилки)	Нема повідомлення про помилку авторизації
Де?(в якому місці спостерігається дефект/баг)	При авторизації
Коли?(за яких умов відбувається помилка)	При другому натисканні на Sign in при невірному паролі
Важливість:	S5 Тривіальна
Пріоритет:	P3 Низький
Кроки відтворення	1. Перейти на головну сторінку сайту 2. Ввести помилкові дані 3. Натиснути Sign in 4. Ввести помилкові дані ще раз 5. Натиснути Sign in
Фактичний результат (обов'язково додати скрин екрану та позначити на ньому місце бага червоною стрілкою)	
Очікуваний результат	

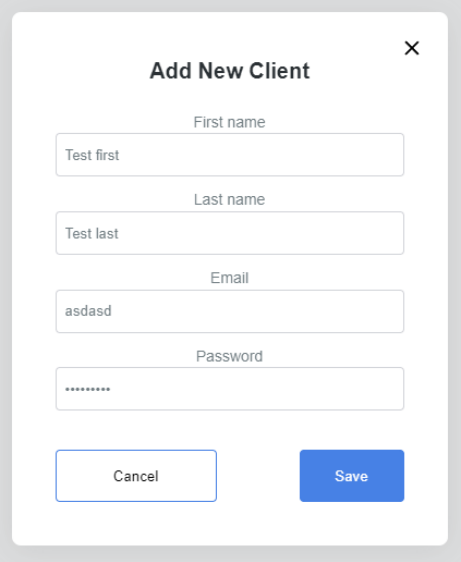
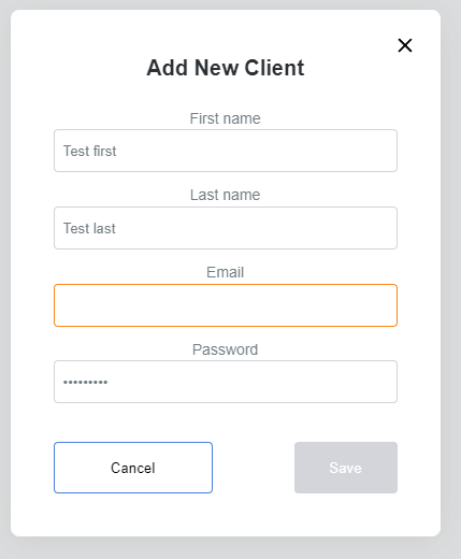
Таблиця 3.2 – Test case 2

Компонент системи (розділ)	Тести
Категорія (Category) (тип багу)	Валідація
Оточення	
операційна система	**Усе**
браузер	**Будь-який**
Передумови *(необов'язково)*	
Відтворення (Reproducibility)	Виникає у всіх завжди
Короткий опис (Summary)	
Що? (опис помилки)	Нема валідації на пусті питання до тесту
Де?(в якому місці спостерігається дефект/баг)	У модалці створення тесту
Коли?(за яких умов відбувається помилка)	Завжди
Важливість:	S3 Значна
Пріоритет:	P2 Середній
Кроки відтворення	1. Перейти на сторінку тестів психолога 2. Настиснути Add new 3. Ввести назву 4. Натиснути Add question

Фактичний результат (обов'язково додати скрин екрану та позначити на ньому місце бага червоною стрілкою)	
Очікуваний результат	

Таблиця 3.3 – Test case 3

Компонент системи (розділ)	Клієнти
Категорія (Category) (тип багу)	Валідація
Оточення	
операційна система	**Усе**
браузер	**Будь-який**
Передумови *(необов'язково)*	
Відтворення (Reproducibility)	Виникає у всіх завжди
Короткий опис (Summary)	

Що? (опис помилки)	Нема валідації інпуті пошти
Де?(в якому місці спостерігається дефект/баг)	У модалці створення клієнта
Коли?(за яких умов відбувається помилка)	Завжди
Важливість:	S3 Значна
Пріоритет:	P2 Середній
Кроки відтворення	1. Перейти на сторінку клієнтів психолога 2. Настиснути Add new 3. Ввести невалідну пошту
Фактичний результат (обов'язково додати скрин екрану та позначити на ньому місце бага червоною стрілкою)	
Очікуваний результат	

3.7 Висновки до глави 3

За попередньо запроектованою логічною моделлю була розгорнута та створена база даних MySQL на cloud сервісі AWS RDS. Була розроблена бекенд частина додатку, запроектована модель станів навігації по клієнтській частині та розроблений React.JS додаток. Серверна та клієнтська частини були розгорнуті на іншому cloud сервісі AWS Elastic Beanstalk. До оточення був доданий домен та SSL сертифікат, за допомогою створення CNAME records в DNS налаштуваннях, для безпечного з’єднання з браузером та сервером.

Був відтворений та протестований весь флоу користувача додатку з обох ролей. За допомогою manual testing було знайдено декілька багів, які були у подальшому виправленні за пріоритетністю.

ВИСНОВКИ

За результатами дипломної роботи запроектовано та розроблено систему психологічного тестування студентів у форматі Web-додатку для використання психологом та його клієнтами.

Проведення психологічного тестування дозволяє оцінити ситуацію в колективі і, як наслідок, сформувати сприятливий психологічний клімат у групі та сприяти зростанню інтересу учнів до навчання.

Основні результати проведеної роботи:

1. зроблено огляд на тему психологічного тестування та програмних засобів психологічного тестування;
2. запроектовано базу даних та змодельовано більшість діаграм для повноцінного огляду додатку;
3. розроблено бекенд Node.JS сервер та фронтенд React.JS додаток, які були розгорнуті разом з БД MySQL на AWS;
4. підключено SSL сертифікат та домен для безпечного з’єднання;
5. проведено manual тестування усього додатку.

При проектуванні додатку було змодельовано діаграми, що описують аспекти роботи з різних сторін:

- структура файлів клієнта;
- діаграма компонентів клієнта;
- діаграма діяльності навігації;
- діаграма компонентів сервера;
- діаграма структури модулів сервера;
- модель діяльності;
- модель пакетів;
- модель розгортання;
- модель предметної галузі;
- моделювання процесу тестування;

- модель прецедентів;
- логічна схема бази даних;
- модель предметної області;
- діаграма функціональних можливостей користувачів.

При розробці використовувалися такі мови програмування, як JavaScript, SQL та мови розмітки HTML і CSS.

Для серверної частини був використаний фреймворк Node.JS з підключенням таких бібліотек:

- bcrypt для хешування паролю;
- cookie-session для збереження сесії користувача;
- jsonwebtoken-refresh для створення JWT токена сесії;
- mysql для підключення до бази даних;
- та інші.

Для клієнтської частини використовувався фреймворк React.JS з бібліотеками:

- react-redux для зберігання даних та станів додатку;
- reduxjs/toolkit для покращення роботи бібліотеки react-redux;
- react-toastify для виводу повідомлень помилок;
- та інші.

На сервері був створений middleware прошарок для контролю сесії та оновлення JWT токена актуальними даними. Також, через високе споживання ресурсів при запуску клієнту та серверу одночасно на cloud сервісах, був створений build клієнтської частини, який сервер хостить як статичні файли.

За допомогою мови SQL були написані усі запити до бази даних для формування масивів даних для клієнта і для запису та оновлення інформації користувачів та тестів. Для де-яких запитів були використані inner join та case вирази.

При розгортанні сервера та БД було використано cloud сервіси AWS. База даних була розгорнута на RDS. Сервер був розгорнутий за допомогою Elastic

Beanstalk та CodePipeline на EC2 екземплярі з подальшим підключенням домену від NameCheap з SSL сертифікатом.

Тестування проводилось за типом manual testing. Було виявлено кілька багів, один з яких був класифікований як значний. У подальшому ці баги були виправлені.

Ця платформа була розроблена для роботи психолога у школах та університетах, з метою покращення та автоматизації проведення тестів психологами у навчальних закладах або для особистого використання.

СПИСОК ЛІТЕРАТУРИ

1. Офіційна документація React // Електронна версія на сайті https://reactjs.org/

2. Онлайн-підручник з JavaScript // Електронна версія на сайті https://learn.javascript.ru/

3. Документація з Node.js // Електронна версія на сайті http://www.nodebeginner.ru/

4. Документація з MongoDB // Електронна версія на сайті https://www.mongodb.com/

5. Документація з React // Електронна версія на сайті http://krambertech.github.io/spa-webinar/

6. Документація з React // Електронна версія на сайті https://habr.com/ua/company/ruvds/blog/432636/

7. Історія веб дизайну, 2017. — RedKrab — URL: https://webevolution/blog/sajti/istoriya-veb-dizajna/

8. Веб 2.0, 2021. — Wikipedia — URL: https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1_2.0

9. Google Class, 2021. — Wikipedia — URL: https://ua.wikipedia.org/wiki/Google_%D0%9A%D0%BB%D0%B0%D1%81%D1%81

10. Microsoft Teams, 2021. — Wikipedia — URL: https://ua.wikipedia.org/wiki/Microsoft_Teams

11. WEB-дизайн як навчальна дисципліна, 2021. — KURSOVIKS — URL: https://ua.kursoviks.com.ua/kompyuterni/web-dizayn

12. Професійна підготовка студентів соціально-педагогічної сфери – освітня складова суспільного розвитку [Електронний ресурс]: матеріали II Міжнародної науково-практичної конференції (дистанційної) (Київ, 1 квітня 2014 року) / Національний педагогічний університет імені М. П. Драгоманова,

Бердянський державний педагогічний університет, Хмельницька гуманітарно-педагогічна академія, Українсько-американський гуманітарний інститут «Вісконсинський міжнародний університет (США) в Україні»; за заг. ред. І. М. Ковчиної. - Київ: [б. в.], 2014. - 120 с.

13. Основи веб-дизайну [Електронний ресурс]: навчальний посібник / О. Г. Пасічник, О. В. Пасічник, И. В. Стеценко. - К.: Видавнича група BHV, 2009. - 336 с.

14. Педагогіка вищої школи [Електронний ресурс]: навч. посібник / за ред. З. Н. Курлянд. - К.: Знання, 2005. - 399 с.

15. Теорія і методи соціальної роботи [Електронний ресурс]: навчальний посібник / М. П. Лукашевич, І. І. Мигович; Міжрегіональна академія управління персоналом. - 2-ге вид., допов. і випр.. - К.: [б. в.], 2003. - 168 с.

16. Педагогіка у запитаннях і відповідях [Електронний ресурс] : навч. посіб. / А. І. Кузьмінський, В. Л. Омеляненко. - К.: Знання, 2006. - 311 с.. - (Навчально-методичний комплекс з педагогіки)

17. Інформаційно-комунікаційні технології у післядипломній освіті [Електронний ресурс] / В. С. Назаренко, В. В. Кузьменко. - Херсон: Херсонська академія неперервної освіти, 2013. - 171 с.

18. A history of the university in Europe [Electronic resource]: in 4 vol. / ed. Walter Ruegg. - Cambridge: Cambridge university press, 2003

19. Проблеми та перспективи розвитку освіти, науки і техніки в Україні та світі [Електронний ресурс]: зб. праць за матеріалами Всеукраїнської науково-практичної конференції, 20-21 травня 2016 р. / Інститут історії України НАН України; уклад. С. М. Ховрич. - Київ : [б. в.], 2016. - 139 с.

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для тестування пацієнтів у психолога з використанням технологій Node.js та React

Виконав студент(ка) 4 курсу
групи ПД-41
Луппа О.А.
Керівник роботи
Гаманюк І.М.

Київ – 2022

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Метою роботи** є пошук оптимального способу створення інформаційної системи «тестування у психолога», що має можливість тестувати пацієнтів, підказки для клієнтів підчас тестування та інтерфейс користувача WEB-додатка.
- **Об'єктом дослідження** є процес автоматизації роботи психолога.
- **Предметом дослідження** є розробка програмного забезпечення для тестування пацієнтів у психолога.

2

АНАЛОГИ

Веб-сайт psytests.org

Веб-сайт psytests.org містить велику колекцію психологічних тестів. Тестування безкоштовне і не вимагає реєстрації.

Система тестування INDIGO

Система тестування INDIGO – це професійний інструмент для автоматизації процесу тестування та обробки результатів, призначений для вирішення широкого кола завдань.

Тестування системи StartExam

Система тестування співробітників StartExam є хмарним рішенням. Доступ як до особистого кабінету адміністратора, так і до сторінки проходження тесту здійснюється через веб-інтерфейс. Метою сервісу є проведення зрізів знань серед співробітників.

3

ТЕХНІЧНЕ ЗАВДАННЯ

- Розробити інформаційну систему для можливого тестування пацієнтів у психолога у вигляді веб-сайту.
- Використання HTML, CSS, JS(Node.js та React) для створення веб-сайту, використання MySQL та PHP для роботи з базою даних, розробка форми та структури форми для проектування системи тестування для отримання замовлень на продукцію через web-представництво.
- Сайт має бути адаптивним і крос-браузерним.
- Необхідно передбачити перевірку правильності введених даних.
- Додайте на сайт допоміжні сторінки, створивши таким чином структуру (наприклад, сторінку привітання, сторінку з формою реєстрації, сторінку, яка відкривається після успішного заповнення форми, сторінку контактів) і зв'яжіть ці сторінки одна з одною за допомогою посилань або кнопки.

4

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

- Мова програмування JavaScript (JS) —надає веб-сторінкам можливість реагувати на дії користувача та перетворювати статичні сторінки в динамічні.
- Node.js - це програмна платформа, яка переводить JavaScript в машинний код і перетворює JavaScript з вузькоспеціалізованої мови на мову загального призначення з власним середовищем розробки.
- React розроблений Facebook і показує свою високу ефективність усередині динамічних програм з високим трафіком.
- MySQL — вільна система керування реляційними базами даних, яка була розроблена компанією «TcX» для підвищення швидкодії обробки великих баз даних.

5

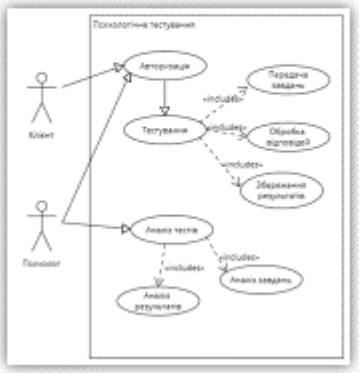
МЕТОДИ ТА КЛАСИ ПРОГРАМИ

Функціональні можливості користувачів

6

МЕТОДИ ТА КЛАСИ ПРОГРАМИ

Модель прецедентів

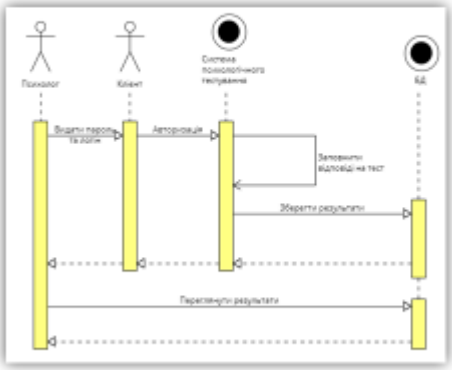


Модель станів

7

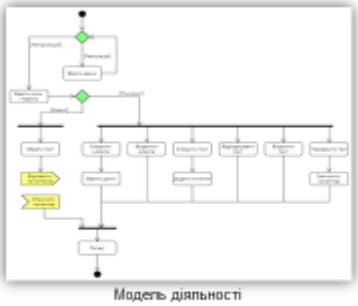
ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ БД

Моделювання процесу тестування



8

МЕТОДИ ТА КЛАСИ ПРОГРАМИ



Модель діяльності

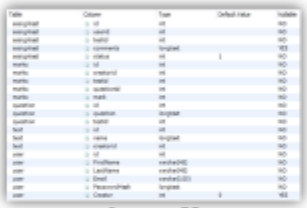
Модель розгортання

Архітектура системи

9

ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ БД

Логічна схема бази даних



Структура БД

Модель предметної галузі

10

МЕТОДИ ТА КЛАСИ ПРОГРАМИ

Реєстрація

Додавання тесту

Проходження тесту

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

- Луппа О.А. Актуальність та мета розробки ПЗ для психологів та пацієнтів / Застосування програмного забезпечення в інфокомунікаційних технологіях : матеріали наук.-тех. конф., м. Київ, 20 квітня 202 р. / Державний університет телекомунікацій, кафедра інженерії програмного забезпечення, Київ, 2022
- Луппа О.А. Використання UML діаграми прецедентів для моделювання основних функцій системи з тестування пацієнтів / Застосування програмного забезпечення в інфокомунікаційних технологіях : матеріали наук.-тех. конф., м. Київ, 20 квітня 202 р. / Державний університет телекомунікацій, кафедра інженерії програмного забезпечення, Київ, 2022

12

ВИСНОВКИ

За результатами дипломної роботи запроектовано та розроблено систему психологічного тестування студентів у форматі Web-додатку для використання психологом та його клієнтами.

Основні результати проведеної роботи:

1. зроблено огляд на тему психологічного тестування та програмних засобів психологічного тестування;
2. запроектовано базу даних та змодельовано діаграми: структура файлів клієнта, діаграма компонентів клієнта, діаграма діяльності навігації, діаграма компонентів сервера, діаграма структури модулів сервера, модель діяльності, модель пакетів, модель розгортання, модель предметної галузі, моделювання процесу тестування, модель прецедентів, логічна схема бази даних, модель предметної області, діаграма функціональних можливостей користувачів;
3. розроблено бекенд Node.JS сервер та фронтенд React.JS додаток, які були розгорнуті разом з БД MySQL на AWS;
4. підключено SSL сертифікат та домен для безпечного з'єднання;
5. проведено manual тестування усього додатку.

13

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК В

Сервер

app.js

```
const express = require("express");
const cors = require("cors");
const path = require("path");
const app = express();
const server = require("http").createServer(app);
const cookieSession = require("cookie-session");
const auth = require("./routes/auth");
const clients = require("./routes/clients");
const tests = require("./routes/tests");
const logger = require('logger-line-number')

const port = process.env.PORT || 4000;
```

```
app.use(
  cookieSession({
    name: "session",
    keys: ["token"],
  })
);

app.use(express.json());

// for deploy
app.use(cors({ credentials: true, origin: "*" }));

// app.use(cors({ credentials: true, origin: "http://localhost:3000" }));

app.use("/api/auth", auth)
app.use("/api/clients", clients)
app.use("/api/tests", tests)

app.use((err, req, res, next) => {
  const status = err.status || 500;
  res.status(status).json({ message: err.message });
});

// for deploy
app.use("/", express.static(path.join(__dirname, "client", "build")));
app.get("*", (req, res) => {
  res.sendFile(path.resolve(__dirname, "client", "build", "index.html"));
});


server.listen(port, () => {
  logger.log(`Listening on port ${port}`)
});
```

routes/auth.js

```
const { Router } = require('express');
const router = Router();
const ctrl = require('../controllers/auth');
const ctrlHandler = require('../helpers/controllerHandler');

router.post('/signup', ctrlHandler(ctrl.signup));
router.post('/signin', ctrlHandler(ctrl.signin));
router.get('/refreshSession', ctrlHandler(ctrl.checkSession));
router.post('/logout', ctrlHandler(ctrl.logout));

module.exports = router;
```

routes/clients.js

```
const { Router } = require('express');
const router = Router();
const ctrl = require('../controllers/clients');
const ctrlHandler = require('../helpers/controllerHandler');
const guard = require("../helpers/guard");

router.get('/', guard, ctrlHandler(ctrl.getAllClients));
router.post('/', guard, ctrlHandler(ctrl.addClient));
router.post('/delete', guard, ctrlHandler(ctrl.removeClient));

module.exports = router;
```

routes/tests.js

```
const { Router } = require('express');
const router = Router();
const ctrl = require('../controllers/tests');
const ctrlHandler = require('../helpers/controllerHandler');
const guard = require("../helpers/guard");

router.get('/all', guard, ctrlHandler(ctrl.getAllTests));
router.post('/add', guard, ctrlHandler(ctrl.addTest));
router.post('/edit', guard, ctrlHandler(ctrl.editTest));
router.post('/remove', guard, ctrlHandler(ctrl.removeTest));
router.post('/assign', guard, ctrlHandler(ctrl.assignTest));
router.post('/getActive', guard, ctrlHandler(ctrl.getActiveTest));
router.post('/answerActive', guard, ctrlHandler(ctrl.answerActiveTest));
router.post('/clientTests', guard, ctrlHandler(ctrl.clientTests));
router.post('/clientTestComment', guard, ctrlHandler(ctrl.clientTestComment));

module.exports = router;
```

controllers/auth.js

```
const requests = require('./requests');
const PasswordHelper = require('../helpers/password');
const { HttpCode } = require('../helpers/constants');
const jwt = require('jsonwebtoken-refresh');

require('dotenv').config();

const signup = async (req, res) => {
  const { email, password, firstName, lastName } = req.body;

  const user = await requests.getInfoByEmail(email);

  if (user) {
```

```
    return res.status(HttpCode.CONFLICT).json({
      success: false,
      message: 'User with this email already exist',
    });
  }

  const passwordHash = PasswordHelper.hashWithSalt(password,
process.env.SALT_SECRET);

  const addUserResult = await requests.addClient({ email, password: passwordHash,
firstName, lastName });

  return res.status(HttpCode.OK).json({
    message: 'Successfully registered',
    addUserResult
  });
};

const signin = async (req, res) => {
  const { email, password } = req.body;

  const user = await requests.getInfoByEmail(email);

  if (!user) {
    console.log("t")
    return res.status(HttpCode.NOT_FOUND).json({
      success: false,
      message: 'User doesn`t exist',
    });
  }

  const verifyPassword = await PasswordHelper.compare(password, user.PasswordHash);

  if (!verifyPassword) {
    return res.status(HttpCode.BAD_REQUEST).json({
      success: false,
      message: 'Password incorrect',
    });
  }

  const token = jwt.sign({ ...user }, process.env.SALT_SECRET, {
    expiresIn: '7 days',
  });

  req.session.token = token;

  return res.status(HttpCode.OK).json({
    user,
    token,
  });
```

```
};

const checkSession = async (req, res) => {
  const tokenSession = req.session.token;
  if (!tokenSession) {
    return res.status(HttpCode.UNAUTHORIZED).json({
      message: "Session doesn't exist",
    });
  }

  jwt.verify(tokenSession, process.env.JWT_SECRET );
  const originalDecoded = jwt.decode(tokenSession, { complete: true });
  const refreshed = jwt.refresh(originalDecoded, "7d", process.env.JWT_SECRET );
  req.session.token = refreshed;
  console.log(originalDecoded)

  return res.status(HttpCode.OK).json({
    user: originalDecoded.payload,
    token: req.session.token,
  });
};

const logout = async (req, res) => {
  req.session.token = null;

  return res.status(HttpCode.OK).json({
    user: originalDecoded.payload,
    token: req.session.token,
  });
};

module.exports = {
    signup,
    signin,
    logout,
    checkSession
};
```

controllers/clients.js

```
const requests = require('./requests');
const PasswordHelper = require('../helpers/password');
const { HttpCode } = require('../helpers/constants');
const jwt = require('jsonwebtoken-refresh');

require('dotenv').config();

const getAllClients = async (req, res) => {
```

```
    const clients = await requests.getClients(req.user.id)

    return res.status(HttpCode.OK).json({
        clients
    });
};

const addClient = async (req, res) => {
    const { email, password, firstName, lastName } = req.body;

    const user = await requests.getInfoByEmail(email);

    if (user) {
        return res.status(HttpCode.CONFLICT).json({
        success: false,
        message: 'User with this email already exist',
        });
    }

    const passwordHash = PasswordHelper.hashWithSalt(password,
process.env.SALT_SECRET);

    const addUserResult = await requests.addClient({ email, password: passwordHash,
firstName, lastName, creator: req.user.id });

    return res.status(HttpCode.OK).json({
        message: 'Successfully added',
        addUserResult
    });
};

const removeClient = async (req, res) => {
    await requests.removeClient(req.user.id, req.body.id)

    return res.status(HttpCode.OK).json({
        message: 'Successfully removed'
    });
};

module.exports = {
    getAllClients,
    addClient,
    removeClient
};
```

controllers/requests.js

```
const DB = require('../helpers/DB');
```

```
const getInfoByEmail = async (email) => {
    const results = await new DB().ExecuteQuery(`select * from user where Email =
'${email}'`);
    return results.length > 0 ? results[0] : false;
};

const getClients = async (userId) => {
    const results = await new DB().ExecuteQuery(`select * from user where
Creator=${userId}`);
    return results.length > 0 ? results : null;
};

const removeClient = async (userId, id) => {
    const results = await new DB().ExecuteQuery(`delete from user where
Creator=${userId} and id=${id}`);
    return results;
};

const addClient = async ({
    email,
    password,
    firstName,
    lastName,
    creator = 0
  }) =>
{
    const query = `insert into user (Email, PasswordHash, FirstName, LastName,
Creator) values ('${email}', '${password}', '${firstName}', '${lastName}',
${creator})`;
    const result = await new DB().ExecuteQuery(query);
    return {
        Id: result?.insertId,
    };
};

const addTest = async ({
    questions,
    name,
    creator
  }) =>
{
    const query = `insert into test (name, creatorid) values ('${name}',
${creator})`;
    const result = await new DB().ExecuteQuery(query);
    console.log(questions)

    for(let q of Object.values(questions)){
        const queryQ = `insert into question (question, testid) values
('${q.question}', ${result?.insertId})`;
```

```
        await new DB().ExecuteQuery(queryQ);
    }
    return {
        Id: result?.insertId,
    };
};

const editTest = async ({
    id,
    questions,
    name,
    creator
  }) =>
{
    const query = `update test set name='${name}' where id=${id}`;
    const result = await new DB().ExecuteQuery(query);

    await new DB().ExecuteQuery(`delete from question where testid=${id}`);
    console.log(questions)

    for(let q of Object.values(questions)){
        const queryQ = `insert into question (question, testid) values
('${q.question}', ${id})`;
        await new DB().ExecuteQuery(queryQ);
    }
    return {
        Id: result?.insertId,
    };
};

const removeTest = async (userId, id) => {
    await new DB().ExecuteQuery(`delete from question where testid=${id}`);
    await new DB().ExecuteQuery(`delete from test where id=${id} and creatorid =
${userId}`);
    return;
};

const getTests = async (userId, creator) => {
    if(creator == 0){
        const results = await new DB().ExecuteQuery(`select t.* from test t where
t.creatorid=${userId}`);
        for(let r of results){
            const resultsQ = await new DB().ExecuteQuery(`select * from question
where testid=${r.id}`);
            const resultsA = await new DB().ExecuteQuery(`select count(*) as
assigns from assigntest where testid=${r.id}`);
            r['questions'] = resultsQ
            r['assigns'] = resultsA[0].assigns
        }
        return results.length > 0 ? results : null;
```

```
    }
    else{
        const results = await new DB().ExecuteQuery(`select t.*, (
                                                        case
                                                            when a.status = 1 then
'To test'
                                                            when a.status = 2 then
'Review'
                                                            else 'Checked'
                                                        end
                                                    ) as status, a.comments from
test t
                                                    inner join assigntest a on
a.testid = t.id
                                                    where a.userid = ${userId}`);
        for(let r of results){
            const resultsQ = await new DB().ExecuteQuery(`select * from question
where testid=${r.id}`);
            r['questions'] = resultsQ
        }
        return results.length > 0 ? results : null;
    }
};

const assignTest = async (testId, clientId) => {
    await new DB().ExecuteQuery(`insert into assigntest (userid, testid) values
(${clientId}, ${testId})`);
    return;
};

const getActiveTest = async (id) => {
    const results = await new DB().ExecuteQuery(`select t.* from test t where
t.id=${id}`);
    for(let r of results){
        const resultsQ = await new DB().ExecuteQuery(`select * from question where
testid=${r.id}`);
        r['questions'] = resultsQ
    }
    return results.length > 0 ? results[0] : null;
};

const answerActiveTest = async (userId, creatorId, testId, answers) => {
    await new DB().ExecuteQuery(`update assigntest set status=2 where
testid=${testId} and userid=${userId}`);
    for(let a of answers){
        await new DB().ExecuteQuery(`insert into marks (creatorid, testid,
questionid, mark) values (${creatorId}, ${testId}, ${a.id}, ${a.answer})`);
    }
    return;
};
```

```
const clientTests = async (userId) => {
    const results = await new DB().ExecuteQuery(`select a.id as assignId, a.status,
t.*, sum(m.mark) as totalMark from test t
                                                inner join assigntest a on a.testid
= t.id
                                                inner join question q on
q.testid=t.id
                                                inner join marks m on
m.questionid=q.id
                                                where a.userid = ${userId}`);
    for(let t of results){
        const resultsQ = await new DB().ExecuteQuery(`select q.*, m.mark from test
t
                                                        inner join assigntest a on
a.testid = t.id
                                                        inner join question q on
q.testid=t.id
                                                        inner join marks m on
m.questionid=q.id
                                                        where a.userid = ${userId}
and t.id=${t.id}`);
        t.question = resultsQ
    }
    return results.length > 0 ? results : null;
};

const clientTestComment = async (assignId, comment) => {
    await new DB().ExecuteQuery(`update assigntest set comments='${comment}',
status=3 where id=${assignId}`);
    return;
};

module.exports = {
    getInfoByEmail,
    getClients,
    removeClient,
    addClient,
    getTests,
    addTest,
    editTest,
    removeTest,
    assignTest,
    getActiveTest,
    answerActiveTest,
    clientTests,
    clientTestComment
};
```

controllers/tests.js

```
const requests = require('./requests');
const { HttpCode } = require('../helpers/constants');

require('dotenv').config();

const getAllTests = async (req, res) => {
    const creator = req.user.Creator
    const clients = await requests.getTests(req.user.id, creator)

    return res.status(HttpCode.OK).json({
        clients
    });
};

const getActiveTest = async (req, res) => {
    const id = req.body.id
    const test = await requests.getActiveTest(id)

    return res.status(HttpCode.OK).json({
        test
    });
};

const addTest = async (req, res) => {
    const { questions, name } = req.body;

    const addUserResult = await requests.addTest({ name, questions, creator:
req.user.id });

    return res.status(HttpCode.OK).json({
        message: 'Successfully added',
        addUserResult
    });
};

const editTest = async (req, res) => {
    const { questions, name, id } = req.body;

    const addUserResult = await requests.editTest({ id, name, questions, creator:
req.user.id });

    return res.status(HttpCode.OK).json({
        message: 'Successfully edited',
        addUserResult
    });
};

const removeTest = async (req, res) => {
```

```
    await requests.removeTest(req.user.id, req.body.id)

    return res.status(HttpCode.OK).json({
        message: 'Successfully removed'
    });
};

const assignTest = async (req, res) => {
    await requests.assignTest(req.body.testId, req.body.clientId)

    return res.status(HttpCode.OK).json({
        message: 'Successfully assigned'
    });
};

const answerActiveTest = async (req, res) => {
    const userId = req.user.id
    await requests.answerActiveTest(userId, req.body.creatorId, req.body.testId,
req.body.answers)

    return res.status(HttpCode.OK).json({
        message: 'Successfully assigned'
    });
};

const clientTests = async (req, res) => {
    const userId = req.body.id
    const result = await requests.clientTests(userId)

    return res.status(HttpCode.OK).json({
        message: 'Successfully assigned',
        tests: result
    });
};

const clientTestComment = async (req, res) => {
    const {assignId, comment} = req.body
    const result = await requests.clientTestComment(assignId, comment)

    return res.status(HttpCode.OK).json({
        message: 'Successfully assigned',
        tests: result
    });
};

module.exports = {
    getAllTests,
    addTest,
    editTest,
    removeTest,
```

```
    assignTest,
    getActiveTest,
    answerActiveTest,
    clientTests,
    clientTestComment
};
```

helpers/constants.js

```
const HttpCode = {
    OK: 200,
    CREATED: 201,
    NO_CONTENT: 204,
    BAD_REQUEST: 400,
    UNAUTHORIZED: 401,
    NOT_FOUND: 404,
    CONFLICT: 409,
    SERVER_ERROR: 500,
};

module.exports = {
    HttpCode,
};
```

helpers/controllerHandler.js

```
const controllerHandler = (controller) => {
  return async (req, res, next) => {
    try {
      await controller(req, res, next);
    } catch (error) {
      next(error);
    }
  };
};

module.exports = controllerHandler;
```

helpers/DB.js

```
const sql = require('mysql');
require('dotenv').config();
class DB {
  constructor(query) {
    this.query = query;
```

```
  }
  async ExecuteQuery(query) {
    console.log('query', query);
    return new Promise((resolve) => {
      try {
        this.connection = sql.createConnection({
          host: process.env.DB_SERVER,
          user: process.env.DB_USER,
          password: process.env.DB_PWD,
          database: process.env.DB_NAME,
        });
        this.connection.connect();
        this.connection.query(`${query}`, (error, results, fields) => {
          if (error) {
            console.log('query', query);
            console.log(error);
            resolve(error);
            return;
          }
          // console.log(results)
          // let result = {}
          // result.recordset = results
          // result.recordsets = results
          if (typeof results[Symbol.iterator] === 'function') {
            resolve([...results]);
            return;
          } else {
            resolve(results);
            return;
          }
        });
        this.connection.end();
      } catch (error) {
        console.log('query', query);
        console.error(error);
      }
    });
  }
}
module.exports = DB;
```

helpers/guard.js

```
const { HttpCode } = require("./constants");
const jwt = require("jsonwebtoken-refresh");

require("dotenv").config();
const JWT_SECRET = process.env.JWT_SECRET;
```

```
const guard = async (req, res, next) => {
  try {
    const token = req.session.token;
    if (!token) {
      return res.status(HttpCode.UNAUTHORIZED).json({
        message: "Session doesn't exist",
      });
    }

    jwt.verify(token, JWT_SECRET);
    const originalDecoded = jwt.decode(token, { complete: true });
    const refreshed = jwt.refresh(originalDecoded, "7d", JWT_SECRET);
    req.session.token = refreshed;
    req.user = originalDecoded.payload
    return next();
  } catch (error) {
    console.log(error);
    req.session.token = null;
    return res.status(HttpCode.UNAUTHORIZED).json({
      message: "Session expired",
    });
  }
};

module.exports = guard;
```

helpers/password.js

```
const bcrypt = require('bcrypt');

const hashWithSalt = (password, saltNumber) => {
  const salt = bcrypt.genSaltSync(Number(saltNumber));
  return bcrypt.hashSync(password, salt);
};

const compare = async (password, dbPassword) => {
  return await bcrypt.compare(password, dbPassword);
};

module.exports = { hashWithSalt, compare };
```

Клієнт

index.js

```
import React from "react";
import ReactDOM from "react-dom";
import { Provider } from "react-redux";
import { BrowserRouter } from "react-router-dom";
import "./index.css";
import App from "./App";
import { store } from "./redux/store";

ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </Provider>
  </React.StrictMode>,
  document.getElementById("root")
);
```

App.js

```
import { useDispatch } from "react-redux";
import { Routes, Route } from "react-router-dom";
import SingIn from "./components/singIn/SingIn";
import SingUp from "./components/singUp/SingUp";
import Clients from "./components/clients/Clients";
import Tests from "./components/tests/Tests";
import ActiveTest from "./components/takeTest/ActiveTest";
import ClientResults from "./components/clientResults/ClientResults";
import Error from "./components/error/Error";
import { refreshSessionOperation } from "./redux/user/user.operation";
import { useEffect, useState } from "react";

import PrivateRoute from "./components/Route/PrivateRoute";
import PublicRoute from "./components/Route/PublicRoute";

function App() {
  const dispatch = useDispatch();

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

  return (
    <div>
      <Routes>
        <Route exat path="/sign-in" element={<PublicRoute />}>
          <Route path="/sign-in" element={<SingIn />} />
        </Route>
```

```
        <Route exat path="/sign-up" element={<PublicRoute />}>
          <Route path="/sign-up" element={<SingUp />} />
        </Route>

        <Route exat path="/" element={<PrivateRoute />}>
          <Route path="/" element={<Clients />} />
        </Route>
        <Route exat path="/tests" element={<PrivateRoute />}>
          <Route path="/tests" element={<Tests />} />
        </Route>
        <Route exat path="/tests/test" element={<PrivateRoute />}>
          <Route path="/tests/test" element={<ActiveTest />} />
        </Route>
        <Route exat path="/clients/test" element={<PrivateRoute />}>
          <Route path="/clients/test" element={<ClientResults />} />
        </Route>
      </Routes>


      <Error />
    </div>
  );
}

export default App;
```

services/api.js

```
import axios from "axios";

const clientApi = axios.create({
  withCredentials: true
})

export const signin = async (data) =>
  clientApi
    .post("/api/auth/signin", data)
    .then((result) => result)
    .catch((reason) => reason.response)

export const signup = async (data) =>
    clientApi
      .post("/api/auth/signup", data)
      .then((result) => result)
      .catch((reason) => reason.response)

export const logout = async () =>
```

```
  clientApi
    .post("/api/auth/logout")
    .then((result) => result)
    .catch((reason) => reason.response)

export const refreshSession = async () =>
    clientApi
      .get("/api/auth/refreshSession")
      .then((result) => result)
      .catch((reason) => reason.response)

export const getClients = async () =>
    clientApi
      .get("/api/clients/")
      .then((result) => result)
      .catch((reason) => reason.response)

export const addClient = async (data) =>
    clientApi
      .post("/api/clients/", data)
      .then((result) => result)
      .catch((reason) => reason.response)

export const removeClient = async (data) =>
    clientApi
      .post("/api/clients/delete", data)
      .then((result) => result)
      .catch((reason) => reason.response)

export const addTest = async (data) =>
    clientApi
      .post("/api/tests/add", data)
      .then((result) => result)
      .catch((reason) => reason.response)

export const editTest = async (data) =>
    clientApi
      .post("/api/tests/edit", data)
      .then((result) => result)
      .catch((reason) => reason.response)

export const removeTest = async (data) =>
    clientApi
      .post("/api/tests/remove", data)
      .then((result) => result)
      .catch((reason) => reason.response)

export const getTests = async () =>
    clientApi
      .get("/api/tests/all")
```

```
      .then((result) => result)
      .catch((reason) => reason.response)

export const assignTest = async (data) =>
    clientApi
      .post("/api/tests/assign", data)
      .then((result) => result)
      .catch((reason) => reason.response)

export const getActiveTest = async (data) =>
    clientApi
      .post("/api/tests/getActive", data)
      .then((result) => result)
      .catch((reason) => reason.response)

export const answerActiveTest = async (data) =>
    clientApi
      .post("/api/tests/answerActive", data)
      .then((result) => result)
      .catch((reason) => reason.response)

export const clientTests = async (data) =>
    clientApi
      .post("/api/tests/clientTests", data)
      .then((result) => result)
      .catch((reason) => reason.response)

export const clientTestComment = async (data) =>
    clientApi
      .post("/api/tests/clientTestComment", data)
      .then((result) => result)
      .catch((reason) => reason.response)
```

redux/store.js

```
import { configureStore } from "@reduxjs/toolkit";
import rootReducer from "./root.reducer";

const store = configureStore({
  reducer: rootReducer,
});

export { store };
```

redux/root.reducer.js

```
import { combineReducers } from "@reduxjs/toolkit";
```

```
import {
  userReducer,
  clientsReducer,
  testsReducer,
  activeTestReducer,
  loading,
  errorMsg,
  clientTestsReducer
} from "./user/user.reducer";

const rootReducer = combineReducers({
  user: userReducer,
  clients: clientsReducer,
  tests: testsReducer,
  activeTest: activeTestReducer,
  clientTests: clientTestsReducer,
  errorMsg,
  loading
});

export default rootReducer;
```

redux/user/user.actions.js

```
import { createAction } from "@reduxjs/toolkit";

const signUpRequest = createAction("Auth/signUpRequest");
const signUpSuccess = createAction("Auth/signUpSuccess");
const signUpError = createAction("Auth/signUpError");

const signInRequest = createAction("Auth/signInRequest");
const signInSuccess = createAction("Auth/signInSuccess");
const signInError = createAction("Auth/signInError");

const logoutRequest = createAction("Auth/logoutRequest");

const refreshSessionRequest = createAction("User/refreshSessionRequest");
const refreshSessionSuccess = createAction("User/refreshSessionSecces");
const refreshSessionError = createAction("User/refreshSessionError");

const getClientsRequest = createAction("User/getClientsRequest");
const getClientsSuccess = createAction("User/getClientsSecces");
const getClientsError = createAction("User/getClientsError");

const removeClientRequest = createAction("User/removeClientRequest");
const removeClientSuccess = createAction("User/removeClientSecces");
const removeClientError = createAction("User/removeClientError");

const addClientRequest = createAction("User/addClientRequest");
const addClientSuccess = createAction("User/addClientSecces");
```

```
const addClientError = createAction("User/addClientError");

const addTestRequest = createAction("User/addTestRequest");
const addTestSuccess = createAction("User/addTestSecces");
const addTestError = createAction("User/addTestError");

const editTestRequest = createAction("User/editTestRequest");
const editTestSuccess = createAction("User/editTestSecces");
const editTestError = createAction("User/editTestError");

const removeTestRequest = createAction("User/removeTestRequest");
const removeTestSuccess = createAction("User/removeTestSecces");
const removeTestError = createAction("User/removeTestError");

const getTestsRequest = createAction("User/getTestsRequest");
const getTestsSuccess = createAction("User/getTestsSecces");
const getTestsError = createAction("User/getTestsError");

const assignTestRequest = createAction("User/assignTestRequest");
const assignTestSuccess = createAction("User/assignTestSecces");
const assignTestError = createAction("User/assignTestError");

const getActiveTestRequest = createAction("User/getActiveTestRequest");
const getActiveTestSuccess = createAction("User/getActiveTestSecces");
const getActiveTestError = createAction("User/getActiveTestError");

const answerActiveTestRequest = createAction("User/answerActiveTestRequest");
const answerActiveTestSuccess = createAction("User/answerActiveTestSecces");
const answerActiveTestError = createAction("User/answerActiveTestError");

const clientTestsRequest = createAction("User/clientTestsRequest");
const clientTestsSuccess = createAction("User/clientTestsSecces");
const clientTestsError = createAction("User/clientTestsError");

const clientTestCommentRequest = createAction("User/clientTestCommentRequest");
const clientTestCommentSuccess = createAction("User/clientTestCommentSecces");
const clientTestCommentError = createAction("User/clientTestCommentError");

const resetError = createAction("Auth/resetError");

export {
  signUpRequest,
  signUpSuccess,
  signUpError,
  signInRequest,
  signInSuccess,
  signInError,
  logoutRequest,
  refreshSessionRequest,
  refreshSessionSuccess,
```

```
  refreshSessionError,
  getClientsRequest,
  getClientsSuccess,
  getClientsError,
  removeClientRequest,
  removeClientSuccess,
  removeClientError,
  addClientRequest,
  addClientSuccess,
  addClientError,
  addTestRequest,
  addTestSuccess,
  addTestError,
  editTestRequest,
  editTestSuccess,
  editTestError,
  removeTestRequest,
  removeTestSuccess,
  removeTestError,
  getTestsRequest,
  getTestsSuccess,
  getTestsError,
  assignTestRequest,
  assignTestSuccess,
  assignTestError,
  getActiveTestRequest,
  getActiveTestSuccess,
  getActiveTestError,
  answerActiveTestRequest,
  answerActiveTestSuccess,
  answerActiveTestError,
  clientTestsRequest,
  clientTestsSuccess,
  clientTestsError,
  clientTestCommentRequest,
  clientTestCommentSuccess,
  clientTestCommentError,
  resetError,
};
```

redux/user/user.operation.js

```
import {
  signin,
  signup,
  logout,
  refreshSession,
  getClients,
  addClient,
```

```
  removeClient,
  addTest,
  editTest,
  removeTest,
  getTests,
  assignTest,
  getActiveTest,
  answerActiveTest,
  clientTests,
  clientTestComment
} from "../../services/api";
import {
  signUpRequest,
  signUpSuccess,
  signUpError,

  signInRequest,
  signInSuccess,
  signInError,

  logoutRequest,

  refreshSessionRequest,
  refreshSessionSuccess,
  refreshSessionError,

  getClientsRequest,
  getClientsSuccess,
  getClientsError,
  removeClientRequest,
  removeClientSuccess,
  removeClientError,
  addClientRequest,
  addClientSuccess,
  addClientError,

  addTestRequest,
  addTestSuccess,
  addTestError,
  editTestRequest,
  editTestSuccess,
  editTestError,
  removeTestRequest,
  removeTestSuccess,
  removeTestError,
  getTestsRequest,
  getTestsSuccess,
  getTestsError,

  assignTestRequest,
```

```
  assignTestSuccess,
  assignTestError,

  getActiveTestRequest,
  getActiveTestSuccess,
  getActiveTestError,

  answerActiveTestRequest,
  answerActiveTestSuccess,
  answerActiveTestError,

  clientTestsRequest,
  clientTestsSuccess,
  clientTestsError,

  clientTestCommentRequest,
  clientTestCommentSuccess,
  clientTestCommentError,
} from "./user.actions";
import { toast } from 'react-toastify'

const errorHandler = (status, dispatch) => {
  if(status === 401){
    dispatch(logoutRequest())
  }
};

export const signUpOperation =
  ({ email, password, firstName, lastName }) =>
  (dispatch) => {
    dispatch(signUpRequest());
    signup({ email, password, firstName, lastName })
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(signUpError(result.data.message));
        } else {
          dispatch(signUpSuccess());
        }
      })
      .catch((error) => {
        dispatch(signUpError(error.message));
      });
};

export const signInOperation =
  ({ email, password }) =>
  (dispatch) => {
    dispatch(signInRequest());
    signin({ email, password })
```

```
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(signInError(result.data.message));
        } else {
          dispatch(signInSuccess(result.data));
        }
      })
      .catch((error) => {
        dispatch(signInError(error.message));
      });
};

export const logoutOperation = () => (dispatch) => {
  logout()
    .then((result) => {
      dispatch(logoutRequest());
    })
    .catch((error) => {
      console.log(error)
    });
};

export const refreshSessionOperation = () => (dispatch) => {
  dispatch(refreshSessionRequest());
  refreshSession()
    .then((result) => {
      if (result.status !== 200) {
        errorHandler(result.status, dispatch)
        dispatch(signInError(result.data.message));
      } else {
        dispatch(refreshSessionSuccess(result.data));
        dispatch(signInSuccess(result.data));
      }
    })
    .catch((error) => {
      dispatch(refreshSessionError(error.message));
    });
};

export const getClientsOperation = () =>
  (dispatch) => {
    dispatch(getClientsRequest());
    getClients()
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(getClientsError(result.data.message));
        } else {
          dispatch(getClientsSuccess(result.data.clients));
```

```
        }
      })
      .catch((error) => {
        dispatch(getClientsError(error.message));
      });
};

export const addClientOperation = (data) =>
  (dispatch) => {
    dispatch(addClientRequest());
    addClient(data)
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(addClientError(result.data.message));
        } else {
          dispatch(addClientSuccess(result.data.clients));
          dispatch(getClientsOperation())
        }
      })
      .catch((error) => {
        dispatch(addClientError(error.message));
      });
};

export const removeClientOperation = (id) =>
  (dispatch) => {
    dispatch(removeClientRequest());
    removeClient({id})
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(removeClientError(result.data.message));
        } else {
          dispatch(removeClientSuccess(result.data.clients));
          dispatch(getClientsOperation())
        }
      })
      .catch((error) => {
        dispatch(removeClientError(error.message));
      });
};

export const getTestsOperation = () =>
  (dispatch) => {
    dispatch(getTestsRequest());
    getTests()
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
```

```
          dispatch(getTestsError(result.data.message));
        } else {
          dispatch(getTestsSuccess(result.data.clients));
          dispatch(getClientsOperation())
        }
      })
      .catch((error) => {
        dispatch(getTestsError(error.message));
      });
};

export const addTestOperation = (data) =>
  (dispatch) => {
    dispatch(addTestRequest());
    addTest(data)
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(addTestError(result.data.message));
        } else {
          dispatch(addTestSuccess(result.data.clients));
          dispatch(getTestsOperation())
        }
      })
      .catch((error) => {
        dispatch(addTestError(error.message));
      });
};

export const editTestOperation = (data) =>
  (dispatch) => {
    dispatch(editTestRequest());
    editTest(data)
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(editTestError(result.data.message));
        } else {
          dispatch(editTestSuccess(result.data.clients));
          dispatch(getTestsOperation())
        }
      })
      .catch((error) => {
        dispatch(editTestError(error.message));
      });
};

export const removeTestOperation = (id) =>
  (dispatch) => {
    dispatch(removeTestRequest());
```

```
    removeTest({id})
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(removeTestError(result.data.message));
        } else {
          dispatch(removeTestSuccess(result.data.clients));
          dispatch(getTestsOperation())
        }
      })
      .catch((error) => {
        dispatch(removeTestError(error.message));
      });
};

export const assignTestOperation = (data) =>
  (dispatch) => {
    dispatch(assignTestRequest());
    assignTest(data)
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(assignTestError(result.data.message));
        } else {
          dispatch(assignTestSuccess(result.data.clients));
          dispatch(getTestsOperation())
        }
      })
      .catch((error) => {
        dispatch(assignTestError(error.message));
      });
};

export const getActiveTestOperation = (data) =>
  (dispatch) => {
    dispatch(getActiveTestRequest());
    getActiveTest(data)
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(getActiveTestError(result.data.message));
        } else {
          dispatch(getActiveTestSuccess(result.data.test));
        }
      })
      .catch((error) => {
        dispatch(getActiveTestError(error.message));
      });
};
```

```
export const answerActiveTestOperation = (data) =>
  (dispatch) => {
    dispatch(answerActiveTestRequest());
    answerActiveTest(data)
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(answerActiveTestError(result.data.message));
        } else {
          dispatch(answerActiveTestSuccess(result.data.test));
        }
      })
      .catch((error) => {
        dispatch(answerActiveTestError(error.message));
      });
};

export const clientTestsOperation = (data) =>
  (dispatch) => {
    dispatch(clientTestsRequest());
    clientTests(data)
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(clientTestsError(result.data.message));
        } else {
          dispatch(clientTestsSuccess(result.data.tests));
        }
      })
      .catch((error) => {
        dispatch(clientTestsError(error.message));
      });
};

export const clientTestCommentOperation = (data) =>
  (dispatch) => {
    dispatch(clientTestCommentRequest());
    clientTestComment(data)
      .then((result) => {
        if (result.status !== 200) {
          errorHandler(result.status, dispatch)
          dispatch(clientTestCommentError(result.data.message));
        } else {
          dispatch(clientTestCommentSuccess(result.data.tests));
          dispatch(clientTestsOperation)
        }
      })
      .catch((error) => {
        dispatch(clientTestCommentError(error.message));
      });
```

```
};
```

redux/user/user.reducer.js

```
import { createReducer } from "@reduxjs/toolkit";

import {
  signUpRequest,
  signUpSuccess,
  signUpError,
  signInRequest,
  signInSuccess,
  signInError,
  resetError,
  refreshSessionRequest,
  refreshSessionSuccess,
  refreshSessionError,
  getClientsRequest,
  getClientsSuccess,
  getClientsError,
  removeClientRequest,
  removeClientSuccess,
  removeClientError,
  addClientRequest,
  addClientSuccess,
  addClientError,
  addTestRequest,
  addTestSuccess,
  addTestError,
  editTestRequest,
  editTestSuccess,
  editTestError,
  removeTestRequest,
  removeTestSuccess,
  removeTestError,
  getTestsRequest,
  getTestsSuccess,
  getTestsError,
  assignTestRequest,
  assignTestSuccess,
  assignTestError,
  getActiveTestRequest,
  getActiveTestSuccess,
  getActiveTestError,
  answerActiveTestRequest,
  answerActiveTestSuccess,
  answerActiveTestError,
  clientTestsRequest,
  clientTestsSuccess,
  clientTestsError,
```

```
  clientTestCommentRequest,
  clientTestCommentSuccess,
  clientTestCommentError,
  logoutRequest,
} from "./user.actions";

export const userReducer = createReducer(null, {
  [signInSuccess]: (_, { payload }) => payload,
  [refreshSessionError]: (_, __) => null,
  [logoutRequest]: (_, __) => null,
});

export const clientsReducer = createReducer(null, {
  [getClientsSuccess]: (_, { payload }) => payload,
  [logoutRequest]: (_, __) => null,
});

export const testsReducer = createReducer(null, {
  [getTestsSuccess]: (_, { payload }) => payload,
  [logoutRequest]: (_, __) => null,
});

export const clientTestsReducer = createReducer(null, {
  [clientTestsSuccess]: (_, { payload }) => payload,
  [logoutRequest]: (_, __) => null,
});

export const activeTestReducer = createReducer(null, {
  [getActiveTestSuccess]: (_, { payload }) => payload,
  [logoutRequest]: (_, __) => null,
});

export const errorMsg = createReducer(null, {
  [resetError]: (_, __) => null,
  [signUpError]: (_, { payload }) => payload,
  [signInError]: (_, { payload }) => payload,
  [getClientsError]: (_, { payload }) => payload,
  [addClientError]: (_, { payload }) => payload,
  [removeClientError]: (_, { payload }) => payload,
  [addTestError]: (_, { payload }) => payload,
  [editTestError]: (_, { payload }) => payload,
  [removeTestError]: (_, { payload }) => payload,
  [getTestsError]: (_, { payload }) => payload,
  [assignTestError]: (_, { payload }) => payload,
  [getActiveTestError]: (_, { payload }) => payload,
  [answerActiveTestError]: (_, { payload }) => payload,
  [clientTestsError]: (_, { payload }) => payload,
  [clientTestCommentError]: (_, { payload }) => payload,
});
```

```
export const loading = createReducer(false, {
  [addTestRequest]: () => true,
  [addTestSuccess]: () => false,
  [addTestError]: () => false,
  [clientTestsRequest]: () => true,
  [clientTestsSuccess]: () => false,
  [clientTestsError]: () => false,
  [clientTestCommentRequest]: () => true,
  [clientTestCommentSuccess]: () => false,
  [clientTestCommentError]: () => false,
  [answerActiveTestRequest]: () => true,
  [answerActiveTestSuccess]: () => false,
  [answerActiveTestError]: () => false,
  [getActiveTestRequest]: () => true,
  [getActiveTestSuccess]: () => false,
  [getActiveTestError]: () => false,
  [assignTestRequest]: () => true,
  [assignTestSuccess]: () => false,
  [assignTestError]: () => false,
  [editTestRequest]: () => true,
  [editTestSuccess]: () => false,
  [editTestError]: () => false,
  [removeTestRequest]: () => true,
  [removeTestSuccess]: () => false,
  [removeTestError]: () => false,
  [getTestsRequest]: () => true,
  [getTestsSuccess]: () => false,
  [getTestsError]: () => false,
  [getClientsRequest]: () => true,
  [getClientsSuccess]: () => false,
  [getClientsError]: () => false,
  [removeClientRequest]: () => true,
  [removeClientSuccess]: () => false,
  [removeClientError]: () => false,
  [addClientRequest]: () => true,
  [addClientSuccess]: () => false,
  [addClientError]: () => false,
  [signUpRequest]: () => true,
  [signUpSuccess]: () => false,
  [signUpError]: () => false,
  [signInRequest]: () => true,
  [signInSuccess]: () => false,
  [signInError]: () => false,
});
```

redux/user/user.selectors.js

```
export const getUser = (state) => state.user;

export const getClients = (state) => state.clients;
```

```
export const getTests = (state) => state.tests;

export const getClientTests = (state) => state.clientTests;

export const getActiveTest = (state) => state.activeTest;

export const errorMsg = (state) => state.errorMsg;

export const getLoading = (state) => state.loading;
```

components/clientResults/ClientResults.js

```
import { useEffect, useState } from "react";
import { clientTestsOperation, answerActiveTestOperation } from
"../../redux/user/user.operation";
import { useDispatch, useSelector } from "react-redux";
import { getLoading, getClientTests, getUser } from
"../../redux/user/user.selectors";
import { useSearchParams, useNavigate  } from "react-router-dom";
import BtnBlue from "../buttons/BtnBlue";
import CommentTestModal from "../modal/CommentTestModal"
import style from "./Tests.module.css";
import Container from "../container/Container";
import Spiner from "../spiner/Spiner";

import Header from "../header/Header";

const ClientResults = () => {
    let [searchParams, setSearchParams] = useSearchParams();
    const tests = useSelector(getClientTests);
    const [isOpenComment, setIsOpenComment] = useState(false);
    const [testToComment, setTestToComment] = useState(null);
    const togleComment = (e) => {
        if(isOpenComment == true){
            setTestToComment(null)
        }
        else{
            setTestToComment(e.currentTarget.getAttribute("rowid"))
        }
        setIsOpenComment(!isOpenComment)
    };
    const dispatch = useDispatch();
    const navigate = useNavigate()
    useEffect(() => {
        dispatch(clientTestsOperation({id: searchParams.get("id")}))
    }, [dispatch]);

    const loading = useSelector(getLoading);
    const user = useSelector(getUser);
```

```
const isAdmin = user.user.Creator == 0 ? true : false

return (
    <Container>
    {loading ? (
        <div className={style.flexSpiner}>
        <Spiner style={{ color: "black", fontSize: "4em" }} />
        </div>
    ) : (
        <>
            <Header />
            <div style={{
                display: "flex",
                margin: "10px 10%"
            }}>
                <p style={{
                    fontSize: "28px",
                    color: "black",
                    fontWeight: "600",
                    width: "95%",
                    textAlign: "start"
                }}>
                    Client results
                </p>
            </div>
            <div style={{
                display: "flex",
                margin: "10px 10%",
                flexDirection: "row"
            }}>
                {tests?.map((t) => (
                    <div style={{
                        width: "20%",
                        padding: "10px",
                        border: "1px solid grey",
                        borderRadius: "4px",
                        marginRight: "10%"
                    }}>
                        <p style={{
                            fontSize: "20px",
                            color: "black",
                            fontWeight: "600",
                            width: "95%",
                            textAlign: "start"
                        }}>
                            {t.name}
                        </p>
                        <div style={{
                            display: "flex",
                            margin: "10px 0",
```

```
                flexDirection: "column"
            }}>
                {t.question.length > 0 ? (
                    <>
                        {t.question.map((q) => (
                            <>
                                <p style={{
                                    fontSize: "16px",
                                    fontWeight: "600",
                                    color: "black",
                                    width: "95%",
                                    textAlign: "start"
                                }}>
                                    {q.question}
                                </p>
                                <p style={{
                                    fontSize: "16px",
                                    color: "black",
                                    width: "95%",
                                    textAlign: "start",
                                    marginBottom: "10px"
                                }}>
                                    {q.mark}
                                </p>
                            </>
                        ))}
                        <p style={{
                            fontSize: "16px",
                            fontWeight: "600",
                            color: "black",
                            width: "95%",
                            textAlign: "start"
                        }}>
                            Total
                        </p>
                        <p style={{
                            fontSize: "16px",
                            color: "black",
                            width: "95%",
                            textAlign: "start",
                            marginBottom: "10px"
                        }}>
                            {t.totalMark}
                        </p>
                    </>
                ) : (
                    <p style={{
                        fontSize: "16px",
                        color: "black",
                        width: "95%",
```

```
                                        textAlign: "start"
                                    }}>
                                        No answered questions
                                    </p>
                                )}
                                {t.status === 3 ? (
                                    <BtnBlue btnStyle={{
                                        color: "white",
                                        background: "grey",
                                        borderColor: "grey",
                                        width: "100%",
                                        height: "30px"
                                    }} name={"Already submited"}
                                    onClick={togleComment}
                                    rowid={t.assignId}
                                    disable={true}/>
                                ) : (
                                    <BtnBlue btnStyle={{
                                        color: "white",
                                        width: "100%",
                                        height: "30px"
                                    }} name={"Submit and Add comment"}
                                    onClick={togleComment}
                                    rowid={t.assignId}/>
                                )}
                            </div>
                        </div>
                    ))}
                </div>
                {isOpenComment && (
                    <CommentTestModal
                        name={"Add comment"}
                        testId={testToComment}
                        togleModal={togleComment}
                        display={{ display: "none" }}
                    />
                )}
            </>
        )}
        </Container>
    );
};

export default ClientResults;
```

components/clients/Clients.js

```
import { useEffect, useState } from "react";
import { getClientsOperation } from "../../redux/user/user.operation";
import { useDispatch, useSelector } from "react-redux";
```

```
import { getLoading, getClients, getUser } from "../../redux/user/user.selectors";
import BtnBlue from "../buttons/BtnBlue";
import Row from "../row/Row";
import TableHeader from "../row/TableHeader";
import AddClientModal from "../modal/AddClientModal"
import style from "./Clients.module.css";
import Container from "../container/Container";
import Spiner from "../spiner/Spiner";
import { Navigate } from "react-router-dom";

import Header from "../header/Header";

const Clients = () => {
    const clients = useSelector(getClients);
    const [isOpenAdd, setIsOpenAdd] = useState(false);
    const togleAdd = () => setIsOpenAdd(!isOpenAdd);
    const dispatch = useDispatch();
    useEffect(() => {
        dispatch(getClientsOperation())
    }, [dispatch]);

    const loading = useSelector(getLoading);
    const user = useSelector(getUser);
    const isAdmin = user.user.Creator == 0 ? true : false

    return (
        <Container>
        {!isAdmin ? (<Navigate to="/tests" />) : (<></>)}
        {loading ? (
            <div className={style.flexSpiner}>
            <Spiner style={{ color: "black", fontSize: "4em" }} />
            </div>
        ) : (
            <>
                <Header />
                <div style={{
                    display: "flex",
                    margin: "10px 10%"
                }}>
                    <p style={{
                        fontSize: "28px",
                        color: "black",
                        fontWeight: "600",
                        width: "95%",
                        textAlign: "start"
                    }}>Clients</p>
                    <BtnBlue onClick={togleAdd} btnStyle={{
                        width: "5%",
                        color: "white",
                        textAlign: "center",
```

```
                        height: "40px"
                    }} name={'Add new'}>Add new</BtnBlue>
                </div>
                {clients ? (
                    <>
                        <div style={{
                            display: "flex",
                            flexDirection: "column",
                            margin: "10px 10% 0 10%"
                        }}>
                            <TableHeader data={clients} type={"client"} />
                        </div><div style={{
                            display: "flex",
                            flexDirection: "column",
                            margin: "0 10%"
                        }}>
                            {clients.map((client) => (
                                <Row data={client} type={"client"} key={client.id}
/>
                            ))}
                        </div>
                    </>
                ) : (
                    <h1>No clients yet</h1>
                )}
                {isOpenAdd && (
                    <AddClientModal
                        name={"Add New Client"}
                        togleModal={togleAdd}
                        display={{ display: "none" }}
                    />
                )}
            </>
        )}
        </Container>
    );
};

export default Clients;
```

components/container/Container.js

```
import style from "./Container.module.css";

function Container({ children }) {
  return <div className={style.container}>{children}</div>;
}

export default Container;
```

components/error/Error.js

```
import React, { useEffect } from 'react'
import { useSelector, useDispatch } from 'react-redux'

import { errorMsg } from '../../redux/user/user.selectors'
import { resetError } from '../../redux/user/user.actions'
import { ToastContainer, toast } from 'react-toastify'
import 'react-toastify/dist/ReactToastify.css'

const notifyErrorOptions = {
  position: 'top-center',
  autoClose: 2000,
  hideProgressBar: true,
  closeOnClick: false,
  pauseOnHover: true,
  draggable: true,
  progress: undefined,
}

function Error() {
    const authError = useSelector(errorMsg)
    const dispatch = useDispatch()

    useEffect(() => {
        if(authError){
            toast.error(authError, notifyErrorOptions)
        }
        else{
            dispatch(resetError())
        }
    }, [authError, dispatch])

    return (
        <div>
            <ToastContainer newestOnTop rtl={false} pauseOnFocusLoss={false} />
        </div>
    )
}

export default Error
```

components/header/Header.js

```
import sprite from "../../img/sprite.svg";
import style from "./Header.module.css";
import { useDispatch, useSelector } from "react-redux";
import { getUser } from "../../redux/user/user.selectors";
```

```
import { logoutOperation } from "../../redux/user/user.operation";
import { Link } from "react-router-dom";

const Header = () => {
  const user = useSelector(getUser);
  const dispatch = useDispatch();
  const isAdmin = user.user.Creator == 0 ? true : false

  const logout = () => {
    dispatch(logoutOperation());
  };

  return (
    <div className={style.header}>
      <h1 style={{
        fontSize: "24px",
        fontWeight: "600"
      }}>Psychologist</h1>
      <div >
        {isAdmin ? (
          <Link className={style.nav} to="/">Clients</Link>
        ) : (<></>)}
        <Link className={style.nav} to="/tests">Tests</Link>
      </div>
      <div className={style.rightSide}>
        <p style={{paddingRight:'10px'}}>{user.user.FirstName + " " +
user.user.LastName}</p>
        <svg onClick={() => logout()} className={style.logOut}>
          <use href={sprite + "#icon-logout"}></use>
        </svg>
      </div>
    </div>
  );
};

export default Header;
```

components/header/Header.module.css

```
.header {
  background: var(--dark-text);
  display: flex;
  align-items: center;
  justify-content: space-between;

  width: auto;
  padding: 10px 40px 10px 40px;
}

.logoHead {
```

```
  width: 137px;
  height: 40px;
}

.logOut {
  width: 18px;
  height: 18px;

  cursor: pointer;
}

.rightSide {
  display: flex;
  align-items: center;
}
.whatsAppSetting {
  display: flex;
  align-items: center;
}

.whatsAppIcon {
  width: 24px;
  height: 24px;
  margin-right: 6px;
}

.connect {
  font-weight: 500;
  font-size: 14px;
  color: #00e676;
  margin-right: 6px;
  cursor: default;
}

.disconnect {
  font-size: 14px;
  text-decoration: underline;
  color: #fff;
  margin-right: 24px;
  cursor: pointer;
}

.loader {
  color: var(--grey-color);
  font-weight: 500;
  font-size: 14px;
  margin-left: 4px;
  margin-right: 24px;
}
```

```
.flexSpiner {
  display: flex;
  align-items: center;
}
.nav{
  text-decoration: none;
  color: white;
  font-size: 22px;
  margin-right: 20px;
}
.nav:hover{
  color: grey;
}
```

components/modal/AddClientModal.js

```
import React, { useRef, useState, useCallback, createElement } from "react";
import { useSelector, useDispatch } from "react-redux";
import style from "./AddModal.module.css";
import sprite from "../../img/sprite.svg";
import BtnGrey from "../buttons/BtnGrey";
import BtnWhite from "../buttons/BtnWhite";
import BtnBlue from "../buttons/BtnBlue";
import { styleBtn } from "./stylebtnModal";
import { addClientOperation } from "../../redux/user/user.operation";
import { useEffect } from "react";
import { isValidEmail } from "../../helpers/helpers";

const AddClientModal = ({
  name,
  togleModal,
  display,
  messageEdit,
  companyId,
}) => {
  const [firstName, setFirstName] = useState("");
  const [lastName, setLastName] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const dispatch = useDispatch();

  useEffect(() => {

  }, [dispatch]);

  const handleChange = (e) => {
    if (e.currentTarget.name === "firstName") {
        setFirstName(e.currentTarget.value);
    } else if(e.currentTarget.name === "lastName"){
```

```
      setLastName(e.currentTarget.value);
  } else if(e.currentTarget.name === "email"){
      setEmail(e.currentTarget.value);
  } else if(e.currentTarget.name === "password"){
      setPassword(e.currentTarget.value);
  }
};

const handleSubmit = async (e) => {
  e.preventDefault();

  dispatch(addClientOperation({firstName, lastName, email, password}));
  togleModal();
};

return (
  <div className={style.bgModal}>
    <form onSubmit={handleSubmit} className={style.form}>
      <svg onClick={togleModal} className={style.icon}>
        <use href={sprite + "#icon-close"}></use>
      </svg>
      <h2 className={style.head}>{name}</h2>
      <label className={style.label}>
        First name
        <input
          onChange={handleChange}
          className={style.input}
          name="firstName"
          type="text"
          value={firstName}
          style={
            firstName.length > 0
              ? { border: "1px solid #d3d5da" }
              : { color: "#2b3135", border: "1px solid #ff8616" }
          }
        ></input>
      </label>
      <label className={style.label}>
        Last name
        <input
          onChange={handleChange}
          className={style.input}
          name="lastName"
          type="text"
          value={lastName}
          style={
            lastName.length > 0
              ? { border: "1px solid #d3d5da" }
              : { color: "#2b3135", border: "1px solid #ff8616" }
          }
```

```
          ></input>
        </label>
        <label className={style.label}>
          Email
          <input
            onChange={handleChange}
            className={style.input}
            name="email"
            type="email"
            value={email}
            style={
              email.length > 0 && isValidEmail(email)
                ? { border: "1px solid #d3d5da" }
                : { color: "#2b3135", border: "1px solid #ff8616" }
            }
          ></input>
        </label>
        <label className={style.label}>
          Password
          <input
            onChange={handleChange}
            className={style.input}
            name="password"
            type="password"
            value={password}
            style={
              password.length >= 8
                ? { border: "1px solid #d3d5da" }
                : { color: "#2b3135", border: "1px solid #ff8616" }
            }
          ></input>
        </label>
        <div className={style.btnFlex}>
          <div onClick={togleModal}>
            <BtnWhite
              name={"Cancel"}
              iconStyle={{ display: "none" }}
              btnStyle={styleBtn.btnCancel}
            />
          </div>
          <BtnBlue
            type="submit"
            disable={
              !(firstName.length > 0 && lastName.length > 0 && email.length > 0 &&
password.length >= 8 && isValidEmail(email))
            }
            name={"Save"}
            iconStyle={{ display: "none" }}
            btnStyle={
```

```
                firstName.length > 0 && lastName.length > 0 && email.length > 0 &&
password.length >= 8 && isValidEmail(email)
                    ? styleBtn.btnSave
                    : styleBtn.btnDisable
            }
          />
        </div>
      </form>
    </div>
  );
};

export default AddClientModal;
```

components/modal/AddTestsModal.js

```
import React, { useRef, useState, useCallback, createElement } from "react";
import { useSelector, useDispatch } from "react-redux";
import uuid from 'react-uuid'
import style from "./AddModal.module.css";
import sprite from "../../img/sprite.svg";
import BtnGrey from "../buttons/BtnGrey";
import BtnWhite from "../buttons/BtnWhite";
import BtnBlue from "../buttons/BtnBlue";
import BtnRed from "../buttons/BtnRed";
import { styleBtn } from "./stylebtnModal";
import { addTestOperation } from "../../redux/user/user.operation";
import { useEffect } from "react";

const AddTestsModal = ({
  name,
  togleModal,
  display,
  messageEdit,
  companyId,
}) => {
  const [questions, setQuestions] = useState({});
  const [testName, setTestName] = useState('');
  const dispatch = useDispatch();

  useEffect(() => {

  }, [dispatch]);

  const handleChange = (e) => {
    if(questions[e.currentTarget.getAttribute('rowid')]){
      questions[e.currentTarget.getAttribute('rowid')].question =
e.currentTarget.value;
      setQuestions({...questions})
    }
```

```
};

const handleChangeName = (e) => {
  setTestName(e.currentTarget.value)
};

const addQuestion = (e) => {
  const id = uuid()
  questions[id] = {
    question: "",
    id
  }
  setQuestions({...questions})
}

const removeQuestion = (e) => {
  delete questions[e.currentTarget.id]
  setQuestions({...questions})
}

const handleSubmit = async (e) => {
  e.preventDefault();

  dispatch(addTestOperation({questions, name: testName}));
  togleModal();
};

return (
  <div className={style.bgModal}>
    <form className={style.form}>
      <svg onClick={togleModal} className={style.icon}>
        <use href={sprite + "#icon-close"}></use>
      </svg>
      <h2 className={style.head}>{name}</h2>
      <label className={style.label}>
        Test name
        <input
          onChange={handleChangeName}
          className={style.input}
          type="text"
          value={testName}
        ></input>
      </label>
      {Object.entries(questions).map((q) => (
        <label key={q[1].id}  className={style.label}>
          Question
          <input
            onChange={handleChange}
            className={style.input}
            type="text"
```

```
      value={q[1].question}
      rowid={q[1].id}
    ></input>
    <BtnRed
      onClick={removeQuestion}
      name={"Remove"}
      iconStyle={{ display: "none" }}
      btnStyle={{
        borderColor: "red",
        padding: "5px",
        color: "white"
      }}
      rowid={q[1].id}
    />
    <hr/>
  </label>
))}
<div style={{
    width: "100%"
  }} className={style.btnFlex}>
  <div style={{
    width: "30%"
  }} onClick={togleModal}>
    <BtnWhite
      name={"Cancel"}
      iconStyle={{ display: "none" }}
      btnStyle={{ height: "48px", width: "90%" }}
    />
  </div>
  <div style={{
    width: "30%"
  }} onClick={addQuestion}>
    <BtnWhite
      name={"Add question"}
      iconStyle={{ display: "none" }}
      btnStyle={{ height: "48px", width: "90%" }}
    />
  </div>
  <BtnBlue
    onClick={handleSubmit}
    disable={
      !(Object.entries(questions).length > 0 ())
    }
    name={"Save"}
    iconStyle={{ display: "none" }}
    btnStyle={
      Object.entries(questions).length > 0
            ? styleBtn.btnSave
            : styleBtn.btnDisable
    }
```

```
          />
        </div>
      </form>
    </div>
  );
};

export default AddTestsModal;
```

components/modal/AssignTestsModal.js

```
import React, { useRef, useState, useCallback, createElement } from "react";
import Select from 'react-select'
import { useSelector, useDispatch } from "react-redux";
import style from "./AddModal.module.css";
import sprite from "../../img/sprite.svg";
import BtnGrey from "../buttons/BtnGrey";
import BtnWhite from "../buttons/BtnWhite";
import BtnBlue from "../buttons/BtnBlue";
import BtnRed from "../buttons/BtnRed";
import { styleBtn } from "./stylebtnModal";
import { assignTestOperation } from "../../redux/user/user.operation";
import { useEffect } from "react";

const AssignTestsModal = ({
  name,
  togleModal,
  clients,
  testId,
  display,
  messageEdit,
  companyId,
}) => {
  const [clientId, setClientId] = useState('');
  const dispatch = useDispatch();

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

  const handleChange = (e) => {
    setClientId(e.value)
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    dispatch(assignTestOperation({testId, clientId}));
    togleModal();
  };
```

```
  const options = clients.map((c) => {
    return {
      value: c.id,
      label: c.FirstName+" "+c.LastName
    }
  })

  return (
    <div className={style.bgModal}>
      <form className={style.form}>
        <svg onClick={togleModal} className={style.icon}>
          <use href={sprite + "#icon-close"}></use>
        </svg>
        <h2 className={style.head}>{name}</h2>
        <label className={style.label}>
          Client
          <Select onChange={handleChange} options={options}/>
        </label>
        <div style={{
            width: "100%"
          }} className={style.btnFlex}>
          <div style={{
            width: "30%"
          }} onClick={togleModal}>
            <BtnWhite
              name={"Cancel"}
              iconStyle={{ display: "none" }}
              btnStyle={{ height: "48px", width: "90%" }}
            />
          </div>
          <BtnBlue
            onClick={handleSubmit}
            disable={
              !(clientId)
            }
            name={"Save"}
            iconStyle={{ display: "none" }}
            btnStyle={
              clientId
                    ? styleBtn.btnSave
                    : styleBtn.btnDisable
            }
          />
        </div>
      </form>
    </div>
  );
};

export default AssignTestsModal;
```

components/modal/CommentTestModal.js

```
import React, { useRef, useState, useCallback, createElement } from "react";
import { useSelector, useDispatch } from "react-redux";
import style from "./AddModal.module.css";
import sprite from "../../img/sprite.svg";
import BtnGrey from "../buttons/BtnGrey";
import BtnWhite from "../buttons/BtnWhite";
import BtnBlue from "../buttons/BtnBlue";
import { styleBtn } from "./stylebtnModal";
import { clientTestCommentOperation } from "../../redux/user/user.operation";
import { useEffect } from "react";

const CommentTestModal = ({
  name,
  togleModal,
  testId,
  display,
  messageEdit,
  companyId,
}) => {
  const [comment, setComment] = useState("");
  const dispatch = useDispatch();

  useEffect(() => {

  }, [dispatch]);

  const handleChange = (e) => {
    if (e.currentTarget.name === "comment") {
      setComment(e.currentTarget.value);
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    dispatch(clientTestCommentOperation({assignId: testId, comment}));
    togleModal();
  };

  return (
    <div className={style.bgModal}>
      <form onSubmit={handleSubmit} className={style.form}>
        <svg onClick={togleModal} className={style.icon}>
          <use href={sprite + "#icon-close"}></use>
        </svg>
        <h2 className={style.head}>{name}</h2>
        <label className={style.label}>
```

```
          Your comment
          <textarea
            aria-rowcount={8}
            rows={8}
            onChange={handleChange}
            className={style.input}
            name="comment"
            value={comment}
            style={{
              resize: "vertical",
              height: "150px",
              minHeight: "150px",
              maxHeight: "650px"
            }}
          ></textarea>
        </label>
        <div className={style.btnFlex}>
          <div onClick={togleModal}>
            <BtnWhite
              name={"Cancel"}
              iconStyle={{ display: "none" }}
              btnStyle={styleBtn.btnCancel}
            />
          </div>
          <BtnBlue
            type="submit"
            disable={
              comment.length < 16
            }
            name={"Submit"}
            iconStyle={{ display: "none" }}
            btnStyle={
                  comment.length >= 16
                    ? styleBtn.btnSave
                    : styleBtn.btnDisable
            }
          />
        </div>
      </form>
    </div>
  );
};

export default CommentTestModal;
```

components/modal/EditTestsModal.js

```
import React, { useRef, useState, useCallback, createElement } from "react";
import { useSelector, useDispatch } from "react-redux";
```

```
import uuid from 'react-uuid'
import style from "./AddModal.module.css";
import sprite from "../../img/sprite.svg";
import BtnGrey from "../buttons/BtnGrey";
import BtnWhite from "../buttons/BtnWhite";
import BtnBlue from "../buttons/BtnBlue";
import BtnRed from "../buttons/BtnRed";
import { styleBtn } from "./stylebtnModal";
import { editTestOperation } from "../../redux/user/user.operation";
import { useEffect } from "react";

const AddTestsModal = ({
  name,
  togleModal,
  data,
  testId,
  display,
  messageEdit,
  companyId,
}) => {
  const [questions, setQuestions] = useState({});
  const [testName, setTestName] = useState('');
  const dispatch = useDispatch();

  useEffect(() => {
    for(let t of data){
      if(t.id === +testId){
        setTestName(t.name)
        for(let q of t.questions){
          addQuestionEdit(q)
        }
      }
    }
  }, [dispatch]);

  const handleChange = (e) => {
    if(questions[e.currentTarget.getAttribute('rowid')]){
      questions[e.currentTarget.getAttribute('rowid')].question =
e.currentTarget.value;
      setQuestions({...questions})
    }
  };

  const handleChangeName = (e) => {
    setTestName(e.currentTarget.value)
  };

  const addQuestion = (e) => {
    const id = uuid()
    questions[id] = {
```

```
    question: "",
    id
  }
  setQuestions({...questions})
}

const addQuestionEdit = (q) => {
  const id = uuid()
  questions[id] = {
    question: q.question,
    id
  }
  setQuestions({...questions})
}

const removeQuestion = (e) => {
  delete questions[e.currentTarget.id]
  setQuestions({...questions})
}

const handleSubmit = async (e) => {
  e.preventDefault();

  dispatch(editTestOperation({questions, name: testName, id: testId}));
  togleModal();
};

return (
  <div className={style.bgModal}>
    <form className={style.form}>
      <svg onClick={togleModal} className={style.icon}>
        <use href={sprite + "#icon-close"}></use>
      </svg>
      <h2 className={style.head}>{name}</h2>
      <label className={style.label}>
        Test name
        <input
          onChange={handleChangeName}
          className={style.input}
          type="text"
          value={testName}
        ></input>
      </label>
      {Object.entries(questions).map((q) => (
        <label key={q[1].id}  className={style.label}>
          Question
          <input
            onChange={handleChange}
            className={style.input}
            type="text"
```

```
      value={q[1].question}
      rowid={q[1].id}
    ></input>
    <BtnRed
      onClick={removeQuestion}
      name={"Remove"}
      iconStyle={{ display: "none" }}
      btnStyle={{
        borderColor: "red",
        padding: "5px",
        color: "white"
      }}
      rowid={q[1].id}
    />
    <hr/>
  </label>
))}
<div style={{
    width: "100%"
  }} className={style.btnFlex}>
  <div style={{
    width: "30%"
  }} onClick={togleModal}>
    <BtnWhite
      name={"Cancel"}
      iconStyle={{ display: "none" }}
      btnStyle={{ height: "48px", width: "90%" }}
    />
  </div>
  <div style={{
    width: "30%"
  }} onClick={addQuestion}>
    <BtnWhite
      name={"Add question"}
      iconStyle={{ display: "none" }}
      btnStyle={{ height: "48px", width: "90%" }}
    />
  </div>
  <BtnBlue
    onClick={handleSubmit}
    disable={
      !(Object.entries(questions).length > 0)
    }
    name={"Save"}
    iconStyle={{ display: "none" }}
    btnStyle={
      Object.entries(questions).length > 0
            ? styleBtn.btnSave
            : styleBtn.btnDisable
    }
```

```
          />
        </div>
      </form>
    </div>
  );
};

export default AddTestsModal;
```

components/Route/PrivateRoute.js

```
import { useSelector } from "react-redux";
import { getUser } from "../../redux/user/user.selectors";
import { Navigate, Outlet } from "react-router-dom";

const PrivateRoute = () => {
  const user = useSelector(getUser);
  return user ? <Outlet /> : <Navigate to="/sign-in" />;
};

export default PrivateRoute;
```

components/Route/PublicRoute.js

```
import { useSelector } from "react-redux";
import { getUser } from "../../redux/user/user.selectors";
import { Navigate, Outlet } from "react-router-dom";

const PublicRoute = () => {
  const user = useSelector(getUser);
  return user ? <Navigate to="/"/> : <Outlet />;
};

export default PublicRoute;
```

components/row/Row.js

```
import style from "./Row.module.css";
import BtnRed from "../buttons/BtnRed"
import { useState } from "react"
import { removeClientOperation, removeTestOperation } from
"../../redux/user/user.operation";
import { useDispatch, useSelector } from "react-redux";
import { Link } from "react-router-dom";

const Row = ({data, type, togleEdit, togleAssign, isAdmin}) => {
    const dispatch = useDispatch();
    const [showComment, setShowComment] = useState(false)
```

```
const removeClient = (e) => {
    dispatch(removeClientOperation(e.currentTarget.getAttribute("rowid")))
}
const removeTest = (e) => {
    dispatch(removeTestOperation(e.currentTarget.getAttribute("rowid")))
}

const onHoverEnter = (e) => {
    setShowComment(true)
}
const onHoverLeave = (e) => {
    setShowComment(false)
}

return (
    <div className={style.row}>
        {type === "client" ? (
            <>
                <div style={{
                    width: "10%",
                    borderRight: "1px solid grey"
                }}>
                    {data.id}
                </div>
                <div style={{
                    width: "25%",
                    borderRight: "1px solid grey"
                }}>
                    {data.FirstName}
                </div>
                <div style={{
                    width: "25%",
                    borderRight: "1px solid grey"
                }}>
                    {data.LastName}
                </div>
                <div style={{
                    width: "25%",
                    borderRight: "1px solid grey"
                }}>
                    {data.Email}
                </div>
                <div style={{
                    width: "15%",
                    textAlign: "center",
                    display: "flex",
                    justifyContent: "center"
                }}>
                    <Link to={"/clients/test?id="+data.id} style={{
                        width: "40%",
```

```
                            height: "18px",
                            color: "white",
                            fontSize: "14px",
                            background: "grey",
                            borderColor: "grey",
                            marginRight: "5px",
                            padding: "2.1px 10px",
                            borderRadius: "4px"
                        }}>Open results</Link>
                        <BtnRed rowid={data.id} onClick={removeClient}
name={"Delete"} btnStyle={{
                            width: "30%",
                            height: "100%",
                            color: "white"
                        }}/>
                    </div>
                </>
            ) : (
                type === "test" && isAdmin ? (
                    <>
                        <div style={{
                            width: "10%",
                            borderRight: "1px solid grey"
                        }}>
                            {data.id}
                        </div>
                        <div style={{
                            width: "25%",
                            borderRight: "1px solid grey"
                        }}>
                            {data.name}
                        </div>
                        <div style={{
                            width: "25%",
                            borderRight: "1px solid grey"
                        }}>
                            {data.questions.length}
                        </div>
                        <div style={{
                            width: "25%",
                            borderRight: "1px solid grey"
                        }}>
                            {data.assigns}
                        </div>
                        <div style={{
                            width: "15%",
                            textAlign: "center"
                        }}>
                            <BtnRed rowid={data.id} onClick={togleEdit}
name={"Edit"} btnStyle={{
```

```
                                width: "30%",
                                height: "100%",
                                color: "white",
                                background: "grey",
                                borderColor: "grey",
                                marginRight: "5px"
                            }}/>
                            <BtnRed rowid={data.id} onClick={togleAssign}
name={"Assign"} btnStyle={{
                                width: "30%",
                                height: "100%",
                                color: "white",
                                background: "grey",
                                borderColor: "grey",
                                marginRight: "5px"
                            }}/>
                            <BtnRed rowid={data.id} onClick={removeTest}
name={"Delete"} btnStyle={{
                                width: "30%",
                                height: "100%",
                                color: "white",
                            }}/>
                        </div>
                    </>
                ) : (
                    <>
                        <div style={{
                            width: "10%",
                            borderRight: "1px solid grey"
                        }}>
                            {data.id}
                        </div>
                        <div style={{
                            width: "25%",
                            borderRight: "1px solid grey"
                        }}>
                            {data.name}
                        </div>
                        <div style={{
                            width: "25%",
                            borderRight: "1px solid grey"
                        }}>
                            {data.questions.length}
                        </div>
                        <div style={data.status == "Checked" ? ({
                                width: "25%",
                                borderRight: "1px solid grey",
                                cursor: "pointer"
                            }) : ({
                                width: "25%",
```

```
                                borderRight: "1px solid grey",
                            })}
                            onPointerEnter={onHoverEnter}
                            onPointerLeave={onHoverLeave}
                        >
                            {data.status}
                            <div style={showComment ? (
                                {
                                    display: "block",
                                    position: "ablosute",
                                }
                            ) : (
                                {
                                    display: "none",
                                    position: "ablosute"
                                }
                            )}>
                                <b>Psychologist comments:</b> {data.comments}
                            </div>
                        </div>
                        <div style={{
                            width: "15%",
                            textAlign: "center",
                            display: "flex",
                            justifyContent: "center"
                        }}>
                            {data.status == "To test" ? (
                                <Link to={"/tests/test?id="+data.id} style={{
                                    width: "40%",
                                    height: "18px",
                                    color: "white",
                                    fontSize: "14px",
                                    background: "grey",
                                    borderColor: "grey",
                                    marginRight: "5px",
                                    padding: "2.1px 10px",
                                    borderRadius: "4px"
                                }}>Open test</Link>
                            ) : (
                                <>No actions</>
                            )}
                        </div>
                    </>
                )
            )}
        </div>
    );
};

export default Row;
```

components/row/TableHeader.js

```
import style from "./Row.module.css";

const TableHeader = ({data, type, isAdmin}) => {
  return (
    <div className={style.header}>
        {type === "client" ? (
            <>
                <div style={{
                    width: "10%",
                    fontWeight: "600",
                    borderRight: "1px solid grey"
                }}>
                    Id
                </div>
                <div style={{
                    width: "25%",
                    fontWeight: "600",
                    borderRight: "1px solid grey"
                }}>
                    First name
                </div>
                <div style={{
                    width: "25%",
                    fontWeight: "600",
                    borderRight: "1px solid grey"
                }}>
                    Last name
                </div>
                <div style={{
                    width: "25%",
                    fontWeight: "600",
                    borderRight: "1px solid grey"
                }}>
                    Email
                </div>
                <div style={{
                    fontWeight: "600",
                    width: "15%"
                }}>
                    Actions
                </div>
            </>
        ) : (
            type === "test" ? (
                <>
                    <div style={{
                        width: "10%",
```

```
                        fontWeight: "600",
                        borderRight: "1px solid grey"
                    }}>
                        Id
                    </div>
                    <div style={{
                        width: "25%",
                        fontWeight: "600",
                        borderRight: "1px solid grey"
                    }}>
                        Name
                    </div>
                    <div style={{
                        width: "25%",
                        fontWeight: "600",
                        borderRight: "1px solid grey"
                    }}>
                        Number of q`s
                    </div>
                    <div style={{
                        width: "25%",
                        fontWeight: "600",
                        borderRight: "1px solid grey"
                    }}>
                        {isAdmin ? (
                            <>Number of assigns</>
                        ) : (
                            <>Status</>
                        )}
                    </div>
                    <div style={{
                        fontWeight: "600",
                        width: "15%"
                    }}>
                        Actions
                    </div>
                </>
            ) : (
                <><></>
            )
        )}
    </div>
  );
};

export default TableHeader;
```

components/signIn/SignIn.js

```
import { useEffect, useState } from "react";
```

```
import style from "./SignIn.module.css";
import sprite from "../../img/sprite.svg";
import { isValidEmail } from "../../helpers/helpers";
import { signInOperation } from "../../redux/user/user.operation";
import { useDispatch, useSelector } from "react-redux";
import { getErrorMsg, getLoading } from "../../redux/user/user.selectors";
import BtnBlue from "../buttons/BtnBlue";
import { stylebtnSign } from "./styleBtnSign";
import Container from "../container/Container";
import { signInError } from "../../redux/user/user.actions";
import Spiner from "../spiner/Spiner";
import { useSearchParams, useNavigate  } from "react-router-dom";

const SignIn = () => {
  const [password, setPassword] = useState("");
  const [email, setEmail] = useState("");
  const dispatch = useDispatch();
  const navigate = useNavigate();

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

  const changePassword = (e) => {
    setPassword(e.currentTarget.value);
  }
  const changeEmail = (e) => {
    setEmail(e.currentTarget.value);
  }

  const handleSubmit = (e) => {
    e.preventDefault();
    dispatch(signInOperation({ email, password }));
    setEmail("");
    setPassword("");
  };

  const toSignUp = () => {
    navigate("/sign-up")
  }

  const loading = useSelector(getLoading);

  return (
    <Container>
      {loading ? (
        <div className={style.flexSpiner}>
          <Spiner style={{ color: "black", fontSize: "4em" }} />
        </div>
      ) : (
        <div className={style.authForm}>
          <h2 className={style.header}>Welcome to Psychologist</h2>
```

```
<form onSubmit={handleSubmit} className={style.form}>
  <h3 className={style.formHeader}>Sign into your account</h3>
  <label className={style.label}>
    {" "}
    <span
      style={
        email.length <= 0
          ? { color: "#ff8616" }
          : null
      }
      className={style.nameField}
    >
      Email
    </span>
    <input
      name="email"
      type="email"
      value={email}
      onChange={changeEmail}
      className={style.input}
      style={
        email.length > 0
          ? { border: "1px solid #d3d5da" }
          : { color: "#2b3135", border: "1px solid #ff8616" }
      }
    ></input>
  </label>
  <label className={style.label}>
    {" "}
    <span
      style={
        password.length < 8
          ? { color: "#ff8616" }
          : null
      }
      className={style.nameField}
    >
      Password
    </span>
    <input
      name="password"
      type="password"
      value={password}
      onChange={changePassword}
      style={
        password.length < 8
          ? { border: "1px solid #ff8616" }
          : { color: "#2b3135", border: "1px solid #d3d5da" }
      }
      className={style.input}
```

```
              ></input>
            </label>
            <div className={style.btn}>
              <BtnBlue
                disable={!(password.length > 8)}
                name={"Sign In"}
                iconStyle={{ display: "none" }}
                btnStyle={
                  password.length >= 8 && email.length
                    ? stylebtnSign.active
                    : stylebtnSign.disable
                }
              />
            </div>
          </form>
          <BtnBlue
            onClick={toSignUp}
            name={"Sign Up"}
            iconStyle={{ display: "none" }}
            btnStyle={
                stylebtnSign.activeOutForm
            }
          />
        </div>
      )}
    </Container>
  );
};
export default SignIn;
```

components/signUp/SignUp.js

```
import { useEffect, useState } from "react";
import style from "./SignUp.module.css";
import sprite from "../../img/sprite.svg";
import { isValidEmail } from "../../helpers/helpers";
import { signUpOperation } from "../../redux/user/user.operation";
import { useDispatch, useSelector } from "react-redux";
import { getErrorMsg, getLoading } from "../../redux/user/user.selectors";
import BtnBlue from "../buttons/BtnBlue";
import { stylebtnSign } from "./styleBtnSign";
import Container from "../container/Container";
import { signInError } from "../../redux/user/user.actions";
import Spiner from "../spiner/Spiner";
import { useSearchParams, useNavigate  } from "react-router-dom";

const SignUp = () => {
  const [password, setPassword] = useState("");
  const [email, setEmail] = useState("");
  const [firstName, setFirstName] = useState("");
  const [lastName, setLastName] = useState("");
```

```
const dispatch = useDispatch();
const navigate = useNavigate();

useEffect(() => {

}, [dispatch]);

const changePassword = (e) => {
  setPassword(e.currentTarget.value);
}
const changeEmail = (e) => {
  setEmail(e.currentTarget.value);
}
const changeFirstName = (e) => {
  setFirstName(e.currentTarget.value);
}
const changeLastName = (e) => {
  setLastName(e.currentTarget.value);
}

const toSignIn = () => {
  navigate("/sign-in")
}

const handleSubmit = (e) => {
  e.preventDefault();
  dispatch(signUpOperation({ email, password, firstName, lastName }));
  setEmail("");
  setPassword("");
  setFirstName ("");
  setLastName("");
  navigate("/sign-in")
};

const loading = useSelector(getLoading);

return (
  <Container>
    {loading ? (
      <div className={style.flexSpiner}>
        <Spiner style={{ color: "black", fontSize: "4em" }} />
      </div>
    ) : (
      <div className={style.authForm}>
        <h2 className={style.header}>Welcome to Psychologist</h2>
        <form onSubmit={handleSubmit} className={style.form}>
          <h3 className={style.formHeader}>Sign into your account</h3>
          <label className={style.label}>
            {" "}
            <span
```

```
    style={
      email.length
        ? null
        : { color: "#ff8616" }
    }
    className={style.nameField}
  >
    Email
  </span>
  <input
    name="email"
    type="email"
    value={email}
    onChange={changeEmail}
    className={style.input}
    style={
      email.length
        ? { border: "1px solid #d3d5da" }
        : { color: "#2b3135", border: "1px solid #ff8616" }
    }
  ></input>
</label>
<label className={style.label}>
  {" "}
  <span
    style={
      password.length >= 8
        ? null
        : { color: "#ff8616" }
    }
    className={style.nameField}
  >
    Password
  </span>
  <input
    name="password"
    type="password"
    value={password}
    onChange={changePassword}
    style={
      password.length < 8
        ? { border: "1px solid #ff8616" }
        : { color: "#2b3135", border: "1px solid #d3d5da" }
    }
    className={style.input}
  ></input>
</label>
<label className={style.label}>
  {" "}
  <span
```

```
    style={
      firstName.length
        ? null
        : { color: "#ff8616" }
    }
    className={style.nameField}
  >
    First name
  </span>
  <input
    name="firstName"
    type="firstName"
    value={firstName}
    onChange={changeFirstName}
    style={
      firstName.length
        ? { border: "1px solid #d3d5da" }
        : { color: "#2b3135", border: "1px solid #ff8616" }
    }
    className={style.input}
  ></input>
</label>
<label className={style.label}>
  {" "}
  <span
    style={
      lastName.length
        ? null
        : { color: "#ff8616" }
    }
    className={style.nameField}
  >
    Last name
  </span>
  <input
    name="lastName"
    type="lastName"
    value={lastName}
    onChange={changeLastName}
    style={
      lastName.length
        ? { border: "1px solid #d3d5da" }
        : { color: "#2b3135", border: "1px solid #ff8616" }
    }
    className={style.input}
  ></input>
</label>
<div className={style.btn}>
  <BtnBlue
    disable={!(password.length > 8)}
```

```
                name={"Sign Up"}
                iconStyle={{ display: "none" }}
                btnStyle={
                  password.length >= 8 && email.length && firstName.length &&
lastName.length
                    ? stylebtnSign.active
                    : stylebtnSign.disable
                }
              />
            </div>
          </form>
          <BtnBlue
            onClick={toSignIn}
            name={"Sign In"}
            iconStyle={{ display: "none" }}
            btnStyle={
                stylebtnSign.activeOutForm
            }
          />
        </div>
      )}
    </Container>
  );
};

export default SignUp;
```

components/spinner/Spinner.js

```
import "./Spiner.css";

const Spiner = ({style}) => {
  return (
    <div >
      <div style={style} className="spinner icon-spinner-2" aria-
hidden="true"></div>
    </div>
  );
};

export default Spiner;
```

components/takeTest/ActiveTest.js

```
import { useEffect, useState } from "react";
import { getActiveTestOperation, answerActiveTestOperation } from
"../../redux/user/user.operation";
```

```
import { useDispatch, useSelector } from "react-redux";
import { getLoading, getActiveTest, getUser } from
"../../redux/user/user.selectors";
import { useSearchParams, useNavigate  } from "react-router-dom";
import BtnBlue from "../buttons/BtnBlue";
import style from "./Tests.module.css";
import Container from "../container/Container";
import Spiner from "../spiner/Spiner";

import Header from "../header/Header";

const ActiveTest = () => {
    let [searchParams, setSearchParams] = useSearchParams();
    const test = useSelector(getActiveTest);
    const [answers, setAnswers] = useState({});
    const dispatch = useDispatch();
    const navigate = useNavigate();
    useEffect(() => {
        dispatch(getActiveTestOperation({id: searchParams.get("id")}))
    }, [dispatch]);

    const loading = useSelector(getLoading);
    const user = useSelector(getUser);
    const isAdmin = user.user.Creator == 0 ? true : false

    const handleChange = (e) => {
        answers[e.target.name] = e.target.value
        setAnswers({...answers})
    }

    const handleSubmit = () => {
        const answ = []
        for(let a of Object.entries(answers)){
            answ.push({
                id: a[0],
                answer: a[1]
            })
        }
        dispatch(answerActiveTestOperation({testId: test.id, creatorId:
test.creatorid, answers: answ}))
        onSubmit()
    }

    const onSubmit = () => navigate(`/tests`)

    return (
        <Container>
        {loading ? (
            <div className={style.flexSpiner}>
            <Spiner style={{ color: "black", fontSize: "4em" }} />
```

```
            </div>
        ) : (
            <>
                <Header />
                {test ? (
                    <>
                        <div style={{
                            display: "flex",
                            margin: "10px 10%"
                        }}>
                            <p style={{
                                fontSize: "28px",
                                color: "black",
                                fontWeight: "600",
                                width: "95%",
                                textAlign: "start"
                            }}>
                                {test.name}
                            </p>
                        </div>
                        <div style={{
                            display: "flex",
                            flexDirection: "column",
                            margin: "10px 10% 0 10%"
                        }}>
                            {test.questions.map((q) => (
                                <div key={q.id} style={{
                                    textAlign:"start"
                                }}>
                                    <p style={{
                                        fontSize: "18px",
                                        fontWeight: "600",
                                        color: "black",
                                        paddingBottom: "7px"
                                    }}>
                                        {q.question}
                                    </p>
                                    <div style={{
                                        display: "flex",
                                        flexDirection: "row",
                                        justifyContent: "space-between",
                                        paddingBottom: "15px"
                                    }}
                                    onChange={handleChange}
                                    >
                                        <label>
                                            0
                                            <input type={"radio"} name={q.id}
value={0}/>
                                        </label>
```

```
                                        <label>
                                            1
                                            <input type={"radio"} name={q.id}
value={1}/>
                                        </label>
                                        <label>
                                            2
                                            <input type={"radio"} name={q.id}
value={2}/>
                                        </label>
                                        <label>
                                            3
                                            <input type={"radio"} name={q.id}
value={3}/>
                                        </label>
                                    </div>
                                </div>
                            ))}
                            <div style={{
                                display: "flex",
                                justifyContent: "center",
                                width: "100%"
                            }}>
                                <BtnBlue btnStyle={Object.entries(answers).length
>= test.questions.length ? {
                                        width: "8%",
                                        color: "white",
                                        height: "38px"
                                    } : {
                                        width: "8%",
                                        color: "white",
                                        background: "grey",
                                        cursor: "not-allowed",
                                        borderColor: "grey",
                                        height: "38px"
                                    }} name={"Submit"} onClick={handleSubmit}
disable={Object.entries(answers).length >= test.questions.length ? false : true}/>
                            </div>
                        </div>
                    </>
                ) : (
                    <h1>Test not found</h1>
                )}
            </>
        )}
        </Container>
    );
};

export default ActiveTest;
```

components/tests/Tests.js

```
import { useEffect, useState } from "react";
import { getTestsOperation, getClientsOperation } from
"../../redux/user/user.operation";
import { useDispatch, useSelector } from "react-redux";
import { getLoading, getTests, getUser, getClients } from
"../../redux/user/user.selectors";
import BtnBlue from "../buttons/BtnBlue";
import Row from "../row/Row";
import TableHeader from "../row/TableHeader";
import AddTestsModal from "../modal/AddTestsModal"
import EditTestsModal from "../modal/EditTestsModal"
import AssignTestsModal from "../modal/AssignTestsModal"
import style from "./Tests.module.css";
import Container from "../container/Container";
import Spiner from "../spiner/Spiner";

import Header from "../header/Header";

const Tests = () => {
    const clients = useSelector(getClients);
    const tests = useSelector(getTests);
    const [isOpenAdd, setIsOpenAdd] = useState(false);
    const [isOpenEdit, setIsOpenEdit] = useState(false);
    const [isOpenAssign, setIsOpenAssign] = useState(false);
    const [testToEdit, setTestToEdit] = useState(null);
    const [testToAssign, setTestToAssign] = useState(null);
    const togleAdd = () => setIsOpenAdd(!isOpenAdd);
    const togleEdit = (e) => {
        if(isOpenEdit == true){
            setTestToEdit(null)
        }
        else{
            setTestToEdit(e.currentTarget.getAttribute("rowid"))
        }
        setIsOpenEdit(!isOpenEdit)
    };
    const togleAssign = (e) => {
        if(isOpenAssign == true){
            setTestToAssign(null)
        }
        else{
            setTestToAssign(e.currentTarget.getAttribute("rowid"))
        }
        setIsOpenAssign(!isOpenAssign)
    };
    const dispatch = useDispatch();
```

```
useEffect(() => {
    dispatch(getTestsOperation())
}, [dispatch]);

const loading = useSelector(getLoading);
const user = useSelector(getUser);
const isAdmin = user.user.Creator == 0 ? true : false

return (
    <Container>
    {loading ? (
        <div className={style.flexSpiner}>
        <Spiner style={{ color: "black", fontSize: "4em" }} />
        </div>
    ) : (
        <>
            <Header />
            <div style={{
                display: "flex",
                margin: "10px 10%"
            }}>
                <p style={{
                    fontSize: "28px",
                    color: "black",
                    fontWeight: "600",
                    width: "95%",
                    textAlign: "start"
                }}>{isAdmin ? (
                    <>Tests</>
                ) : (
                    <>Assigned tests</>
                )}</p>
                {isAdmin ? (
                    <BtnBlue onClick={togleAdd} btnStyle={{
                        width: "5%",
                        color: "white",
                        textAlign: "center",
                        height: "40px"
                    }} name={'Add new'}>Add new</BtnBlue>) :
                (
                    <></>
                )}
            </div>
            {tests ? (
                <>
                    <div style={{
                        display: "flex",
                        flexDirection: "column",
                        margin: "10px 10% 0 10%"
                    }}>
```

```
                            <TableHeader isAdmin={isAdmin} data={tests}
type={"test"} />
                        </div><div style={{
                            display: "flex",
                            flexDirection: "column",
                            margin: "0 10%"
                        }}>
                            {tests.map((test) => (
                                <Row isAdmin={isAdmin} togleEdit={togleEdit}
togleAssign={togleAssign} data={test} type={"test"} key={test.id} />
                            ))}
                        </div>
                    </>
                ) : (
                    <h1>No tests yet</h1>
                )}
                {isOpenAdd && (
                    <AddTestsModal
                        name={"Add New Test"}
                        togleModal={togleAdd}
                        display={{ display: "none" }}
                    />
                )}
                {isOpenEdit && (
                    <EditTestsModal
                        data={tests}
                        testId={testToEdit}
                        name={"Edit Test"}
                        togleModal={togleEdit}
                        display={{ display: "none" }}
                    />
                )}
                {isOpenAssign && (
                    <AssignTestsModal
                        testId={testToAssign}
                        clients={clients}
                        name={"Assign Test"}
                        togleModal={togleAssign}
                        display={{ display: "none" }}
                    />
                )}
            </>
        )}
        </Container>
    );
};

export default Tests;
```