Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Інформаційна система для психодіагностики з використанням таблиці Шульте
Назва теми

КвРІСТ 220166.22.01.01 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *126 «Інформаційні системи та технології»*
Шифр, назва

Освітня програма *«Інформаційні системи та технології»*
Назва

Виконав: студент III курсу, група ІСТс-22-1 _____ Віталій БОРТНИК
Підпис Ініціали, прізвище

Керівник _____ Дмитро МЕДЗАТИЙ
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ Тетяна КИСІЛЬ
Підпис, дата Ініціали, прізвище

До захисту допускаю:
зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Ольга ПАВЛОВА
Підпис Ініціали, прізвище

«12» червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень Бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 126 Інформаційні системи та технології

Освітня програма «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ

Зав. кафедри Ольга ПАВЛОВА

" 10 " 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Віталію БОРТНИКУ

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Інформаційна система для психодіагностики з використанням таблиці Шульте

Керівник проекту (роботи) Дмитро МЕДЗАТИЙ, к.т.н., доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 23

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2025 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

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

Огляд досліджуваної проблеми

Проєктування інформаційної системи для психодіагностики з використанням таблиці Шульте

Розробка інформаційної системи для психодіагностики з використанням таблиці Шульте

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Архітектура ПЗ проєкту

Логіка роботи системи

Структури даних

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Тетяна Кисіль, доцент кафедри КІІС		
Антиплагіат	Андрій НІЧЕПОРУК, доцент кафедри КІІС		

7. Дата видачі завдання «_10_»_____01_____2025 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітки
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2025	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2025	виконано
3	Робота над розділом 1 – дослідження предметної області, огляд існуючих рішень, постановка задачі	01.03.2025	виконано
4	Робота над розділом 2 – аналіз та вибір архітектури інформаційної системи, бази даних, стеку технологій та проєктування системи	01.04.2025	виконано
5	Робота над розділом 3 – розробка бази даних, програмна реалізація серверної частини та вебзастосунку, тестування системи	29.04.2025	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2025	виконано
7	Попередній захист ВКР	26.05.2025	виконано
8	Захист ВКР на засіданні ЕК	Червень 2025 року	

Студент _____ Віталій БОРТНИК
Підпис Ініціали, прізвище

Керівник роботи _____ Дмитро МЕДЗАТИЙ
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРІСТ 220166.22.01.01 ПЗ	Пояснювальна записка	72		
			Графічні матеріали			
2		КвРІСТ 220166.22.01.01 Е8	Архітектура ПЗ проєкту	1		
3		КвРІСТ 220166.22.01.01 Е8	Логіка роботи системи	1		
4		КвРІСТ 220166.22.01.01 Е8	Структури даних	1		

КвРІСТ 220166.22.01.01 ВП

Зм	Арк	№ докум	Підпис	Дата
Розробив		Бортник		12.06.25
Перевір.		Медзатий		12.06.25
Н. контр.		Кисіль		12.06.25
Затв.		Павлова		12.06.25

Відомість проекту

Літера	Аркуш	Аркушів
У	1	1

ХНУ, ІСТс-22-1

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Інформаційна система для психодіагностики з використанням таблиці Шульте».

Автор роботи: Віталій БОРТНИК.

Керівник роботи: Медзатий Дмитро Миколайович.

Пояснювальна записка: 72 с., 50 рис., 0 табл., 4 дод., 51 джерел.

Графічна частина: 3 креслення.

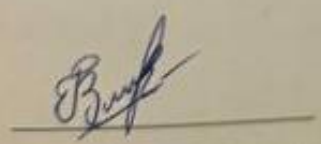
ПСИХОДІАГНОСТИКА, ТЕСТУВАННЯ ІНФОРМАЦІЙНА СИСТЕМА, АРХІТЕКТУРА, ВЕБЗАСТОСУНОК, БАЗА ДАНИХ.

Метою кваліфікаційної роботи є проєктування та розробка інформаційної системи, що дозволяє проводити психодіагностичне тестування з використанням таблиці Шульте у вебсередовищі. Система має забезпечувати можливість налаштування параметрів тесту, фіксацію результатів, збереження статистики та надання інтерпретації даних у зручному форматі.

Об'єктом дослідження є процес комп'ютеризованої психодіагностики за допомогою таблиці Шульте.

Предметом дослідження виступають принципи побудови інформаційної системи, засоби обробки результатів психодіагностики та особливості інтерактивної реалізації методики Шульте у вебдодатку.

Під час розробки були використані методи системного аналізу, проектування програмних систем, порівняльний аналіз архітектурних рішень, а також методи оцінки ефективності вебінтерфейсів і баз даних. Дослідження базувалося на огляді наукових публікацій, технічної документації, а також сучасних практик реалізації подібних систем.



Підпис студента

30.05.2025

Дата

ЗМІСТ

ВСТУП

У сучасному світі багато речей стали модернізованими в результаті появи інтернету та технологій. Інформаційні технології стали невід’ємною частиною повсякденного життя. З кожним днем все більше і більше процесів автоматизуються. Важливу роль в автоматизації відіграють інформаційні системи, які використовуються практично в усіх сферах від побуту до професійної діяльності.

Однією з сфер яка потребує автоматизації є психологія, зокрема – психодіагностика. Даний напрям вимагає швидкості, точності і зручності в обробці результатів тестування. Саме тому все більшої актуальності набуває використання спеціалізованих програмних засобів для проведення психодіагностичних досліджень. Застосування таких рішень дозволяє оптимізувати процес тестування та забезпечити об'єктивність результатів, а також детальний аналіз динаміки змін когнітивних навичок протягом певного періоду часу.

Популярним методом оцінки швидкості реакції та уваги є проста, але водночас ефективна методика, яку активно використовують у навчанні, спорті, медицині, а також для саморозвитку – тестування за допомогою таблиці Шульте. Проходження такого тестування у звичайній формі може бути менш зручним, а обробка результатів часто вимагає додаткового часу та зусиль. Ось тому і виникає потреба у зручному інструменті, який спростить сам процес тестування – автоматично зафіксує результати, надасть детальну статистику, дозволить відстежувати прогрес, а для подальшого аналізу дозволить експортувати дані.

Отже, даний дипломний проєкт є вебзастосунком який реалізує функціональну та зручну інформаційну систему для психодіагностики з використанням таблиці Шульте, яка дозволить проводити тестування в інтерактивному режимі, а також забезпечить збереження, перегляд та аналіз результатів.

1 ОГЛЯД ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

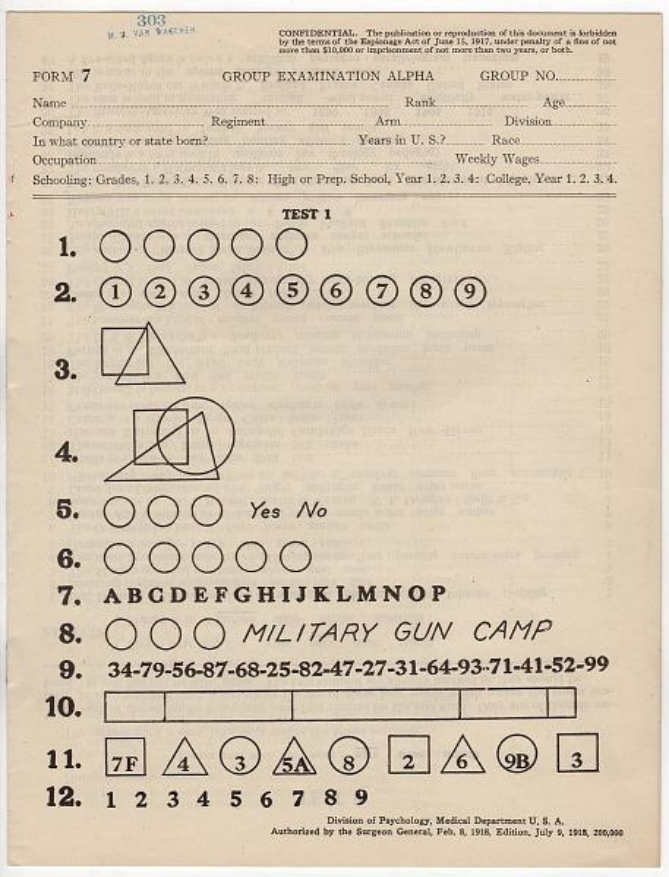
1.1 Дослідження предметної області, розгляд основних завдань та проблем

Психодіагностика як розділ прикладної психології має досить тривалу історію, адже потреба у вимірюванні психічних процесів, таких як увага, пам’ять, реакція, виникла ще з появою перших професій, що вимагали підвищеної точності та швидкості дій. У перших лабораторіях психології в Європі та США вже у XIX столітті активно вивчали швидкість реакції, точність сприймання та межі пам’яті. Франсіс Гальтон, якому приписують заснування психометрії, вимірював та оцінював сенсорні здібності людини, вважаючи їх показниками інтелекту [1]. Такі підходи на сьогоднішній день виглядають доволі дивно, проте саме вони дали поштовх створенню перших тестових методик. Приклад тогочасної лабораторії зараз можна вільно знайти в інтернеті (рис. 1.1).

Francis Galton's First Anthropometric Laboratory at the International Health Exhibition, South Kensington, 1884–5.

Рисунок 1.1 – Антропометрична лабораторія Гальтона на Міжнародній виставці здоров’я [2]

У далекому 1917 році перед американською армією стояв серйозний виклик – як швидко оцінити інтелект і здібності мільйонів новобранців. Рішенням став Army Alpha Test (рис. 1.2), один з перших масштабних тестів на інтелект, розроблений психологом Робертом Йеркесом та командою експертів [3].


303

CONFIDENTIAL. The publication or reproduction of this document is forbidden by the terms of the Espionage Act of June 15, 1917, under penalty of a fine of not more than $10,000 or imprisonment of not more than two years, or both.

FORM 7 GROUP EXAMINATION ALPHA GROUP NO.

Name Rank Age

Company Regiment Arm Division

In what country or state born? Years in U. S.? Race

Occupation Weekly Wages

Schooling: Grades, 1. 2. 3. 4. 5. 6. 7. 8: High or Prep. School, Year 1. 2. 3. 4: College, Year 1. 2. 3. 4.

TEST 1

1. ○ ○ ○ ○ ○

2. ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

3.

4.

5. ○ ○ ○ Yes No

6. ○ ○ ○ ○ ○

7. A B C D E F G H I J K L M N O P

8. ○ ○ ○ MILITARY GUN CAMP

9. 34-79-56-87-68-25-82-47-27-31-64-93-71-41-52-99

10.

11. 7F △4 ③ △5A ⑧ □2 △6 ⑨B □3

12. 1 2 3 4 5 6 7 8 9

Division of Psychology, Medical Department U. S. A.
Authorized by the Surgeon General, Feb. 8, 1918. Edition, July 9, 1918, 200,000

Рисунок 1.2 – Вигляд Army Alpha Test [4]

Він оцінював вербальні здібності, логічне мислення, арифметичні навички, здатність до аналізу та розуміння інструкцій. Army Alpha став проривом у прикладній психології, адже дозволяв протестувати велику кількість людей одночасно, а також започаткував використання психометричних методик у державних структурах. Згодом цей підхід ліг в основу подальших військових і цивільних тестових систем у всьому світі. Також кілька років до цього у Європі активізувалося дослідження нейропсихологічних відхилень у дітей – наприклад, за допомогою тесту Біне-Сімона [5].

Зважаючи на вищесказане, можна зробити висновок, що ще до винайдення комп'ютерів виникла ціла галузь, яка потребувала точних, стандартизованих, але простих у використанні засобів перевірки уваги – що й стало фундаментом для появи таблиці Шульте.

Метод Шульте був розроблений німецьким психіатром і психотерапевтом Вальтером Шульте в середині 20-го століття. Спочатку використовувався в інженерній психології для вимірювання ефективності та швидкості орієнтаційно-пошукових рухів очей операторів. З того часу таблиця зазнала численних змін і доповнень. Базова таблиця Шульте складається з 25 клітинок (5 по вертикалі і 5 по горизонталі), випадковим чином заповнених числами від 1 до 25 (рис. 1.3). Завдання полягає в тому, щоб якомога швидше знайти і вказати числа в порядку зростання або спадання [6].

1	**8**	**3**	**23**	**11**
14	**24**	**12**	**21**	**20**
19	**10**	**5**	**25**	**17**
4	**13**	**15**	**7**	**6**
16	**2**	**18**	**22**	**9**

Рисунок 1.3 – Вигляд класичної Таблиці Шульте [7]

Попри простоту, таблиця виявилася надзвичайно ефективною. Пізніше її стали використовувати для тренування уваги, а також як інструмент нейропсихологічної корекції. В умовах стрімкого розвитку цифрових технологій, коли інформаційне навантаження на людину зростає з кожним роком, ця методика не втрачає актуальності.

Ці таблиці застосовувати для відбору водіїв, пілотів, диспетчерів та інших професій, що потребують високої концентрації [8]. У той час таблиці виготовляли вручну або на друкарських машинках, а час проходження фіксували секундоміром.

Згодом, із поширенням персональних комп'ютерів, таблиця почала використовуватись у вигляді простих додатків, проте більшість із них були обмежені базовим функціоналом такими як генерація таблиці і фіксація часу проходження (рис. 1.4). Повноцінний психодіагностичний підхід залишився недоступним через відсутність автоматизованого збору статистики, гнучких налаштувань і можливості зручної візуалізації результатів.

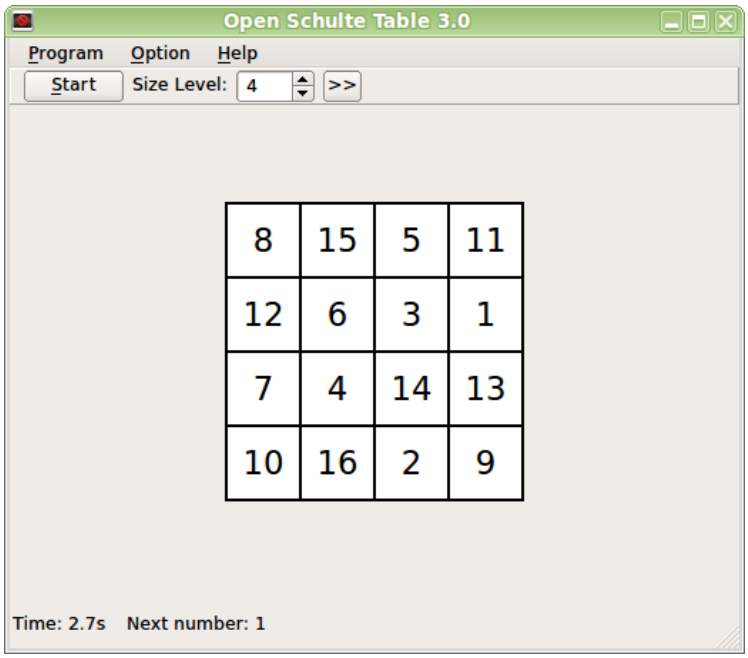


Рисунок 1.4 – Вигляд реалізації Таблиці Шульте у вигляді програми для персонального комп'ютера [9]

У наш час існує нагальна потреба у цифрових інструментах, які могли б поєднувати наукову точність, зручність користування та розширені можливості аналітики. Особливо актуально це у сфері шкільної освіти, де цифрові психодіагностичні засоби можуть допомагати педагогам та психологам своєчасно виявляти труднощі з концентрацією уваги в учнів, що часто є ранніми проявами когнітивних або емоційних проблем.

У сфері підготовки водіїв сучасні інструменти дають змогу оцінити загальний час реакції, що є критично важливим для безпечного керування транспортом [10].

Спортивна психологія, у свою чергу, вимагає оцінки здатності спортсмена утримувати увагу в умовах високого психоемоційного навантаження – і саме цифрові методики дозволяють це робити об’єктивно, фіксуючи найменші зміни в динаміці уваги [11].

Не менш важливою є роль цифрових тестових систем у медичній практиці, зокрема при реабілітації пацієнтів після черепно-мозкових травм або інсультів, коли лікарі повинні слідкувати за відновленням когнітивних функцій. У таких випадках наявність інструменту, що не лише реєструє час, а й зберігає, аналізує та візуалізує результати, значно покращує якість спостереження та ефективність терапії [12].

Проте, як і в багатьох інших психодіагностичних методиках, традиційне виконання тесту Шульте має низку суттєвих недоліків, які значною мірою впливають на такі показники як точність, об’єктивність, масштабованість результатів.

Однією з головних проблем є неточність вимірювання часу. Навіть досвідчений фахівець не може з високою точністю зафіксувати, скільки саме часу людина витратила на пошук кожного конкретного числа, особливо коли фіксація здійснюється за допомогою ручного секундоміра. Це створює потенційні похибки, які знижують достовірність отриманих результатів.

Ще однією проблемою є відсутність аналізу динаміки. Традиційний підхід з використанням паперової версії чи примітивної цифрової реалізації, не дає змоги відстежити прогрес користувача у за певний проміжок часу. Без автоматизованого збереження результатів стає не так зручно визначати, чи дійсно відбувається покращення уваги та чи зменшується кількість помилок із кожною наступною спробою. Окрім цього, традиційні реалізації часто не дають таких гнучких налаштувань як програмна реалізація. У більшості випадків користувач обмежений

фіксованим розміром таблиці, та не може встановлювати різні таймери, змінювати режими тестування чи визначати допустиму кількість помилок.

Однією з серйозних проблем є ігнорування помилок. У традиційній версії тесту користувач може помилково натиснути не те число або пропустити потрібне і ці дії зазвичай не фіксуються. Така проблема знижує діагностичну цінність тесту, адже саме кількість і тип помилок є важливими маркерами уваги.

В решті решт, навіть якщо час проходження фіксується, він найчастіше подається у вигляді звичайного числового значення без подальшої візуалізації. Для ефективної роботи з користувачем який може бути учнем, спортсменом або пацієнтом – надзвичайно важливо мати візуалізацію змін. Відсутність графіків, діаграм за певний період часу ускладнює інтерпретацію результатів і знецінює ефективний інструмент.

Тому в сучасних умовах виникає нагальна потреба в розробці повноцінної інформаційної системи, що реалізує тест Шульте з усіма відповідними функціями: від генерації таблиць до аналізу та експорту результатів.

1.2 Аналітичний огляд існуючих рішень

Перед початком реалізації будь-якого програмного проєкту важливо не лише визначити основну ідею та цільову аудиторію, а й проаналізувати вже існуючі рішення на ринку. Це стосується як готових програмних продуктів зі схожим функціоналом, так і підходів до технічної реалізації. Аналіз аналогів дозволяє краще зрозуміти, які функції користувачі вважають корисними, що слід удосконалити, а також які помилки минулих рішень не варто повторювати. Крім того, це допомагає визначити переваги майбутнього застосунку, а також обрати найбільш доцільний формат реалізації.

Наразі існує кілька основних типів програмного забезпечення, за допомогою яких можна реалізувати ту чи іншу інформаційну систему: десктопні додатки, мобільні додатки та вебзастосунки.

Кожен із цих варіантів має свої особливості, переваги та недоліки. Вибір між ними залежить від того, яким буде основне середовище використання застосунку, які ресурси доступні, наскільки критичною є мобільність, швидкість роботи, потреба в збереженні даних локально чи в хмарі, та багато інших чинників.

Десктопний додаток – це програмне забезпечення, яке встановлюється безпосередньо на комп'ютер користувача. Такі додатки зазвичай орієнтовані на складні та завдання які потребують чимало ресурсів, завдання де важлива висока продуктивність, стабільність роботи, а також можливість повноцінної взаємодії з файловою системою та іншими локальними ресурсами комп'ютера. Серед основних переваг такого типу програм можна назвати швидкодію, яка досягається за рахунок локального виконання всіх операцій, незалежність від інтернет-з'єднання, глибока інтеграція з ОС, можливість працювати з апаратним забезпеченням пристрою [13]. Проте є й недоліки, серед яких те, що користувач завжди прив'язаний до одного пристрою, а для запуску на різних операційних системах, до прикладу, Windows, macOS або Linux, потрібна окрема адаптація або використання універсальних середовищ, що не завжди зручно. Крім того, оновлення додатку вимагає дій з боку самого користувача, по типу завантаження, встановлення чи оновлення, що також може бути незручно.

Мобільний додаток – це застосунок, який розробляється для встановлення на смартфони або планшети. У зв'язку з глобальним поширенням мобільних пристроїв та активним використанням їх у повсякденному житті, розробка мобільних додатків стала особливо популярною. Вони дають змогу користувачу постійно мати під рукою потрібний функціонал починаючи з більш складних операцій автоматизації, закінчуючи звичайними засобами для зв'язку. Важливою перевагою є можливість використання вбудованих апаратних компонентів пристрою – таких як камера, мікрофон, GPS, гіроскоп, біометричні сенсори. Це дозволяє створювати інтуїтивні й інтерактивні програми, що реагують на рухи, голос або місцезнаходження. Розробка мобільних додатків має і свої складнощі. Перш за все, це необхідність адаптації або створення окремих версій під Android та

iOS, адже ці платформи відрізняються за архітектурою, вимогами до дизайну, технічними обмеженнями [14]. Навіть використання кросплатформених фреймворків, не завжди дає змогу повністю уникнути цих труднощів. Також слід враховувати, що мобільні додатки зазвичай мають обмеження у потужності та розмірі екрана, а складний функціонал може уповільнювати роботу програми.

Вебзастосунок – це ще один популярний формат, який має багато переваг, особливо з точки зору універсальності та доступності. Такі застосунки відкриваються в браузері та не потребують встановлення [15]. Це означає, що ними можна користуватись з будь-якого пристрою – ноутбука, ПК, смартфона чи планшета – незалежно від операційної системи. Користувач завжди має доступ до останньої версії, оскільки оновлення відбуваються централізовано на сервері, без необхідності оновлювати клієнтську частину вручну. Це спрощує як підтримку, так і розвиток проєкту. Вебзастосунки можуть ефективно виконувати багато задач, від обміну повідомленнями до складних обчислень у хмарі. Проте вони мають і певні обмеження. Найважливіше з них, це залежність від інтернет-з'єднання. У більшості випадків без доступу до мережі вебдодаток втрачає свою функціональність. Також, на відміну від нативних рішень, вебзастосунки мають обмежений доступ до системних ресурсів пристрою, наприклад, не можуть повноцінно працювати з файловою системою, камерою або іншими локальними модулями без додаткових дозволів. Попри всі недоліки, для реалізації даного проєкту, формат вебзастосунку може бути оптимальним рішенням.

Одним рішень у сфері тренування швидкочитання є мобільний додаток «Таблиці Шульте – Швидкочитання» від Yurkap, розроблений спеціально для Android (рис. 1.5). Ця програма є мобільною версією однойменного вебсервісу й надає користувачеві зручний спосіб працювати з таблицями Шульте прямо на телефоні. Завдяки простому інтерфейсу, україномовній локалізації та регулярним оновленням, застосунок набув великої популярності серед користувачів, про що свідчить високий рейтинг у Google Play – 4.9 із 5 на основі понад 1300 відгуків і понад 50 тисяч завантажень [16].

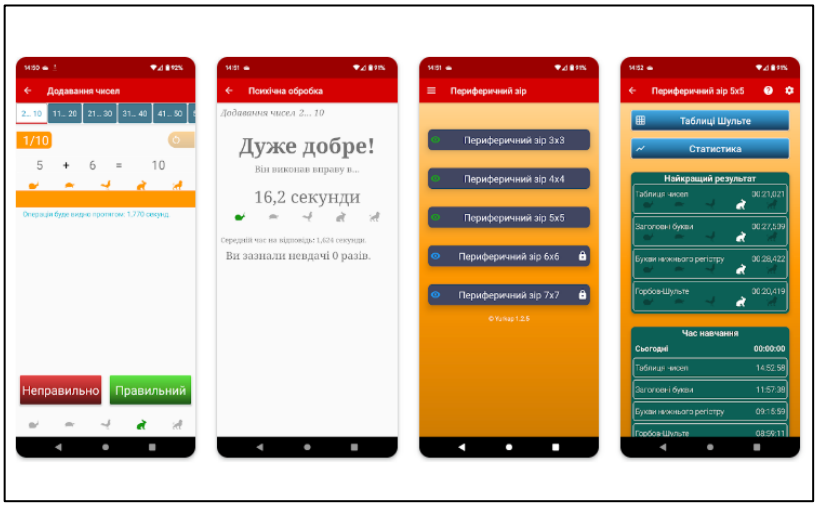


Рисунок 1.5 – Вигляд інтерфейсу «Таблиці Шульте – Швидкочитання» в GooglePlay [16]

Функціонально додаток досить насичений. Його головний режим – це класичні таблиці Шульте різного розміру, починаючи зі стандартних 5×5 і до складніших варіантів, наприклад, 6×6. Цікаво, що тут можна працювати не лише з цифрами, а й з літерами українського алфавіту, що вигідно вирізняє цей додаток серед аналогів, більшість з яких не підтримують кирилицю. Завдання лишається класичним – знайти всі елементи в правильному порядку якомога швидше.

Усі результати зберігаються – додаток веде статистику успішності: можна переглянути кращі результати, середній час проходження та динаміку прогресу за певний період.

Програма абсолютно безкоштовна в базовому функціоналі, відсутня реклама, яка часто відволікає у додатках.

Попри всі плюси, є кілька моментів, які варто враховувати. Додаток офіційно доступний лише для Android, тож користувачі iPhone не мають змоги скористатися ним безпосередньо. Також, незважаючи на наявність кількох вправ, додаток зосереджений виключно на навичках, пов’язаних із читанням.

Schulte Table Online від BrainApps.io. Цей інструмент пропонує користувачам можливість працювати з класичними таблицями Шульте без необхідності встановлення додаткового програмного забезпечення (рис. 1.6).

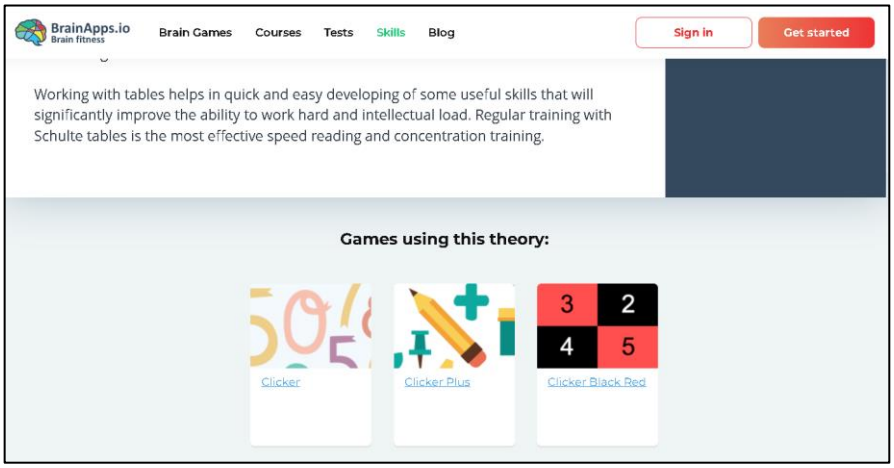


Рисунок 1.6 – Вигляд вебзастосунку Schulte Table Online [6]

Основна мета цього вебзастосунку – розвиток периферійного зору, концентрації та швидкості обробки інформації. Користувачі можуть обирати з трьох різних ігор, в яких різні розміри сітки, що дозволяє адаптувати складність вправи під свій рівень підготовки [6].

Але в цього вебзастосунку є кілька недоліків серед яких відсутність функції збереження результатів, тобто результати не зберігаються між сесіями, отже для довгострокового відстеження прогресу користувачеві доведеться фіксувати дані самостійно.Також значним мінусом є неможливість проходження тестувань без авторизації.

Open Schulte Table – десктопний застосунок який є безкоштовним та з відкритим кодом, він доступний як на Windows так і Linux (рис. 1.7).

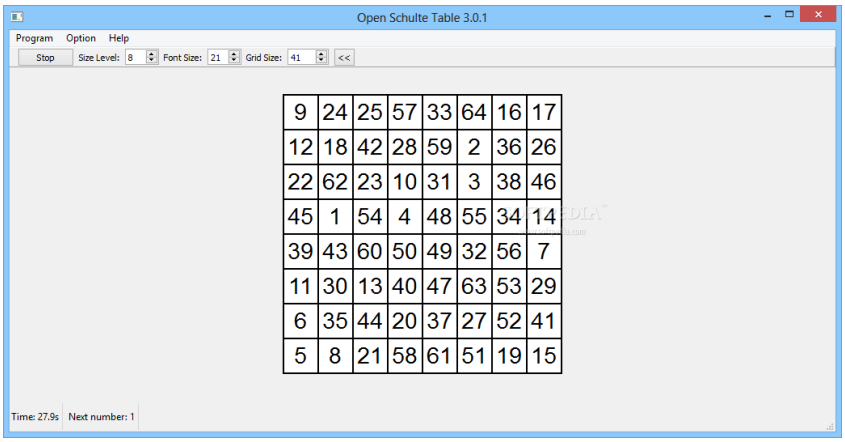


Рисунок 1.7 – Вікно десктопної програми Open Schulte Table [17]

Даний програмний продукт реалізує класичну методику таблиць Шульте, де користувачеві потрібно знаходити числа в порядку зростання, фокусуючи погляд на центрі таблиці. Програма також підтримує різні розміри сітки, що дозволяє адаптувати складність вправ до рівня користувача. Інтерфейс побудований на бібліотеці Qt, що забезпечує кросплатформену сумісність [9].

Превагами є простота використання та можливість запуску на різних платформах. Однак, варто зазначити, що інтерфейс програми досить базовий, та не містить додаткових функцій, таких як збереження статистики, різноманітні режими гри чи інтеграція з іншими сервісами.

1.3 Функціональні та нефункціональні вимоги до системи

Проаналізувавши предметну область, способи реалізації та існуючі аналоги – варто скласти вимоги до даного програмного забезпечення. Зазвичай вимоги поділяються на дві основні категорії: функціональні та нефункціональні. Цей поділ дозволяє структуровано описати, що саме повинна робити система та якими характеристиками вона повинна володіти [18].

Функціональні вимоги визначають конкретні функції та поведінку системи. Вони описують, які дії повинна виконувати система у відповідь на певні вхідні дані або події. Наприклад, можливість користувача увійти до системи, створити новий запис або отримати звіт. Ці вимоги формують основу функціональності програмного продукту та безпосередньо впливають на його архітектуру [19].

Нефункціональні вимоги, з іншого боку, описують загальні властивості системи, такі як продуктивність, надійність, безпека, масштабованість та зручність використання. Вони встановлюють критерії, за якими оцінюється якість роботи системи, але не визначають конкретні функції. Наприклад, вимога, щоб система обробляла запити не довше ніж за 2 секунди [20].

Загалом розмежування функціональних та нефункціональних вимог є доволі важливим для успішної реалізації програмного продукту. Такий підхід дозволяє

створити програмне забезпечення, яке не лише виконує потрібні завдання, але й робить це надійно, а головне швидко та зручно для користувача.

Функціональність майбутнього вебзастосунку має охоплювати повний цикл роботи з таблицею Шульте. Система повинна вміти генерувати таблиці заданого розміру з випадковим розташуванням чисел. Користувач перед початком проходження тесту має змогу обрати ключові параметри: розмір самої таблиці, режим проходження, тобто, чи потрібно шукати числа у порядку зростання чи спадання, а також встановити обмеження – максимальний час на пошук одного числа, максимальний час на проходження таблиці та допустиму кількість помилок.

Після початку тестування система повинна відображати таблицю й фіксувати кожну дію користувача. Для кожного натискання має фіксуватись час реакції, а також чи була дія правильною. Якщо користувач помиляється певну кількість разів або перевищує час, відведений на пошук елемента, тест має завершитись автоматично.

Для зареєстрованих користувачів система повинна зберігати кожну спробу до бази даних. Крім основних результатів – таких як загальний час тесту, кількість правильних і неправильних натискань, причина завершення, до бази зберігаються також усі параметри, з якими тест проводився, і час на кожен крок.

Окремою частиною функціональності є блок аналітики. Він повинен реалізовувати побудову графіків, які демонструють зміну часу проходження, середнього часу на кожне число, кількості помилок та інші метрики. Користувач повинен мати змогу бачити, як змінюються його результати з кожною новою спробою або за певний період. Також потрібно передбачити можливість експорту результатів у зручному для користувача форматі.

Також варто розглянути нефункціональні вимоги.

По-перше, інтерфейс застосунку має бути зручним, сучасним і адаптованим до широкого кола користувачів. Оскільки серед цільової аудиторії передбачаються різні користувачі, такі як педагоги, психологи, студенти або просто особи, зацікавлені у розвитку власних когнітивних здібностей, особливу увагу потрібно

акцентувати на простоті навігації, зрозумілості інструкцій і візуальній структурі елементів. Рекомендовано використовувати принципи мінімалістичного дизайну та зручної кольорової гами, аби уникати візуального перевантаження.

По-друге, система повинна бути високопродуктивною. У процесі проходження тесту особливо важливе значення має точність вимірювання часу, що потребує максимально швидкої реакції системи на дії користувача. Будь-яке сповільнення у відображенні натискання або затримка у фіксації реакції може призвести до викривлення результатів, а отже й до зниження достовірності психодіагностичних висновків.

Також важливою є оптимізація бази даних, що забезпечує збереження результатів тестування у реальному часі. Система повинна бути здатною обробляти запити навіть при одночасному використанні великою кількістю користувачів. Це особливо актуально, якщо розглядати можливість масштабування інормаційної системи у майбутньому.

Важливим є аспект безпеки, тому всі дані мають передаватися через захищений протокол. Усі дані, що зберігаються в базі, мають бути недоступними для сторонніх осіб, навіть у разі несанкціонованого доступу до сервера.

Оскільки вебзастосунок має працювати у браузері, обов’язковою є сумісність із основними сучасними браузерами: Google Chrome, Mozilla Firefox, Microsoft Edge, Safari тощо. Бажано також протестувати відображення та роботу системи на різних платформах і роздільних здатностях екранів, аби гарантувати кросплатформенність та адаптивність інтерфейсу.

1.4 Постановка задачі

Основним завданням даного дипломного проєкту є розробка повнофункціонального вебзастосунку для проведення психодіагностичного тестування на основі методики таблиці Шульте. Метою цього вебзастосунку є надання користувачеві можливості в інтерактивному форматі досліджувати та

покращувати рівень концентрації уваги, швидкість реакції та інші когнітивні навички.

Система, що розробляється має охоплювати повний цикл організації тестування, починаючи від автоматичної генерації таблиці з випадковим розташуванням чисел, налаштування параметрів таких як режим проходження, час на знаходження числа, кількість помилок тощо, і завершуючи детальним аналізом отриманих результатів. У процесі тестування система повинна фіксувати кожен крок користувача, вимірюючи точний час на пошук кожного елемента, а також реєструвати помилки. Такий підхід дозволить не лише оцінити загальний рівень уваги, а й простежити зміну показників у динаміці.

Особливістю реалізації є передбачення двох режимів взаємодії з системою: гостьовий, що працюватиме без збереження історії результатів та режим зареєстрованого користувача. В режимі зареєстрованого користувача є можливість зберігати кожну спробу у базі даних, будувати графіки змін, експортувати звіти в зручних форматах, а також проводити моніторинг прогресу.

Окрім основної психодіагностичної функції, вебзастосунок виконує роль тренажера для розвитку когнітивних навичок. Завдяки регулярному проходженню тестів користувач може покращити швидкість обробки інформації, стійкість уваги та здатність до самоконтролю.

Отже, дана інформаційна система може стати чимось середнім між надто спрощеними додатками без збереження даних та складними професійними інструментами, які не є доступними широкому колу користувачів. Її створення дозволить реалізувати сучасний, інтерактивний та інформативний підхід до психодіагностики в цифровому середовищі.

1.5 Висновки до першого розділу

У межах першого розділу було здійснено комплексний огляд предметної області, пов’язаної з використанням таблиці Шульте в психодіагностиці, а також

проаналізовано специфіку цифрової реалізації подібних систем. Розглянуто історичні передумови появи методики, її застосування в різних галузях – від медицини до освіти та спорту.

Було також проведено огляд існуючих програмних рішень, як мобільних, десктопних, так і вебзастосунків, із зазначенням їх переваг і недоліків. На основі цього аналізу стало можливим визначити основні функціональні й нефункціональні вимоги до майбутньої інформаційної системи. Крім того, було чітко сформульовано задачу, яку має вирішувати програмний продукт.

Отже, перший розділ заклав теоретичну і практичну основу для подальшої розробки інформаційної системи у вигляді вебзастосунку.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПСИХОДІАГНОСТИКИ З ВИКОРИСТАННЯМ ТАБЛИЦІ ШУЛЬТЕ

2.1 Аналіз та вибір архітектури інформаційної системи

На етапі проєктування інформаційної системи одним із ключових завдань є вибір оптимальної архітектури проєкту. Архітектура визначає спосіб організації логіки застосунку, структуру взаємодії між компонентами системи, принципи збереження та обробки даних, а також впливає на масштабованість, продуктивність і зручність подальшої підтримки [21].

Тому доцільним є порівняння кількох варіантів архітектури, які можуть бути використані для побудови цієї та подібних інформаційних систем. Серед таких архітектурних моделей:

- монолітна архітектура;
- мікросервісна архітектура;
- клієнт-серверна архітектура.

Кожен варіант буде варто проаналізувати з точки зору переваг, недоліків і відповідності вимогам даного проєкту.

Монолітна архітектура – це традиційний підхід до розробки програмного забезпечення, при якому всі компоненти додатку такі як інтерфейс користувача, бізнес-логіка, доступ до даних, інтегруються в єдину кодову базу та розгортаються як одне ціле [22]. Такий підхід був домінуючим протягом тривалого часу, особливо для невеликих або середніх проєктів, де простота розробки та розгортання є пріоритетом.

Серед основних переваг монолітної архітектури варто виокремити її простоту в розробці та розгортанні. Оскільки всі компоненти системи розміщені в єдиній кодовій базі, це значно полегшує розуміння структури додатку й дає змогу швидко створювати один виконуваний файл або артефакт для розгортання [23]. Такий підхід є особливо зручним на ранніх етапах розробки або для невеликих

команд. Окрім цього, централізована структура сприяє швидкому налагодженню та тестуванню – потік запитів легко відстежується, а локалізація проблем займає менше часу. Рішення з використанням монолітної архітектури, як правило, вимагають менше інфраструктурних ресурсів для обслуговування, що робить їх економічно доцільними для стартапів і малих проєктів.

Проте монолітна архітектура має і недоліки. Одним із головних недоліків є складність масштабування окремих частин системи, тобто якщо певний компонент потребує додаткових ресурсів, доводиться масштабувати весь застосунок, що є неефективним з точки зору використання обчислювальних потужностей. Крім того, через тісну інтеграцію компонентів виникають труднощі з упровадженням нових технологій або оновленням наявних. Будь-яка зміна може потребувати серйозних витрат часу й ризикує вплинути на стабільність усієї системи. З плином часу монолітні рішення втрачають гнучкість і стають дедалі складнішими в обслуговуванні: внесення нових функцій або оновлень перетворюється на ризикований та трудомісткий процес. До того ж, у разі критичної помилки в одному з компонентів може постраждати весь застосунок, оскільки всі частини тісно пов’язані між собою [24].

Зважаючи на ці всі аспекти, можна зробити висновок, що архітектура підходить для невеликих або середніх проєктів, де простота розробки, розгортання та обслуговування є ключовими факторами. Однак, зі зростанням складності системи, її обмеження стають більш помітними, що може ускладнити масштабування, впровадження нових технологій та підтримку. У контексті розробки інформаційної системи для психодіагностики з використанням таблиці Шульте, яка передбачає взаємодію з інтерфейсом у реальному часі, збереження результатів та потенційне розширення функціоналу, монолітна архітектура може бути менш придатною порівняно з більш гнучкими та масштабованими архітектурними підходами.

Для кращого розуміння функціонування системи з використаннями монолітної архітектури варто розглянути її схему (рис. 2.1).

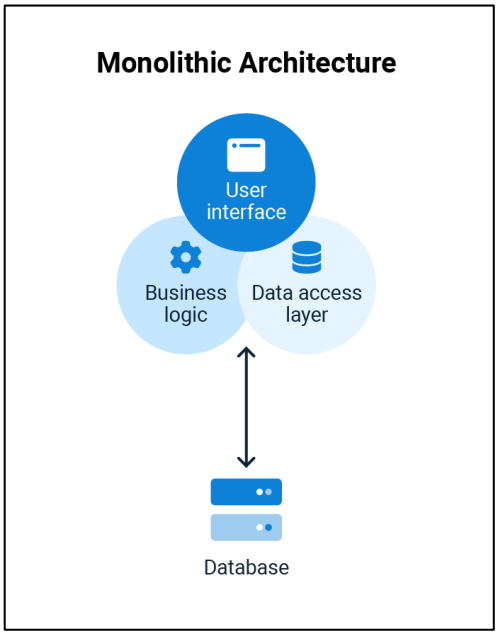


Рисунок 2.1 – Схема монолітної архітектури [25]

Мікросервісна архітектура – це підхід до розробки програмного забезпечення, за якого система поділяється на сукупність окремих сервісів, кожен з яких виконує певну ізольовану бізнес-функцію. Такі сервіси працюють автономно, взаємодіють між собою через чітко визначені програмні інтерфейси та є незалежними один від одного [26]. Кожен мікросервіс зазвичай має власну базу даних, що сприяє ізоляції даних і підвищує стійкість системи до збоїв (рис. 2.2).

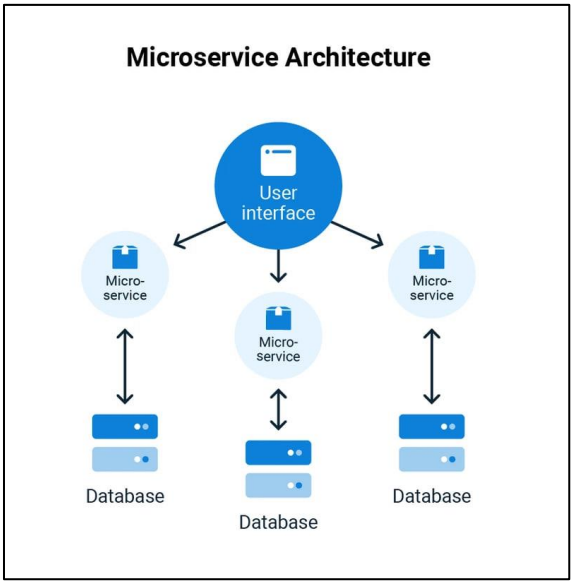


Рисунок 2.2 – Схема мікросервісної архітектури [25]

Цей підхід має чимало переваг, які роблять його привабливим варіантом для сучасної розробки складних систем. Однією з ключових переваг є можливість масштабування кожного мікросервісу окремо, що дозволяє ефективно розподіляти ресурси відповідно до реального навантаження на конкретні частини системи. Крім того, незалежність сервісів дозволяє розгортати й оновлювати окремі модулі без потреби зупиняти або змінювати всю систему, що значно пришвидшує цикл розробки та впровадження змін. Надійність такої архітектури також зростає, адже збій одного компонента не тягне за собою зупинку всієї системи, що підвищує її загальну доступність. Ще однією перевагою є можливість використання різних мов програмування та технологій для реалізації окремих сервісів, що забезпечує гнучкість у виборі інструментів. Також поділ системи на невеликі логічні частини дає змогу організовувати роботу команди за принципом відповідальності, де кожна команда може зосередитися на одному сервісі, що сприяє підвищенню продуктивності [27].

Окрім цього мікросервісна архітектура має і деякі недоліки. Управління великою кількістю сервісів потребує чіткої організації, налагоджених механізмів взаємодії та систем синхронізації даних, що ускладнює процес розробки й підтримки. Під час тестування необхідно забезпечити сумісність між усіма сервісами, що вимагає додаткових ресурсів та ретельного планування. До того ж, моніторинг роботи системи стає складнішим, адже кожен сервіс функціонує окремо, і для повноцінного аналізу потрібно мати інструменти, які дозволяють комплексно оцінювати стан усієї інфраструктури [28].

Мікросервісна архітектура надає значні переваги в гнучкості, масштабованості та стійкості системи, що робить її хорошим вибором для великих та складних проєктів. Оскільки інформаційна система що розроблятиметься не є настільки об’ємним проєктом то використання даної архітектури є недоцільним. Така архітектура лише ускладнить реалізацію та не надасть конкретних переваг.

Клієнт-серверна архітектура – це модель взаємодії в комп’ютерних мережах, де клієнти до прикладу, веббраузери чи мобільні застосунки, надсилають запити до

серверу, який обробляє ці запити та повертає відповідь [29]. Ця архітектура широко використовується в сучасних вебзастосунках, електронній пошті, базах даних та інших сервісах (рис. 2.3).

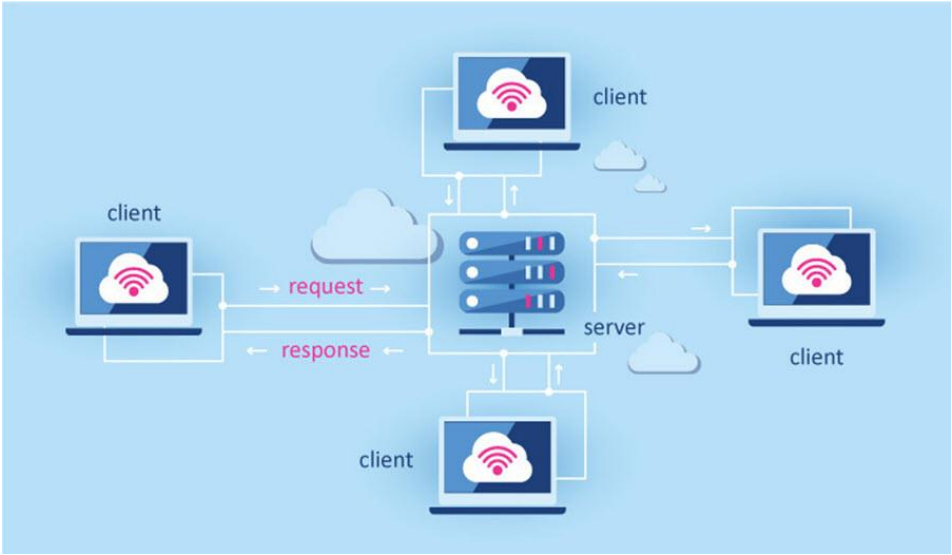


Рисунок 2.3 – Схема клієнт-серверної архітектури [30]

Клієнт-серверна архітектура має суттєві переваги, що роблять її популярною в багатьох інформаційних системах. Однією з основ є централізоване управління даними, де вся інформація зберігається на сервері, це полегшує її обробку, резервне копіювання та впровадження засобів безпеки. Серверна інфраструктура дозволяє обслуговувати значну кількість клієнтів одночасно, а при потребі можна масштабувати систему шляхом додавання нових серверів для балансування навантаження. Оскільки основна обробка інформації виконується на стороні сервера, це забезпечує доступність навіть для користувачів зі слабшими пристроями. Крім того, централізована модель оновлення програмного забезпечення дає змогу миттєво надавати клієнтам нові версії або актуальні дані без потреби в локальному оновленні, що спрощує супровід системи для розробників.

Дана архітектура не позбавлена недоліків. Основним слабким місцем є залежність від сервера: у разі його збою або недоступності всі клієнти автоматично

втрачають можливість користування сервісом. Важливою умовою стабільної взаємодії клієнта і сервера є якісний мережевий канал, оскільки нестабільне або повільне з'єднання знижує продуктивність і може негативно вплинути на досвід користувача. Також в плані безпеки можна сказати, що централізована система приваблює кіберзлочинців, оскільки успішна атака на сервер може поставити під загрозу всю систему. Зокрема, існує ризик атак типу відмови в обслуговуванні (DoS) або спроб несанкціонованого доступу до конфіденційних даних.

Зважаючи на вимоги до майбутньої інформаційної системи, клієнт-серверна архітектура забезпечує оптимальне поєднання гнучкості, масштабованості та безпеки. Вона дозволяє ефективно розподілити обов’язки між клієнтом і сервером, що спрощує розробку та підтримку системи, а також забезпечує зручне управління даними. Тому вибір клієнт-серверної архітектури на основі аналізу інших архітектур є чудовим рішенням.

Клієнт-серверна взаємодія часто реалізується за допомогою REST API.

REST API (Representational State Transfer Application Programming Interface) – це архітектурний стиль для створення веб-сервісів, який дозволяє клієнтам і серверам взаємодіяти через стандартні HTTP-запити. Клієнт надсилає HTTP-запит, а сервер повертає HTTP-відповідь, це дозволяє клієнту виконувати різноманітні операції з ресурсами, які надає сервер [31]. Основними типами запитів є:

– POST – створення ресурсу;
– GET – отримання ресурсу;
– PUT – оновлення ресурсу;
– DELETE – видалення ресурсу.

У межах клієнт-серверної архітектури виділяють дві основні моделі: дворівневу та трирівневу.

Дворівнева архітектура (рис. 2.4) включає два основні вузли:

1) клієнт, який забезпечує інтерфейс користувача та ініціює запити;
2) сервер, який приймає ці запити, обробляє їх за допомогою власних ресурсів, формує та надсилає відповідь.

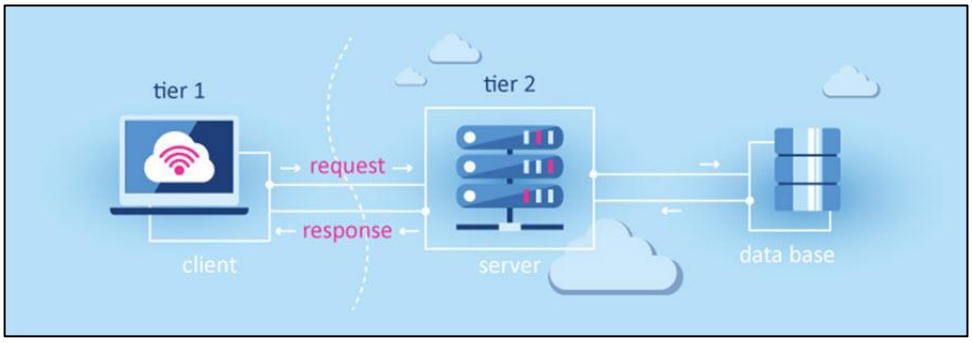


Рисунок 2.4 – Схема дворівневої клієнт-серверної архітектури [30]

У цій моделі взаємодія відбувається без участі додаткових служб чи окремих серверів баз даних – усі операції обробляються в межах одного сервера. Такий підхід спрощує розгортання системи, але обмежує масштабованість і розділення обов'язків між компонентами.

Трирівнева архітектура (рис. 2.5) складаєтьс з трьох логічних рівнів:

1) рівень представлення (presentation layer), тобто інтерфейс, з яким безпосередньо працює користувач;
2) рівень прикладної логіки (application layer) – сервер додатків, який реалізує бізнес-логіку та обробляє запити;
3) рівень доступу до даних (data layer) – сервер бази даних, що зберігає й надає необхідну інформацію.

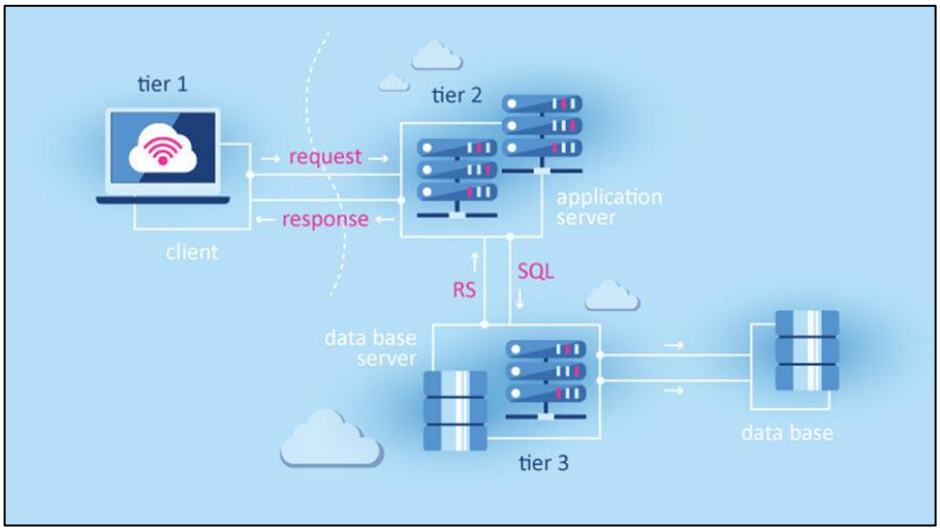


Рисунок 2.5 – Схема трирівневої клієнт-серверної архітектури [30]

Використання окремих серверів для логіки та даних дозволяє ефективно розподілити навантаження між компонентами системи, що особливо важливо для вебзастосунків які одночасно може використовувати велика кількість користувачів.

З огляду на вимоги до функціональності, розділення відповідальностей між кожним рівнем, можливість масштабування та потреби у безпеці, трирівнева клієнт-серверна архітектура є найбільш збалансованим і вірним вибором для побудови інформаційної системи для психодіагностики з використанням таблиці Шульте. Вона дозволить забезпечити якісну організацію коду та покращену підтримку, порівняно з дворівневою архітектурою.

2.2 Аналіз та вибір архітектури побудови вебдодатка

Під час проєктування вебзастосунку важливим етапом є вибір архітектурного підходу до побудови клієнтської частини. Саме архітектура фронтенду визначає, як організована логіка відображення, обробки подій та взаємодії з сервером, а також впливає на продуктивність, зручність масштабування і майбутню підтримку коду.

У системі для психодіагностики з використанням таблиці Шульте важливо забезпечити швидку реакцію інтерфейсу на дії користувача, оскільки затримки можуть вплинути на точність результатів. Результати тесту мають передаватися на сервер через API у реальному часі, що дозволить їх зберігати й аналізувати. Архітектура інтерфейсу повинна бути масштабованою, щоб легко додавати нові функції й адаптувати систему до потреб користувачів. Зважаючи на це, необхідно проаналізувати три найпоширеніші архітектурні моделі побудови вебзастосунків:

- mvc;
- micro-frontend;
- single page application (SPA).

Кожен з підходів варто розглядати з точки зору їх структурних особливостей, переваг, недоліків та відповідності цілям і задачам даного проєкту.

MVC (Model–View–Controller) – це архітектурний шаблон, що передбачає розділення клієнтської логіки на три функціональні компоненти (рис. 2.6).

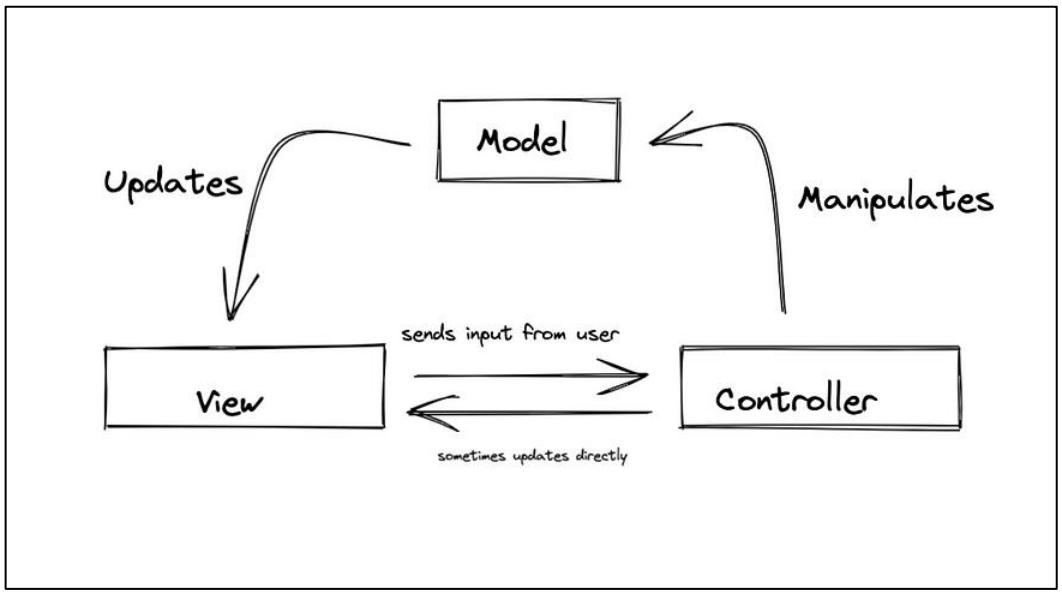


Рисунок 2.6 – Схема роботи MVC [32]

Серед компонентів MVC:

- model (модель), відповідає за зберігання та управління даними, логіку обробки станів застосунку.
- view (подання), відображає дані користувачу і реагує на зміни моделі.
- controller (контролер), приймає вхідні дії користувача (події), обробляє їх і керує оновленням моделі та представлення.

Такий підхід дозволяє досягти чіткого поділу обов'язків у межах клієнтської частини, що особливо корисно при створенні масштабованих та підтримуваних інтерфейсів [33].

Архітектура з поділом на модель, представлення і контролер має низку переваг. Вона дозволяє структурувати код за логічними частинами, що значно спрощує його супровід і подальше маштабування. Моделі можна повторно використовувати в різних компонентах, а також легко організовувати модульне тестування як для моделей, так і для контролерів, що підвищує стабільність і передбачуваність роботи системи.

В той же час така архітектура має і певні недоліки. Реалізація повноцінного MVC на фронтенді потребує глибокого розуміння внутрішньої логіки та ручного керування залежностями, що створює високий поріг входу для новачків. У складних проєктах контролери можуть швидко розростатися, через що код стає важким для підтримки. Крім того, при використанні REST API основна логіка обробки даних зазвичай винесена на сервер, тому моделі на клієнті втрачають свою значущість і не приносять очікуваної користі [34].

Підсумовуючи, можна зробити висновок що MVC-архітектура на клієнті може бути ефективною для окремих типів застосунків, у випадку проєкту, що пов'язаний з інтерактивним тестуванням, фіксацією швидкості реакції, використання цього підходу призведе до ускладнення реалізації без суттєвих переваг. Доцільніше використовувати більш сучасні архітектурні рішення, які краще відповідають вимогам щодо динаміки системи.

Micro-frontend – це архітектурний підхід, при якому інтерфейс користувача великого вебзастосунку розбивається на незалежні, самостійно розроблювані та розгортані частини, які взаємодіють між собою через чітко визначені контракти. Кожен «мікрофронтенд» відповідає за окрему функціональність і може мати власну команду розробників та стек технологій (рис. 2.7).

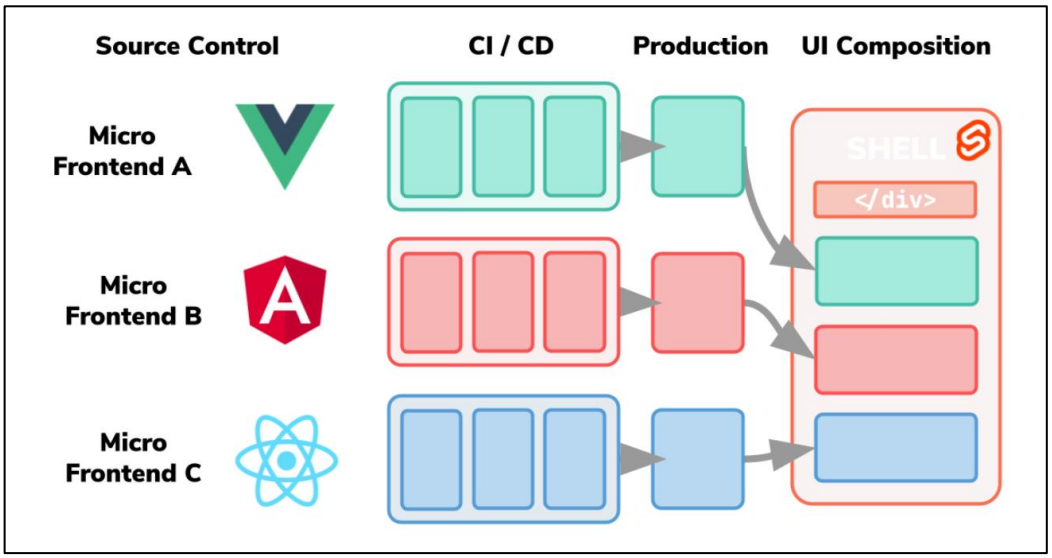


Рисунок 2.7 – Схема реалізації Micro-frontend архітектури [35]

Цей підхід наслідує ідеї мікросервісної архітектури, але на рівні клієнтської частини, забезпечуючи масштабованість і гнучкість у розробці великих фронтенд-систем.

Типова структура micro-frontend системи передбачає наявність контейнерного застосунку (shell), який відповідає за маршрутизацію та об'єднання окремих частин інтерфейсу в єдину цілісну систему. Кожен мікрофронтенд функціонує автономно, може бути незалежно розгорнутий, оновлений або змінений без впливу на інші модулі, що забезпечує гнучкість і масштабованість під час розробки та супроводу [36].

Серед переваг варто зазначити, можливість розподілу роботи між різними командами які в межах одного застосунку можуть використовувати різні технології, завдяки чому можливе швидке оновлення окремих частин вебзастосунку.

Даний підхід має ряд недоліків які є аргументами чому не варто використовувати micro-frontend архітектуру при реалізації інтерфейсу системи.

Підхід micro-frontend вимагає складної інфраструктури, тобто налаштування контейнерного застосунку, маршрутизації, розділеної збірки та інтеграції модулів, наприклад через Webpack Module Federation. Це ускладнює розгортання та підтримку, що не має сенсу для системи з малим функціоналом. Крім того, цей підхід орієнтований на великі команди, які працюють над окремими частинами інтерфейсу. У даному випадку весь фронтенд охоплює логіку створення та відображення таблиці, натискання, вимір часу реакції та передачу результатів, тому реалізація у вигляді одного застосунку є простішою та ефективнішою [37].

З огляду на масштаб системи, відсутність кількох команд розробки, відносно просту логіку взаємодії та вимоги до швидкої реакції інтерфейсу, використання micro-frontend архітектури є недоцільним.

Single Page Application (SPA) – це архітектурний підхід до побудови вебзастосунків, за якого взаємодія з користувачем відбувається в межах єдиної HTML-сторінки (рис. 2.8).

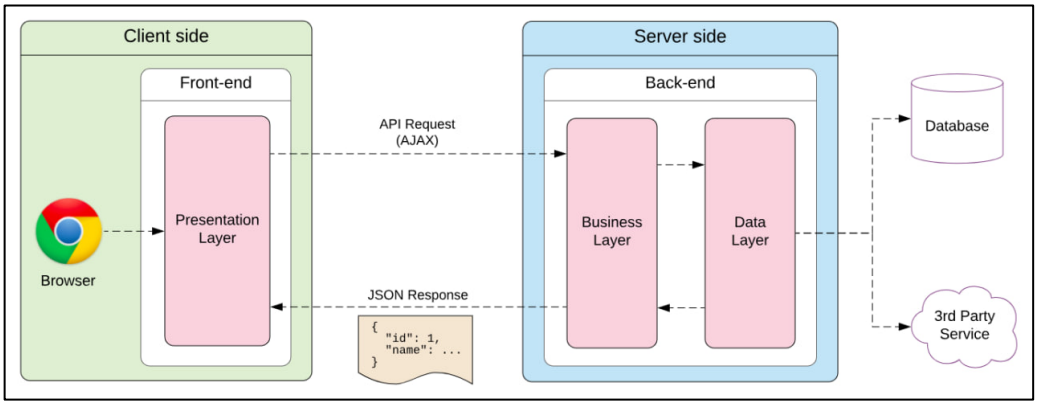


Рисунок 2.8 – Схема взаємодії Single Page Application з сервером [38]

Замість повного перезавантаження сторінок при кожній дії, SPA оновлює лише ту частину інтерфейсу, яка змінюється. Це досягається за рахунок активного використання JavaScript та клієнтських фреймворків, таких як React, Angular чи Vue.

Після першого завантаження SPA отримує HTML, CSS та JavaScript і надалі працює як повноцінна програма в браузері. Для взаємодії з сервером використовуються асинхронні HTTP-запити (зокрема через REST API), що дозволяє динамічно надсилати та отримувати дані без зміни URL чи оновлення всієї сторінки [39].

Однією з ключових переваг архітектури SPA є швидкість реакції інтерфейсу на дії користувача. Оскільки після початкового завантаження вебсторінка не оновлюється повністю, а змінюються лише окремі її елементи, взаємодія з додатком відбувається практично миттєво. Це дозволяє уникнути затримок, характерних для традиційних багатосторінкових сайтів, і особливо актуально у випадку з тестом Шульте, де фіксація натискань і часу реакції потребує максимальної точності та швидкості.

Ще одна перевага полягає в тому, що SPA суттєво зменшує кількість повних запитів до сервера. Уся логіка відображення та навігації виконується на клієнті, а серверна частина залучається переважно для обміну даними через API. Це не лише знижує навантаження на серверну інфраструктуру, а й робить передачу даних більш ефективною та контрольованою [40].

Крім того, підтримка та розширення функціональності в SPA є значно зручнішими завдяки компонентній структурі. Застосунок розбивається на окремі компоненти, які можна ізолювати, змінювати, повторно використовувати та незалежно тестувати. Така модульність особливо корисна при додаванні нових можливостей, зокрема нових режимів тестування чи панелі статистики.

Варто також зазначити, що SPA забезпечує сучасний користувацький досвід, який наближений до нативних застосунків. Такий рівень взаємодії позитивно сприймається користувачами.

Отже, Single Page Application є найдоцільнішим архітектурним підходом для реалізації клієнтської частини інформаційної системи. Він забезпечує високу продуктивність, гнучкість, інтерактивність та відповідність сучасним вимогам до UX, а також добре масштабується у разі подальшого розвитку системи.

2.3 Аналіз та вибір архітектури серверної частини

Розробка серверної частини інформаційної системи вимагає обґрунтованого вибору архітектурного підходу до організації бізнес-логіки, взаємодії з базою даних та забезпечення масштабованості, безпеки, а також супроводу. Архітектура бекенду визначає, як компоненти системи зв’язані між собою, яким чином обробляються запити, зберігаються й передаються дані.

Необхідно проаналізувати такі архітектурні підходи як шарова (layered) та модульна архітектури.

Шарова архітектура є одним із найпоширеніших архітектурних шаблонів у серверній розробці. Її суть полягає в логічному поділі застосунку на окремі шари, кожен з яких виконує свою функцію та взаємодіє лише з сусідніми шарами. Такий підхід забезпечує зрозумілу структуру коду, спрощує підтримку та дозволяє розмежувати відповідальність між частинами системи.

У багатошаровій архітектурі бекенду виділяють три основні шари (рис. 2.9), серед яких контролерний, сервісний і репозиторний. Контролерний шар обробляє

HTTP-запити, передає дані в сервіси й формує відповіді. Сервісний шар реалізує бізнес-логіку, а саме перевірки, обчислення, взаємодію з іншими модулями. Репозиторний шар відповідає за доступ до бази даних, тобто зчитування, запис та оновлення [41].

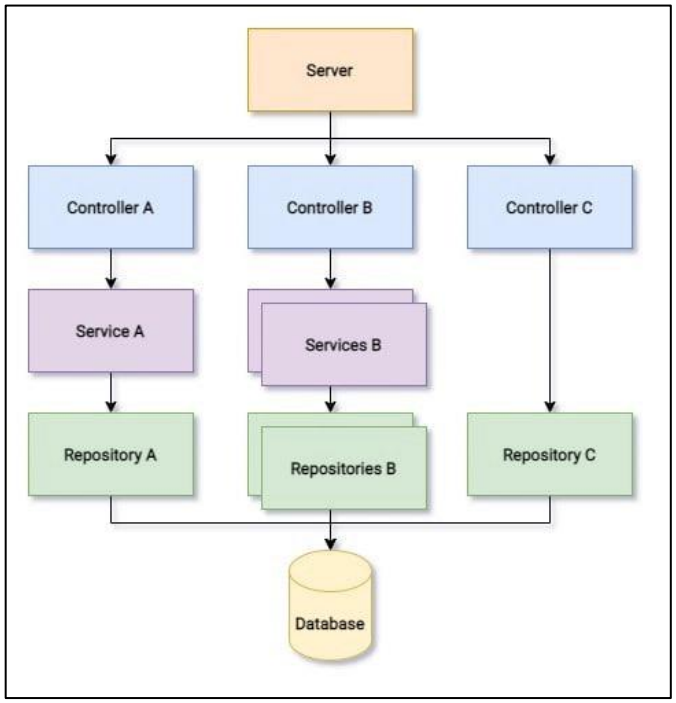


Рисунок 2.9 – Схема шарової архітектури серверної частини [41]

Кожен шар має свою чітко окреслену роль, що дозволяє змінювати чи замінювати один із шарів без необхідності вносити зміни в інші. Завдяки цьому розробники можуть незалежно працювати над окремими частинами системи, не створюючи небажаних залежностей між модулями. Крім того, структура, побудована за принципом «зверху вниз» і дає змогу швидко орієнтуватися в проєкті, що особливо корисно в командній розробці [42].

Серед недоліків це те, що проєктах із простою логікою реалізація повноцінної багаторівневої структури може створити надмірне ускладнення. Кожна нова функція потребує створення окремих методів на кількох рівнях, навіть тоді, коли ці дії є необов'язковими. Це призводить до дублювання коду, збільшення кількості файлів та ускладнення роботи при розробці невеликих застосунків.

Використання повноцінної шарової архітектури є недоцільним для інформаційної системи, яка реалізує лише кілька ключових сценаріїв використання. Функціональність системи досить обмежена, тому додаткове структурування коду в межах окремих рівнів бізнес-логіки ускладнить загальну реалізацію. У випадку цього проєкту значно важливіше зберегти просту, логічно цілісну структуру, яка відповідає реальній складності задач.

Модульна архітектура – це підхід до проєктування програмного забезпечення, за якого система розділяється на незалежні функціональні модулі. Кожен модуль відповідає за реалізацію конкретної частини логіки та виконує чітко визначене завдання. При цьому модулі взаємодіють один з одним через стандартизовані інтерфейси, залишаючись автономними [43].

Такий підхід дозволяє структурувати систему у вигляді набору логічно завершених блоків, які легко підтримувати, тестувати, розвивати та масштабувати. В архітектурі вебзастосунків модульність зазвичай реалізується на рівні директорій або сервісів: наприклад, окремі модулі для користувачів, тестових сесій, аналітики, результатів тощо.

Однією з переваг є гнучкість структури проєкту, тобто можна швидко ізолювати й змінювати конкретну функціональність без ризику вплинути на решту системи. Завдяки цьому будь-яка частина системи до прикладу, обробка результатів тесту чи збереження натискань – може бути реалізована у власному модулі з чіткими зонами відповідальності.

Ще одним вагомим плюсом є простота тестування й масштабування логіки. Оскільки модулі не мають жорстких залежностей один від одного, їх легко перевіряти окремо, а за потреби – розширювати, додаючи нові залежності або функціонал без зміни базової структури проєкту.

Крім того, модульність добре відображає логіку самого застосунку. Основні функції можуть бути реалізовані як самостійні модулі. Це підвищує зрозумілість коду, та при потребі надає змогу новим учасникам проєкту швидко орієнтуватися в його структурі.

Для інформаційної системи з реалізацією тесту Шульте, яка має обмежений але чітко структурований функціонал, модульна архітектура є найбільш збалансованим і практичним рішенням. Вона дозволяє організувати код просто і водночас логічно, не ускладнюючи його зайвими абстракціями. Така архітектура надає всю гнучкість, необхідну для подальшого розширення – наприклад, для реалізації аналітики, звітів, багатокористувацького режиму тощо – без потреби перебудовувати всю систему.

2.4 Вибір бази даних

Одним із ключових етапів проєктування інформаційної системи є вибір бази даних, оскільки саме вона відповідає за надійне зберігання, доступність, а також обробку даних користувачів, результатів тестування та аналітичної інформації. Для розробки вебзастосунку для психодіагностики з використанням таблиці Шульте до бази даних висуваються чіткі вимоги, а саме підтримка структурованого зберігання сесій тестування, можливість фіксації натискань, зберігання параметрів проходження, забезпечення доступу до записів, та подальшого аналізу результатів.

Отже, необхідно розглянути основні типи баз даних, які можуть бути використані у вебзастосунках, зокрема реляційні та нереляційні. Здійснити порівняльний аналіз їхніх можливостей для поставленої задачі, після чого обґрунтувати вибір конкретного типу та реалізації бази даних, яка найкраще відповідає функціональним і нефункціональним вимогам даної системи.

Реляційні СУБД (наприклад, PostgreSQL, MySQL) базуються на суворо структурованій моделі з таблицями, полями, первинними й зовнішніми ключами. Дані зберігаються у вигляді рядків у таблицях, а взаємозв’язки між таблицями реалізуються через нормалізацію [44].

Нормалізація – це процес організації структури реляційної бази даних з метою усунення надмірності даних, уникнення аномалій при оновленні, вставці або видаленні записів, а також забезпечення логічної цілісності. Вона передбачає поділ

даних на кілька пов’язаних таблиць і визначення чітких взаємозв’язків між ними за допомогою первинних (primary key) та зовнішніх ключів (foreign key) [45].

Процес нормалізації здійснюється шляхом послідовного застосування так званих нормальних форм, кожна з яких накладає додаткові обмеження на структуру таблиць.

На першому етапі, в межах першої нормальної форми (1NF), забезпечується відсутність повторюваних груп, а всі поля мають містити лише атомарні значення. Друга нормальна форма (2NF) передбачає усунення часткових залежностей, тобто якщо таблиця має складений первинний ключ, усі неключові атрибути мають залежати від усього ключа, а не від окремої його частини. Якщо така залежність виявляється, дані слід розділити між окремими таблицями. На завершальному етапі третьої нормальної форми (3NF) усуваються транзитивні залежності, тобто ситуації, коли неключовий атрибут має залежність не напряму від ключа, а через інший неключовий атрибут. Тоді поля потрібно винести в окрему таблицю.

Бази типу NoSQL (наприклад, MongoDB, CouchDB) зберігають дані у вигляді документів, зазвичай у форматі JSON. На відміну від реляційних баз, такі системи не вимагають заздалегідь чітко визначеної структури таблиць, тому дозволяють зберігати дані різних типів та змінювати їхню структуру в процесі роботи, без потреби у складних міграціях [46].

Такий тип бази даних може бути зручним у випадках, коли структура записів часто змінюється, або коли дані не піддаються нормалізації. Наприклад, у складних аналітичних системах зі змінними метаданими або логами користувацької поведінки, ці бази дозволяють уникнути жорстких схем і зберігати всі пов’язані дані в одному об’єкті.

Зважаючи на особливості інформаційної системи, що розробляється, найбільш доцільним виглядає вибір реляційної системи управління базами даних (СУБД). У межах проєкту передбачається зберігання структурованих даних, зокрема інформації про користувачів, параметри проходження тесту, послідовність натискань, час реакції та помилки. Такі дані мають чітко окреслену структуру та

логічні зв’язки між собою і можуть бути реалізовані через таблиці, первинні й зовнішні ключі.

Крім того, система повинна зберігати значну кількість однотипних записів, таких як натискання на клітинки, які зручно фіксувати у вигляді рядків у пов’язаних таблицях з можливістю подальшої агрегації.

Важливо також передбачити можливість виконання аналітичних. Реляційні системи управління базами даних (СУБД), на відміну від нереляційних NoSQL-систем, краще пристосовані до складних вибірок, агрегатних функцій та багатотабличних зв’язків [47].

З огляду на те, що архітектура системи планується як трирівнева клієнт-серверна модель з REST API, важливо, щоб обрана база даних легко інтегрувалася з серверною частиною та підтримувала взаємодію через стандартизовані протоколи. Реляційна СУБД у такому випадку забезпечує як структурованість і цілісність даних, так і зручний доступ до них через SQL-запити та ORM-рішення, що добре сумісні з популярними бекенд-фреймворками.

Отже, вибір реляційної бази даних є найоптимальнішим для поставленої задачі. Такий тип сховища дозволяє реалізувати чіткі зв’язки між сутностями, виконувати складні запити для аналізу даних, а також гарантує надійність і масштабованість.

2.5 Вибір стеку технологій

Також варто обрати стек технологій що дозволить ефективно реалізувати як клієнтську, так і серверну частини інфомаційної системи, забезпечить інтеграцію з базою даних, а також взаємодію через API. Стек технологій визначає мову програмування, фреймворки, платформи, засоби зберігання та обміну даними, які будуть використані на кожному рівні системи.

Основною мовою реалізації як вебінтерфейсу, так і серверної частини системи обрано JavaScript. По-перше, JavaScript є стандартом для розробки

клієнтських вебзастосунків, що робить його чудовим вибором для реалізації інтерактивного інтерфейсу у браузері. По-друге, ця мова підтримується також і на серверному боці, завдяки платформі Node.js.

Крім того, JavaScript має широку екосистему бібліотек і фреймворків, що дає змогу швидко реалізувати складні інтерфейси, ефективно обробляти події користувача, створювати REST API та працювати з базами даних. Завдяки цьому є можливість будувати повноцінну функціональну систему без зміни мови програмування між окремими її частинами.

Для створення користувацького інтерфейсу обрано React – популярну JavaScript-бібліотеку для розробки динамічних односторінкових застосунків (SPA). Вона дозволяє ефективно оновлювати DOM (Document Object Model), забезпечує високу продуктивність взаємодії, а також добре поєднується з іншими інструментами фронтенд-розробки. React дозволяє гнучко реалізувати такі функції, як підказки під час проходження тесту, таймер, підрахунок натискань та часу реакції, а також передачу зібраних результатів до серверної частини [48].

У якості серверного середовища обрано NestJS – прогресивний фреймворк для Node.js, що базується на TypeScript і підтримує модульну архітектуру [49]. NestJS зручний для створення REST API, дозволяє організовувати код у вигляді окремих контролерів, сервісів і DTO. Крім того, фреймворк забезпечує підтримку роботи з базами даних, middleware, guard-логікою, ін’єкціями та іншим.

Для зберігання структурованої інформації про користувачів, сесії та результати тестування обрано Supabase [50]. Це хмарна платформа, що надає інтерфейс до реляційної бази даних PostgreSQL. Supabase поєднує в собі зручність керування базою даних, вбудовану авторизацію, наявність REST/RPC AP.

2.6 Проєктування структури бази даних

Ефективне функціонування інформаційної системи неможливе без правильно спроєктованої структури бази даних. На цьому етапі визначаються

ключові об’єкти системи, їхні властивості та взаємозв’язки між ними. З цією метою використовується ER-моделювання – метод побудови концептуальної схеми бази даних за допомогою ER-діаграми (від англ. Entity-Relationship Diagram), яка відображає об’єкти предметної області та зв’язки між ними [51].

Сутність – це реальний або абстрактний об’єкт, який зберігається в базі даних.

Атрибут – це характеристика сутності, тобто окрема одиниця даних, що описує об’єкт.

Зв’язок – це логічна залежність між сутностями.

В даній системі є кілька основних сутностей.

User – користувач, який зареєстрований у системі психодіагностики для проходження тестів. Сутність зберігає базові облікові та контактні дані користувача. Атрибути:

- id – унікальний ідентифікатор користувача;
- email – електронна адреса користувача;
- password_hash – хешований пароль користувача;
- created_at – дата створення облікового запису.

Test_session – сутність, що описує окремий сеанс проходження тесту Шульте. Вона містить усі налаштування, за якими відбувається проходження тесту, а також загальні підсумки сеансу. Атрибути:

- id – унікальний ідентифікатор тест-сесії;
- user_id – зв’язок із користувачем, що проходив тест;
- mode – режим проходження тесту (в зростаючому або спадному порядку);
- grid_size – розмір таблиці (кількість клітинок по ширині/висоті);
- time_limit_enabled – чи було ввімкнено обмеження за часом;
- time_limit – максимальний час проходження (в секундах);
- error_limit_enabled – чи було обмеження за кількістю помилок;
- max_errors – максимальна кількість помилок;
- per_cell_time_limit_enabled – обмеження за часом на кожну клітинку;

– per_cell_time_limit – ліміт часу на одну клітинку (в секундах);
– color_mode – чи був активний кольоровий режим клітинок;
– show_completed_cells – чи відображались натиснуті клітинки;
– shuffle_mode – чи перемішувалась сітка після кожного натискання;
– show_hint – чи була підказка поточного числа;
– started_at – час початку проходження тесту;
– ended_at – час завершення тесту;
– total_time – час проходження;
– correct_clicks – кількість правильних натискань;
– wrong_clicks – кількість помилкових натискань;
– avg_reaction_time – середній час реакції;
– end_reason – причина завершення тесту (вручну, по часу, по помилках).

Test_click – сутність, яка зберігає кожне окреме натискання користувача під час тестування. Це дозволяє будувати детальну аналітику по кожному сеансу. Атрибути:

– id – унікальний ідентифікатор натискання;
– session_id – зв’язок із відповідною тест-сесією;
– number_clicked – число, на яке натиснув користувач;
– is_correct – чи було натискання правильним;
– time_from_start – час у мілісекундах від початку тесту до натискання;
– time_from_last_click – час у мілісекундах від попереднього натискання.

User_analytics – сутність, яка зберігає агреговану статистику користувача по всіх сесіях тестування. Атрибути сутності:

– user_id – унікальний ідентифікатор користувача (ключ і зв’язок із users);
– total_tests – загальна кількість пройдених тестів;
– average_reaction_time – середній час реакції у мілісекундах;
– average_errors – середня кількість помилок;
– best_time – найшвидший час проходження тесту;

– worst_time – найдовший час проходження тесту;
– last_test_at – дата та час останнього проходження тесту;
– created_at – дата створення статистичного запису;
– updated_at – дата останнього оновлення статистики.

Отже, була побудована логічна модель бази даних (рис. 2.10).

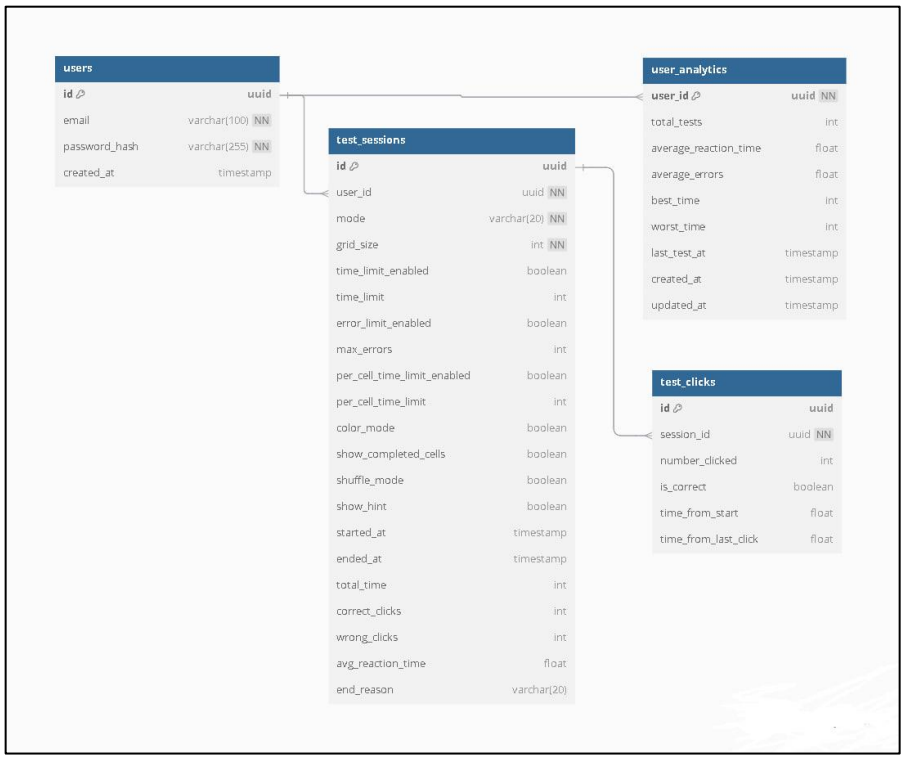


Рисунок 2.10 – ER-діаграма бази даних

ER-моделювання дозволяє наочно відобразити, які об’єкти існують у системі, як вони описуються та як між собою взаємодіють. Це забезпечує логічно узгоджену структуру бази даних, яка легко інтегрується з серверною частиною системи та може бути розширена в майбутньому без порушення її цілісності.

2.7 Проєктування інтерефейсу вебдодатка

У сучасних вебзастосунках інтерфейс відіграє одну з ключових ролей, адже саме через нього користувач взаємодіє з системою. Зручний, інтуїтивно зрозумілий

інтерфейс може суттєво покращити користувацький досвід. Якщо інтерфейс заплутаний, складний чи перевантажений, користувач може втратити бажання користуватися системою, навіть якщо вона має корисні функції.

Інтерфейс вебдодатка має забезпечити інтуїтивну взаємодію користувача з усіма ключовими можливостями системи, зокрема налаштуванням параметрів тесту, його проходженням та переглядом результатів. Крім того, важливо, щоб інтерфейс був адаптивним, тобто зручним для користувачів на різних пристроя.

Під час проєктування інтерфейсу було враховано базові принципи UX/UI-дизайну. Насамперед це мінімалізм, тобто відмова від зайвих візуальних елементів, що можуть відволікати увагу під час проходження тесту. Додаток має бути простим та водночас функціональним. Також варто врахувати зрозумілість для користувача, тобто структура сайту має бути логічною.

Ще одним важливим принципом є єдність стилю: кольорова гама, шрифти та розмітка інтерфейсу мають бути витримані в одному стилі..

З огляду на зазначені вимоги, розроблено прототип майбутнього сайту (рис. 2.11). Прототип складається з шапки (header), яка міститиме назву додатку та кнопку входу яка водночас буде кнопкою меню, містить основну частини (main), у якій буде відбуватися як тестування, так і відображення результатів, та підвал (footer), де розміщуватиметься загальна інформація, контакти або авторські права.

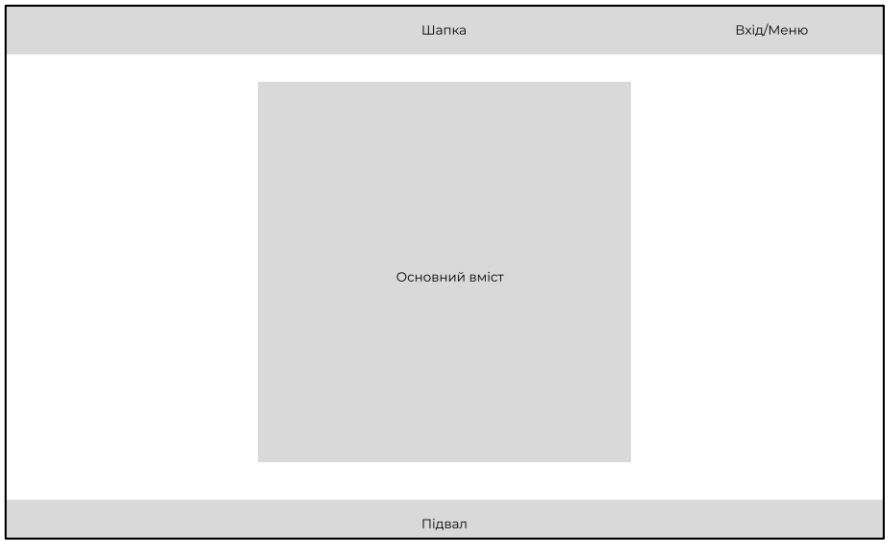


Рисунок 2.11 – Прототип інтерфейсу

Взаємодію користувачів з інтерфейсом системи можна відобразити за допомогою діагарами сценаріїв використання (рис. 2.12).

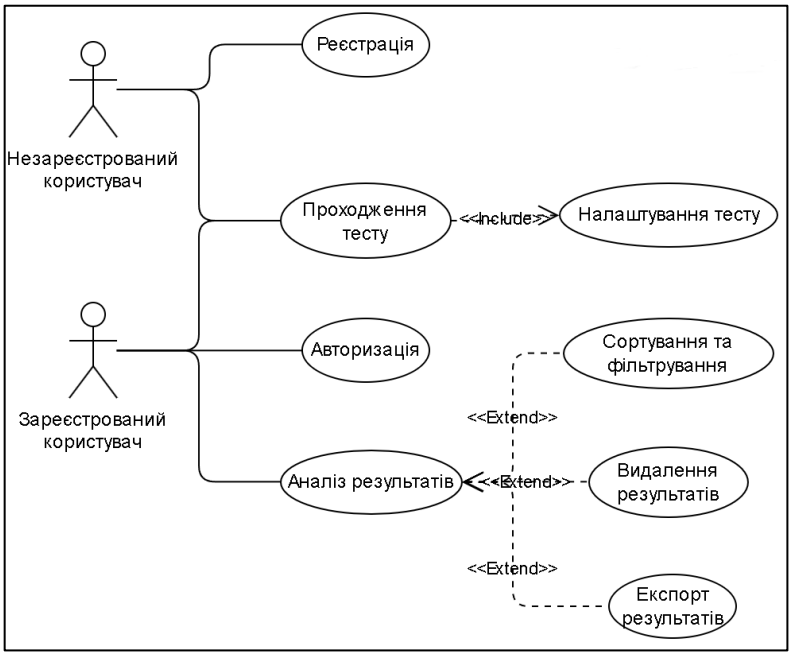


Рисунок 2.12 – Діаграма сценаріїв використання

Загалом діаграма сценаріїв використання (Use Case Diagram) – це один із типів діаграм у мові моделювання UML (Unified Modeling Language), який використовується для опису взаємодії користувачів із програмною системою. Вона демонструє, які функціональні можливості, тобто сценарії використання, доступні різним типам користувачів системи.

2.8 Проєктування серверної частини системи

Моделювання серверної частини не менш важливе при розробці інформаціїйної системи. Завдяки бекенду відбуватиметься взаємодія з базою даних та обробка результатів тестування. Зважаючи на те, що для реалізації було обрано модульну архітектуру, яка реалізовуватиметься з використанняс фреймоврку NestJS, за допомогою діаграми компонентів можна чітко відобразити всі необхідні модулі, що розроблятимуться (рис. 2.13).

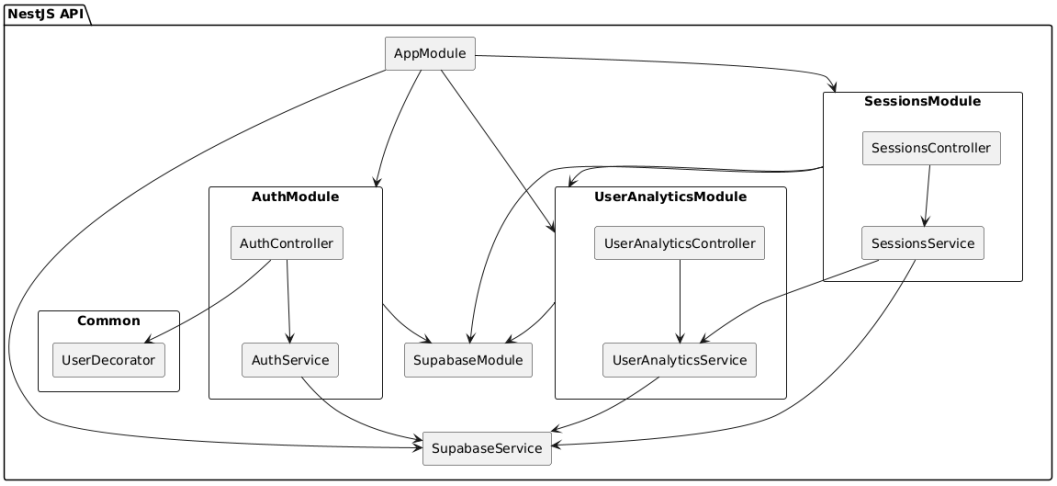


Рисунок 2.13 – Діаграма компонентів

Діаграма компонентів – це тип UML-діаграми, що відображає логічну структуру програмної системи у вигляді окремих компонентів (модулів) та зв’язків між ними. Вона демонструє, які інтерфейси реалізує кожен компонент і як вони взаємодіють між собою.

2.9 Висновки до другого розділу

У другому розділі було здійснено поетапне проєктування інформаційної системи. Обґрунтовано вибір трирівневої клієнт-серверної архітектури, що забезпечує гнучке розділення функцій між клієнтом, сервером та базою даних.

На основі аналізу підходів до побудови інтерфейсу визначено доцільність використання SPA-моделі з фреймворком React, розроблено прототип інтерфейсу.

Для бекенду обрано модульну архітектуру як найкращу для структурованого, але не надто складного проєкту, зображено модулі за допомогою діаграми станів.

Порівнявши типи баз даних, перевагу надано реляційній системі управління базою данних завдяки її структурованості, підтримці зв’язків і зручності в обробці даних. Визначено основний стек технологій, який забезпечує ефективну реалізацію та подальшу масштабованість системи. А також розроблено ER-діаграму, майбутньої бази даних.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПСИХОДІАГНОСТИКИ З ВИКОРИСТАННЯМ ТАБЛИЦІ ШУЛЬТЕ

3.1 Розробка бази даних

Після завершення логічного проєктування структури бази даних, варто здійснити її безпосередню реалізацію з використанням платформи Supabase, хмарного рішення на основі PostgreSQL, що забезпечує зручний інтерфейс для керування базою даних, генерацію REST API, підтримку Row Level Security (RLS).

В даному випадку Supabase виконуватиме подвійну роль –як хостинг бази даних, та як провайдер авторизації.

Для початку роботи з Supabase потрібно зареєструватись або увійти в обліковий запис на офіціному сайті. Після чого створити новий проєкт, вказавши його назву, регіон розміщення сервера та пароль до бази даних PostgreSQL (рис. 3.1).

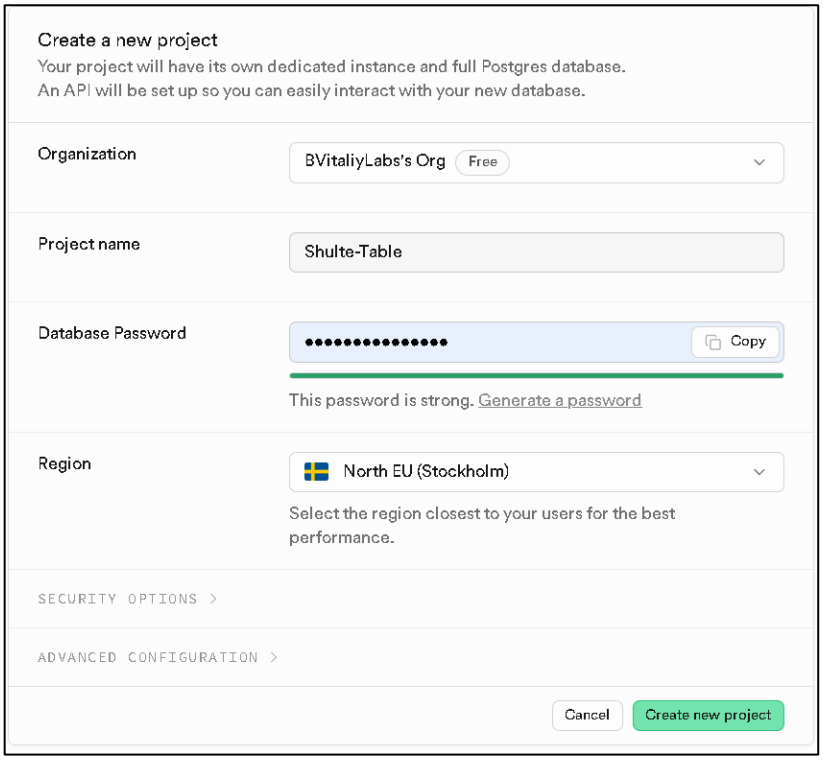


Рисунок 3.1 – Створення проєкту в Supabase

Успішно створивши проєкт, можна розглянути бокову панель навігації, яка забезпечує основну частину взаємодії, та містить ключовіі функції платформи.

Першим елементом є Project Overview, тобто стартова сторінка проєкту, де розміщується загальна інформація, така як URL API, ключі доступу, статус бази даних, статистика використання ресурсів, а також швидкі посилання на документацію Supabase. Цей розділ слугує своєрідною панеллю управління всім проєктом і дозволяє швидко оцінити стан системи.

Другий розділ, це Table Editor, який надає можливість працювати з таблицями бази даних у графічному інтерфейсі. Тут можна створювати нові таблиці, додавати або редагувати поля, налаштовувати типи даних, створювати зв'язки між таблицями, а також керувати політиками доступу. Цей інструмент є зручним для проєктування структури без написання SQL-коду, що може бути хорошим інструментом для початківців.

Для більш досвідчених користувачів передбачено SQL Editor, вбудований редактор SQL-запитів, що дозволяє безпосередньо виконувати команди створення, оновлення, вибірки або видалення даних. Також він дозволяє зберігати запити, переглядати історію змін і тестувати запити перед їх використанням у програмному коді.

Розділ Database – це розширене представлення структури бази даних. Тут можна переглянути всі таблиці, представлення (views), функції, тригери, індекси, типи, а також підключити необхідні розширення PostgreSQL. Через цей модуль виконується глибше налаштування бази, включно з правами доступу та структурними змінами.

Важливу роль у системі відіграє розділ Authentication, де налаштовується система аутентифікації користувачів. У цьому блоці можна керувати обліковими записами, переглядати список зареєстрованих користувачів, налаштовувати методи входу за допомогою різних провайдерів по типу email, google, magic link, визначати політики безпеки, а також тривалість дії JWT-токенів, які використовуються для авторизації.

Розділ Storage дозволяє керувати файлами в межах проєкту. Тут створюються так звані «бакети» (відра), в яких можна зберігати зображення, документи чи будь-які інші файли. Кожен «бакет» може бути публічним або приватним, а взаємодія з ним здійснюється через вебінтерфейс та через API-запити. Цей функціонал особливо актуальний, коли потрібно реалізувати, наприклад, зберігання аватарів користувачів або експорт звітів.

Інноваційним інструментом є Edge Functions – розділ для створення серверних функцій, які виконуються на стороні Supabase без потреби у зовнішньому сервері. Такі функції пишуться на TypeScript і можуть обробляти запити, взаємодіяти з базою або іншими API, що особливо зручно для обробки подій, валідації даних або реалізації кастомної логіки.

Останнім у списку розміщено розділ Realtime, який відповідає за реалізацію функціоналу реального часу. За допомогою нього можна налаштувати підписку на події в таблицях (додавання, оновлення, видалення записів), це дозволяє створювати інтерактивні інтерфейси для чатів, дашбордів або онлайн-аналітики.

Так як платформа Supabase побудована на основі реляційної СУБД PostgreSQL, для реалізації структури бази даних необхідно підготувати SQL-скрипти, які створюватимуть відповідні таблиці, визначатимуть їх поля, типи даних, первинні та зовнішні ключі, а також забезпечуватимуть зв’язки між сутностями.Тому необхідно скористатись нещодавно розглянутим SQL Editor, де ці запити виконуються вручну або зберігаються у вигляді шаблонів для подальшого повторного застосування (рис. 3.2).

.

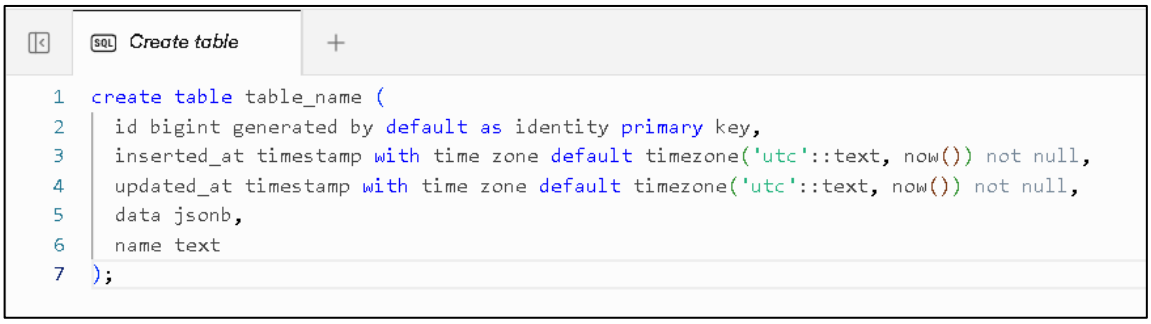


Рисунок 3.2 – Приклад використання типового шаблону в SQL Editor

Використовуючи цей приклад можна створити власну таблицю, задаючи відповідні первинні та зовнішні ключі, до прикладу таблицю test_settions (рис. 3.3).

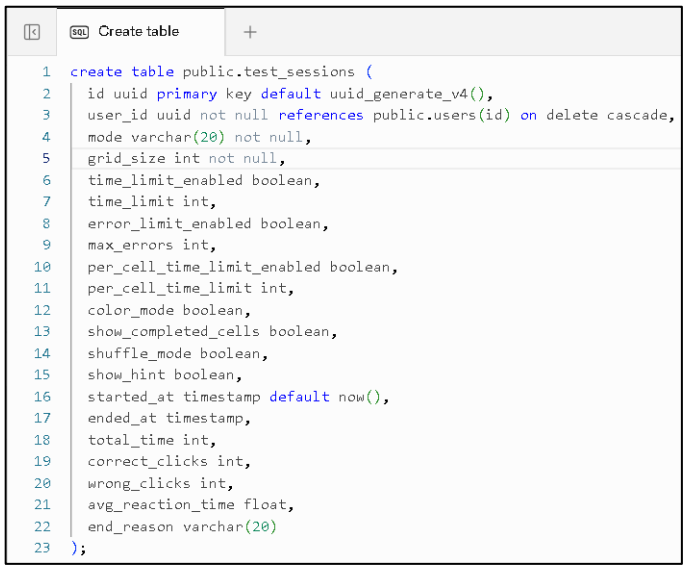


Рисунок 3.3 – SQL-запит для створення таблиці сесій

Створена таблиця має поле id, яке є унікальним ідентифікатором кожної сесії, що автоматично генерується функцією uuid_generate_v4(). Також можна помітити що таблиця містить зовнішній ключ user_id який посилається на таблицю users, цим самим створюючи зв'язок типу багато до одного, тобто один користувач може мати багато сесій. Для кращого розуміння встановлення зв’язків, доцільно створити ще й таблицю test_click (рис. 3.4), яка теж напряму пов’язана з таблицею test_sessions через поле session_id, яке посилається на test_sessions.id.

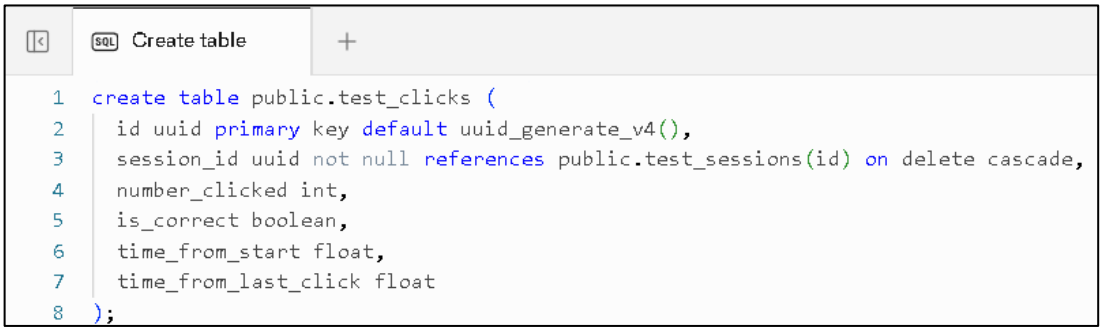


Рисунок 3.4 – SQL-запит для створення зв’язаної таблиці натискань

Завдяки такому зв’язку, забезпечується логічний зв’язок, і усі натискання, що здійснюються в межах однієї сесії, можуть бути вибрані з бази за одним ідентифікатором. Наприклад отримання повного списку натискань песної сесії можна здійснити запитом як на рисунку 3.5.

```
1  SELECT * FROM test_clicks WHERE session_id = 'value_of_session_id';
```

Рисунок 3.5 – SQL-запит на вибірку всіх натискань сесії

Наступним важливим кроком після створення таблиць є налаштування Row Level Security (RLS).

Щоб увімкнути RLS для певної таблиці, потрібно перейти у розділ Table Editor, обрати відповідну таблицю, і в її верхньому правому куті активувати перемикач «Enable Row Level Security» (рис. 3.6). Після цього стає доступною вкладка Policies, де можна створити правила доступу.

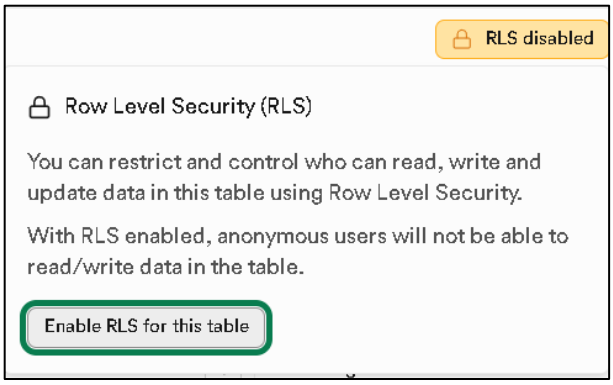


Рисунок 3.6 – Вмикання RLS

Основна логіка цього механізму полягає в тому, щоб дозволити доступ лише до тих рядків, де значення певного поля, до прикладу user_id, збігається з ідентифікатором поточного автентифікованого користувача.Це означає, що користувач бачитиме лише ті сесії, які належать саме йому. Тому, налаштування RLS є невід’ємною частиною побудови захищеної інформаційної системи, особливо коли мова йде про збереження психодіагностичних даних користувачів.

Також необхідно налаштувати авторизацію, яка використовуватиметься у проєкті для управління доступом користувачів до функціоналу системи.

У межах платформи Supabase для цього використовується вбудований модуль authentication, який забезпечує підтримку реєстрації, входу в систему та керування сесіями за допомогою JWT-токенів. Налаштування авторизації передбачає активацію методу автентифікації через email і пароль, конфігурацію параметрів тривалості сесії, а також забезпечення автоматичної видачі токена після входу (рис. 3.7).

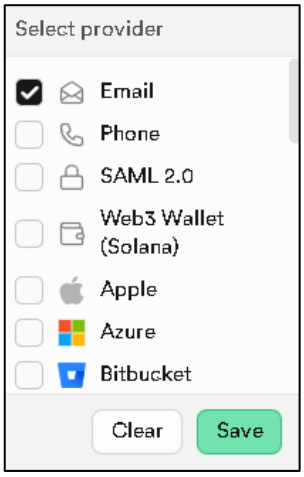


Рисунок 3.7 – Налаштування автентифікації через електронну пошту

В падальшому цей токен дозволить ідентифікувати користувача за допомогою вбудованої функції auth.uid(). Це налаштування забезпечить безпечний доступ до даних, а також слугуватиме основою для всіх операцій із базою, які залежать від ідентичності користувача.

3.2 Розробка серверної частини

Для забезпечення повноцінної взаємодії між клієнтською частиною, базою даних і механізмами авторизації потрібно розробити серверну частину застосунку з використанням NestJS, який є сучасним фреймворком для Node.js, та підтримує модульну архітектуру, а також забезпечує високу масштабованість і

структурованість проєкту. Основними задачами серверної частини є забезпечення прийому та обробки запитів, які надходять від клієнтської частини застосунку, тобто вебінтерфейсу. Крім того, серверна логіка відповідає за взаємодію з базою даних Supabase, вона виконує операції зчитування, запису та оновлення інформації згідно з логікою застосунку. Важливою складовою є також перевірка авторизації користувачів: на цьому етапі система аналізує JWT-токени, які передаються клієнтом, і визначає, чи має користувач право на виконання конкретного запиту. Серверна частина реалізує всю бізнес-логіку проєкту, таку як, агрегацію результатів тестів, обробку потенційних помилок і оновлення аналітичних даних, що відображають динаміку результатів користувача.

Зважаючи на спроектовані раніше модулі, для початку можна створити базову файлову струтуру проєкту (рис. 3.8), яка й формуватиме архітектуру.

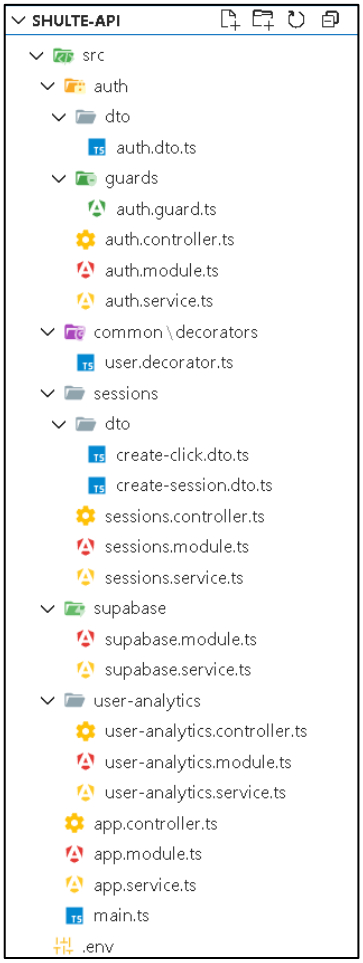


Рисунок 3.8 – Файлова структура серверної частини системи

Перш ніж розглядати кожен модуль, варто зрозуміти як саме за допомогою NestJS реалізовується модульна архітектура і яку функціональну роль відіграє така структура у побудові бекенду.

NestJS побудований за принципами інверсії управління (IoC) та впровадженні залежностей (DI), що передбачає поділ застосунку на окремі логічні частини, так звані модулі. Кожен модуль вміщує в собі всі необхідні елементи для реалізації певної функціональності, оновними з якиї є контролери, сервіси, DTO-класи (Data Transfer Objects), обмежувачі доступу (guards) та декоратори.

Контролери у NestJS відповідають за прийом HTTP-запитів від клієнтської частини. Саме вони визначають, які маршрути, існують у застосунку, які параметри можуть передаватись у запиті, і яку відповідь отримає клієнт. Контролери не мають містити бізнес-логіки, тому що їхнє завдання полягає в тому, щоб делегувати обробку запиту до відповідного сервісу.

Сервіси ж, реалізують бізнес-логіку застосунку. У них зосереджено всю обробку даних, взаємодію з базою, обчислення, перевірки, фільтрацію та інше. Сервіси можуть використовуватися і в контролерах і в інших сервісах, цим самим забезпечуючи гнучкість і повторне використання логіки.

DTO-класи (Data Transfer Objects) є спеціальними об’єктами, які визначають структуру даних, що передаються у запитах, до прикладу в даній системі при створенні сесії або збереженні натискання. DTO допомагають уніфікувати дані, а також використовуються для валідації вхідних значень, забезпечуючи захист від некоректного введення з боку клієнта.

Обмежувачі доступу (guards) використовуються для перевірки прав доступу до певного ресурсу. Наприклад, guard може перевіряти, чи передано дійсний JWT-токен у запиті і вже після цього вирішувати, чи дозволити виконання маршруту. У NestJS guards виконуються перед викликом контролера.

Декоратори, спеціальні конструкції, які дозволяють додавати метадані до класів, методів або параметрів. В NestJS вони широко використовуються, наприклад, для вказування HTTP-методів (@Get, @Post).

Отже, варто перейти безпоередньо до реалізації. Спочатку слід встановити базові залежністі за допомогою команди npm install @supabase/supabase-js @nestjs/swagger.

Після цього необхідно створити файл змінних середовища .env у корені проєкту. Він зберігатиме такі поля як SUPABASE_URL, що зберігатиме адресу бази даних, та SUPABASE_SERVICE_ROLE_KEY, який є секретним ключем для доступу на рівні сервісу. Це дозволить розділити конфігурацію середовища та бізнес-логіку застосунку, що підвищує безпеку та гнучкість.

Далі доцільно створити сервіс SupabaseService, який буде інкапсулювати логіку взаємодії з Supabase (рис. 3.9).

```
import { Injectable } from '@nestjs/common';
import { createClient, SupabaseClient } from '@supabase/supabase-js';

@Injectable()
export class SupabaseService {
  public client: SupabaseClient;

  constructor() {
    const supabaseUrl = process.env.SUPABASE_URL!;
    const supabaseKey = process.env.SUPABASE_SERVICE_ROLE_KEY!;
    this.client = createClient(supabaseUrl, supabaseKey);
  }

  async getUserFromToken(token: string) {
    return this.client.auth.getUser(token);
  }
}
```

Рисунок 3.9 – Реалізація SupabaseService

В цьому сервісі клієнт ініціалузуватиметься за допомогою значень з .env, також тут реалізується метод для перевірки JWT-токена.

Для доступності цього та інших сервісів у всіх частинах застосунку, він виноситься в окремий SupabaseModule та експортується (рис. 3.10).

```
import { Module } from '@nestjs/common';
import { SupabaseService } from './supabase.service';

@Module({
  providers: [SupabaseService],
  exports: [SupabaseService],
})
export class SupabaseModule {}
```

Рисунок 3.10 – Вигляд SupabaseModule

В свою чергу, AppModule є центральним модулем і імпортує всі модулі.

У процесі налаштування початкової конфігурації важливо також оновити вхідну точку застосунку main.ts. Тут необхідно активувати підтримку CORS (Cross-Origin Resource Sharing), що дозволяє клієнтській частині застосунку безпечно взаємодіяти з бекендом. Це забезпечується за допомогою методу enableCors(), де вказуються дозволені методи, домен фронтенду та параметр credentials. Окрім цього, варто інтегрувати Swagger, який є інструментом для автоматичної генерації документації API (рис. 3.11). За допомогою SwaggerModule можна створити інтерфейс, який дозволяє переглядати всі доступні маршрути, їхні методи, приклади запитів і відповідей. Це спрощує процес тестування та розробки застосунку.

```
app.enableCors({
  origin: process.env.FRONTEND_URL || 'http://localhost:5173',
  methods: 'GET,POST,PUT,DELETE',
  credentials: true,
});

const swaggerConfig = new DocumentBuilder()
  .setTitle('Schulte API')
  .setVersion('1.0')
  .addBearerAuth()
  .build();

const document = SwaggerModule.createDocument(app, swaggerConfig);
SwaggerModule.setup('api', app, document);
```

Рисунок 3.11 – Активація підтримки CORS та інтеграція Swagger

Після налаштування базової архітектури необхідно реалізувати модуль авторизації, який відповідатиме за реєстрацію, вхід користувача, а також перевірку його доступу до захищених ресурсів системи.

У межах AuthModule створюється контролер AuthController (рис. 3.12), який містить три основні маршрути:

– реєстрація (POST /auth/register);
– вхід (POST /auth/login);
– перевірка профілю (GET /auth/profile).

```typescript
@ApiTags('Auth')
@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Post('register')
  async register(@Body() dto: AuthDto) {
    return this.authService.register(dto.email, dto.password);
  }

  @Post('login')
  async login(@Body() dto: AuthDto) {
    return this.authService.login(dto.email, dto.password);
  }

  @Get('profile')
  @UseGuards(AuthGuard)
  @ApiBearerAuth()
  async profile(@User() user) {
    return user;
  }
}
```

Рисунок 3.12 – Код AuthController

Обробку запитів виконує відповідний сервіс AuthService, який реалізує виклики до Supabase через методи signUp і signInWithPassword.

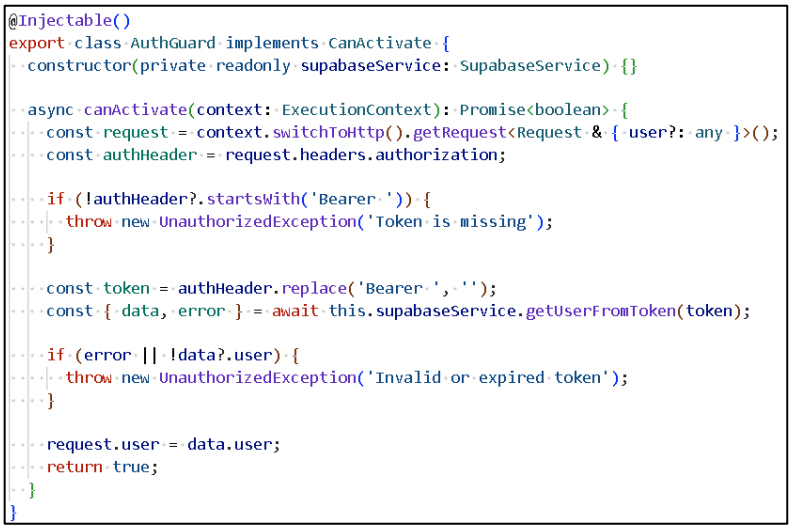
Для структуризації вхідних даних створюється клас AuthDto, який містить поля email та password. За допомогою декораторів з бібліотеки class-validator у DTO додається базова валідація, а саме перевірка на наявність значення, мінімальна довжина пароля, відповідність формату електронної пошти (рис. 3.13).

```typescript
import { IsEmail, IsNotEmpty, MinLength } from 'class-validator';
import { ApiProperty } from '@nestjs/swagger';

export class AuthDto {
  @ApiProperty({ example: 'test@gmail.com' })
  @IsEmail()
  email: string;

  @ApiProperty({ example: 'Pass1234', minLength: 6 })
  @IsNotEmpty()
  @MinLength(6)
  password: string;
}
```

Рисунок 3.13 – Код AuthDto з валідацією за допомогою class-validator

Важливим елементом безпеки є реалізація перевірки доступу до захищених маршрутів. Для цього розробляться AuthGuard (рис. 3.14), який аналізує заголовок запиту (Authorization: Bearer <token>), витягує токен, передає його до Supabase,

після чого перевіряє валідність. Якщо токен недійсний, маршрут не буде виконано, і система поверне помилку авторизації.



```
@Injectable()
export class AuthGuard implements CanActivate {
  constructor(private readonly supabaseService: SupabaseService) {}

  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest<Request & { user?: any }>();
    const authHeader = request.headers.authorization;

    if (!authHeader?.startsWith('Bearer ')) {
      throw new UnauthorizedException('Token is missing');
    }

    const token = authHeader.replace('Bearer ', '');
    const { data, error } = await this.supabaseService.getUserFromToken(token);

    if (error || !data?.user) {
      throw new UnauthorizedException('Invalid or expired token');
    }

    request.user = data.user;
    return true;
  }
}
```

Рисунок 3.14 – Реалізація AuthGuard

Основною задачею серверної частини є збереження результатів проходження тесту Шульте, що включає фіксацію параметрів сесії та детальних даних про натискання користувача на цифри. Для цього потрібно створити окремий модуль SessionsModule, який реалізує відповідну логіку. Цей модуль як і всі інші матиме свій SessionsController, що містить такі маршрути:

– створення нової сесії (POST /sessions);
– отримання всіх сесій користувача (GET /sessions);
– перегляду конкретної сесії тестування (GET /sessions/:id);
– видалення сесії (DELETE /sessions/:id).

При створенні сесії клієнт надсилає параметри тесту та масив натискань, які передаватимуться у форматі DTO (CreateSessionDto і CreateClickDto). Сервіс SessionsService спочатку створватиме сесію в таблиці test_sessions, після чого додаватиме натискання до таблиці test_clicks, зв’язуючи їх з відповідною сесією.

Також, потрібно зробити щоб при запиті всіх сесій (GET /sessions) кожна з них також має включати масив пов’язаних натискань, це дозволить клієнту отримати повну інформацію без додаткових запитів (3.15).

```
async findAll(userId: string): Promise<any[]> {
  const { data: sessions, error } = await this.supabase.client
    .from('test_sessions')
    .select('*')
    .eq('user_id', userId);

  if (error) throw error;

  const sessionIds = sessions.map(s => s.id);

  const { data: clicks, error: clicksError } = await this.supabase.client
    .from('test_clicks')
    .select('*')
    .in('session_id', sessionIds);

  if (clicksError) throw clicksError;

  return sessions.map(session => ({
    ...session,
    clicks: clicks.filter(c => c.session_id === session.id),
  }));
}
```

Рисунок 3.15 – Метод отримання натискань всередині сесій

Останій модуль який варто реалізувати це UserAnalyticsModule який забезпечує обробку статистичних даних, що дозволить відображати середній час, кількість помилок, найкращі та найгірші результати. Після кожного завершеного тесту викликатиметься метод updateUserAnalytics() в сервісі UserAnalyticsService, який автоматично аналізуватиме і оновлюватиме статистику.

Контролер, цього модуля UserAnalyticsController реалізує єдиний маршрут GET /user-analytics/:userId, який дозволяє клієнту отримати актуальні аналітичні дані за ідентифікатором користувача.

Реалізувавши всі модулі була отримана наступна діаграма класів (рис. 3.16).

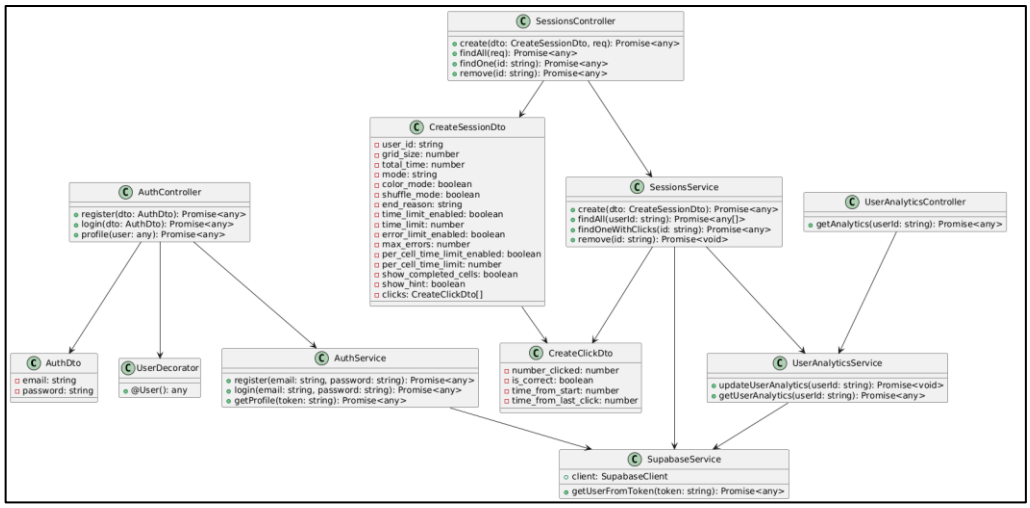


Рисунок 3.16 – Діаграма класів серверної частини системи

3.3 Розробка вебзастосунку

Як і планувалося раніше, клієнтська частина інформаційної системи буде реалізована як односторінковий вебзастосунок (SPA) з використанням бібліотеки React. Такий підхід дозволить забезпечити динамічну зміну вмісту сторінки без її повного перезавантаження, що є важливою вимогою до сучасного інтерфейсу з високим рівнем інтерактивності.

Кожен компонент у React може бути або функціональним, або класовим. У сучасній розробці переважає використання функціональних компонентів з хуками. Компонент приймає вхідні параметри (props), підтримує внутрішній стан (state) і може реагувати на події.

Спершу необхідно створити файлову структуру вебзастосунку, яка забезпечить чіткий поділ логіки, відображення, обробки запитів та стилів (рис. 3.17).

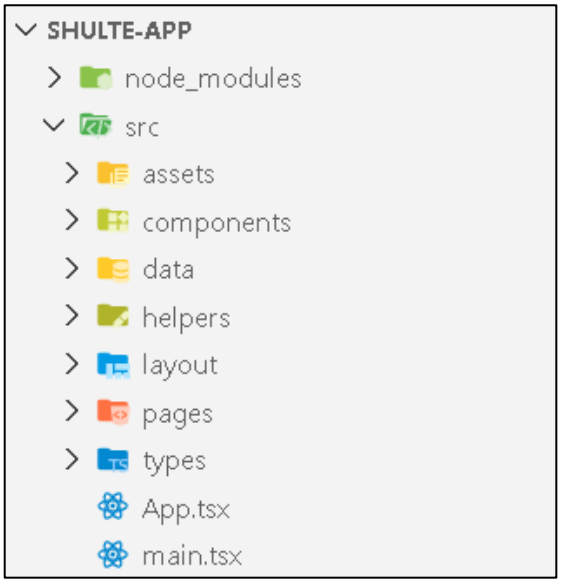


Рисунок 3.17 – Файлова структура вебзастосунку

Також варто розглянути кожен елемент цієї структури окремо:

– node_modules містить всі зовнішні бібіліотеки та залежності;

– src є основною папкою з усім вихідним кодом;

– assets містить ресурси такі як зображення, шрифти та інше;
– components містить повнорювані компоненти інтерфейсу;
– hooks є папкою яка містить файли, що реалізують логіку;
– layout містить структуру сторінок;
– pages безпосередньо містить сторінки;
– services містить файли для взаємодії з API;
– styles містить глобальні стилі;
– types слугує для збереження типів і інтерфейсів;
– utils є папкою для допоміжних функцій і констант;

Також проєкт містить файл App.tsx який є головним компонентом React, з якого починається рендер усіх компонентів, та файл main.tsx, котрий є точкою входу в застосунок.

Для початку необхідно створити Layout, який формуватиме базову структуру сторінок тестування та результатів. Ця сторінка містить шапку в якій знаходитиметься кнопка для виклику форми авторизації, основну частину та піідвал, як і зазначалось при проєктуванні (рис. 3.18).

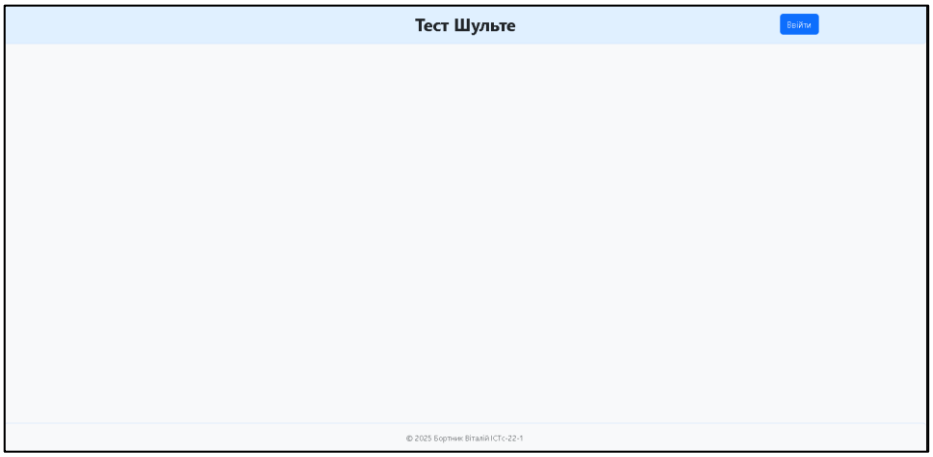


Рисунок 3.18 – Вигляд створеного Layout

Після цього можна перейти до створення сторінки тестування, яка складатиметься з самої таблиці Шульте та панелі, яка використовуватиметься як для налаштування тестування так і для відображення ходу тестування.

Перш за все необзідно розробити панель, яка потім буде імпортована на сторінку тестування (рис. 3.19).

```
return (
  <div className="bg-white p-4 rounded shadow-sm" style={{ maxWidth: '700px', margin: 'auto' }}>
    <div className="row justify-content-center mb-3 text-center">
      <div className="col-auto form-check">…
      </div>
      <div className="col-auto form-check">…
      </div>
      <div className="col-auto form-check">…
      </div>
    </div>
    <div className="row justify-content-center mb-3 gx-3 gy-2 text-start">…
    </div>
    <div className="col-auto d-flex align-items-center flex-wrap gap-2">…
    </div>
    <div className="text-center mt-4">
      <button className="btn btn-primary btn-sm" onClick={onStart}>Почати тест</button>
    </div>
  </div>
);
```

Рисунок 3.19 – Код для реалізації панелі

В результаті буде отриманий інтерфейс панелі зображений на рисунку 3.20.

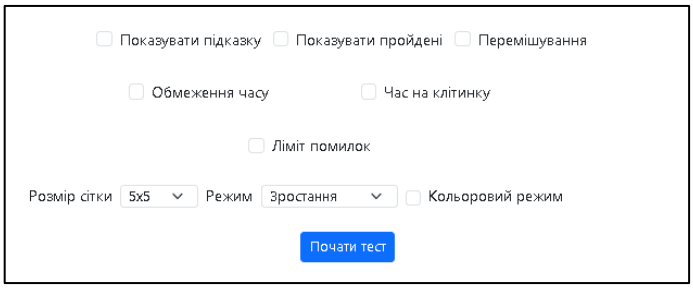


Рисунок 3.20 – Вигляд панелі налаштувань

Теж необхідно написати код для генерацііі таблиці Шульте (рис. 3.21).

```
const Grid: React.FC<Props> = ({
  grid,
  gridSize,
  current,
  onCellClick,
  showCompletedCells = true,
  highlightCurrent = true,
  colors = {},
}) => {
  return (
    <div
      className="grid-container"
      style={{…
      }}
    >
      {grid.map((num) => {
        const isCompleted = showCompletedCells && (
          (current > num && current > 1) || (current < num && current < grid.length)
        );
```

Рисунок 3.21 – Код генерації таблиці Шульте

Після написання коду основних складових сторінки потрібно імпортувати їх безпосередньо в файл сторінки тестування TestPage.tsx. Також на цій сторінці необхідно реалізувати логіку взаємодії користувача з інтерфейсом для процесу тестування, в результаті чого буде отримано сторінку яка містить весь необхідний функціонал (рис. 3.22).

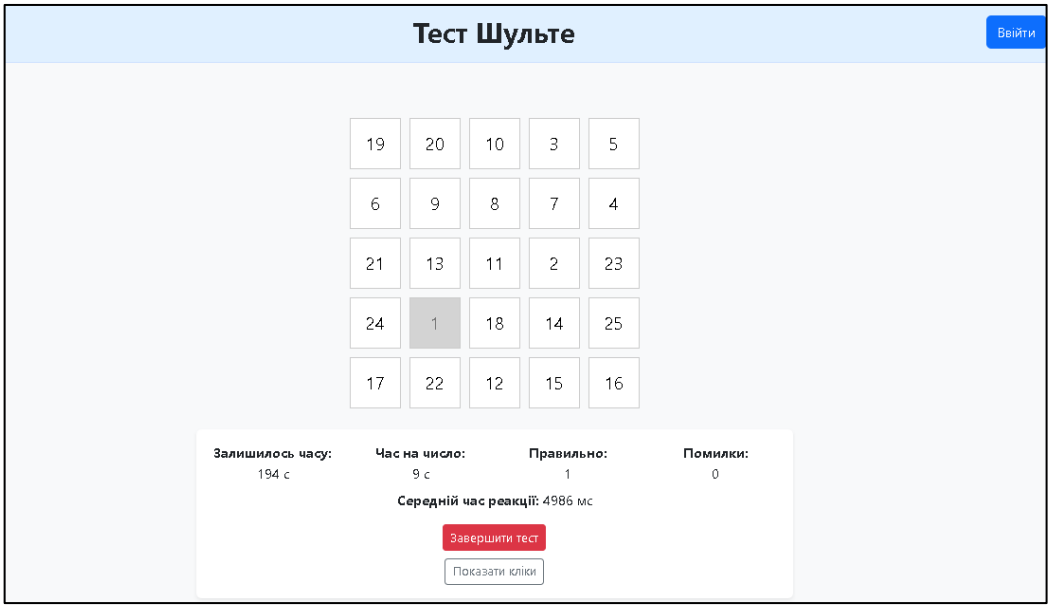


Рисунок 3.22 – Вигляд реалізованої сторінки тестування

Наступним кроком буде створення форми для авторизації, на якій можна як ввійти в існуючий обліковий запис, так і зареєструватись в системі. Варто зазначити що при розробці потрібно врахувати валідацію даних в полях вводу (рис. 3.23).

```
const LoginForm: React.FC<LoginFormProps> = ({ onLoginSuccess }) => {
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [isRegistering, setIsRegistering] = useState(false);
    const [error, setError] = useState<{ email?: string, password?: string, general?: string }>({});

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

        const newError: { email?: string, password?: string, general?: string } = {};

        if (!email || !password) {
            newError.email = 'Емейл та пароль обов\'язкові';
        }

        const passwordRegex = /^(?!.*\s)(?=.*[a-zA-Z]).{6,}$/;
        if (!passwordRegex.test(password)) {
            newError.password = 'Пароль повинен містити хоча б 6 символів, не містити пробілів і мати латинські літери';
        }
```

Рисунок 3.23 – Фрагмент коду форми авторизації

Результатом написання коду, буде повноцінна форма авторизації (рис. 3.24)

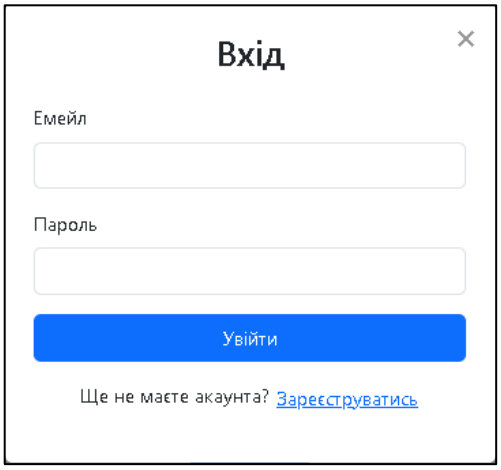


Рисунок 3.24 – Вигляд форми авторизації

Результати проходжень тестування авторизованими користувачами повинні, передаватися на сервер для збереження, отже необхідно реалізувати сервіс (рис. 3.25).

```
export const saveTestResults = async (data: any, token: string) => {
  try {
    const response = await fetch('http://localhost:3000/sessions', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      console.error('Error saving results:', response.statusText);
      alert('Помилка при збереженні результату.');
    } else {
      const result = await response.json();
      console.log('Збережено:', result);
    }
  } catch (error) {
    console.error('Error saving results:', error);
    alert('Сервер недоступний або сталася помилка мережі.');
  }
};
```

Рисунок 3.25 – Фрагмент коду сервісу для роботи з результатами

Отримання результатів так само як і їх предача здійснюватиметься за допомогою цього сервісу.

Далі так як і сторінку тестування, засобами React необхідно реалізувати сторінку результатів, яка міститиме аналітику результатів тестувань користувача.

Ця сторінка складатиметься з блоку загальної статистики, таблиці резуль татів з можливістю фільтрування та сортування, а також діаграм які візуалізуватимуть результати. Отримання результатів так само як і їх предача здійснюватиметься за допомогою сервісу. Після реалізації, сторінка результатів матиме вигляд зображений на рисунку 3.26.

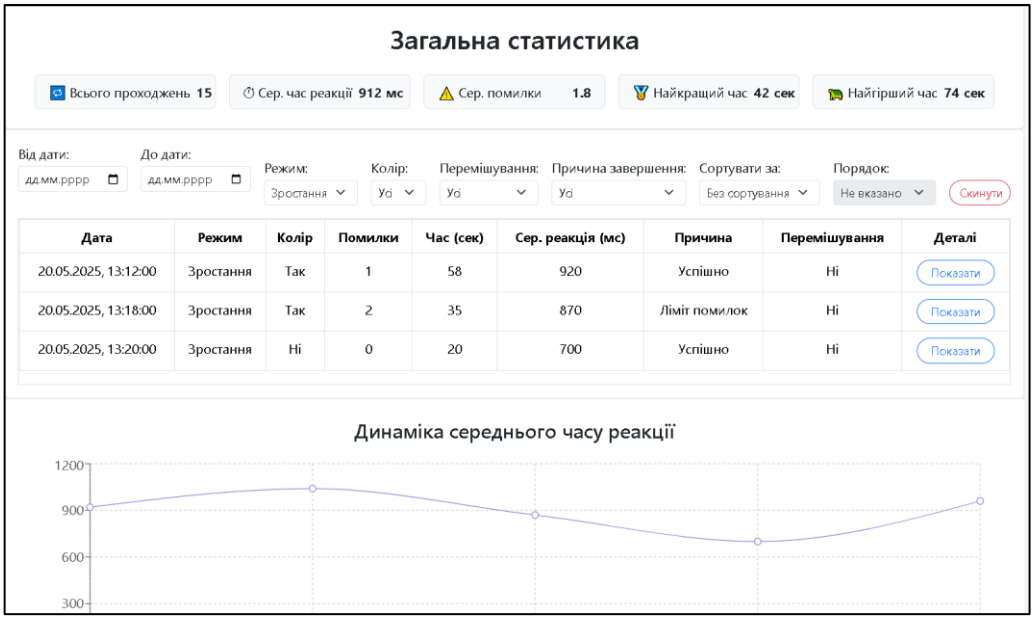


Рисунок 3.26 – Фрагмент сторінки результатів

Завершальним етапом буде налаштування всіх переходів між сторінками, яке відбувається у файлі App.tsx, де кожну сторінку потрібно розмістити в блоці <Routes>, вказавши path, тобто шлях, та element – сторінку на яку відбуватиметься перехід (рис. 3.27).

```
function App() {
  return (
    <Router>
      <Layout>
        <Routes>
          <Route path="/" element={<TestPage />} />
          <Route path="/results" element={<AnalyticsPage />} />
        </Routes>
      </Layout>
    </Router>
  );
}
```

Рисунок 3.27 – Налаштування переходів між сторінками

3.4 Тестування системи

Тестування є невід'ємною частиною процесу розробки інформаційної системи та дозволяє виявити помилки, оцінити стабільність і підтвердити коректність реалізації основних функцій.

Оскільки архітектура розробленої системи передбачає чіткий поділ на клієнтську та серверну частину саме ці компоненти й потрібно протестувати.

Серверна частина інформаційної системи реалізована за допомогою фреймворку NestJS, який забезпечує модульну структуру, підтримку REST-архітектури та зручну інтеграцію з базою даних Supabase. Ця частина системи відповідає за обробку HTTP-запитів від клієнтської частини, виконання бізнес-логіки, авторизацію користувачів, збереження результатів проходження тесту Шульте та взаємодію з базою даних. З метою спрощення тестування і перевірки функціональності API, як і зазначалося раніше, до серверної частини було інтегровано Swagger UI. Це дозволить здійснювати тестування кінцевих точок API безпосередньо через браузер, перевіряючи правильність формування запитів, обробки даних та отримання відповідей у візуальному форматі (рис. 3.28).

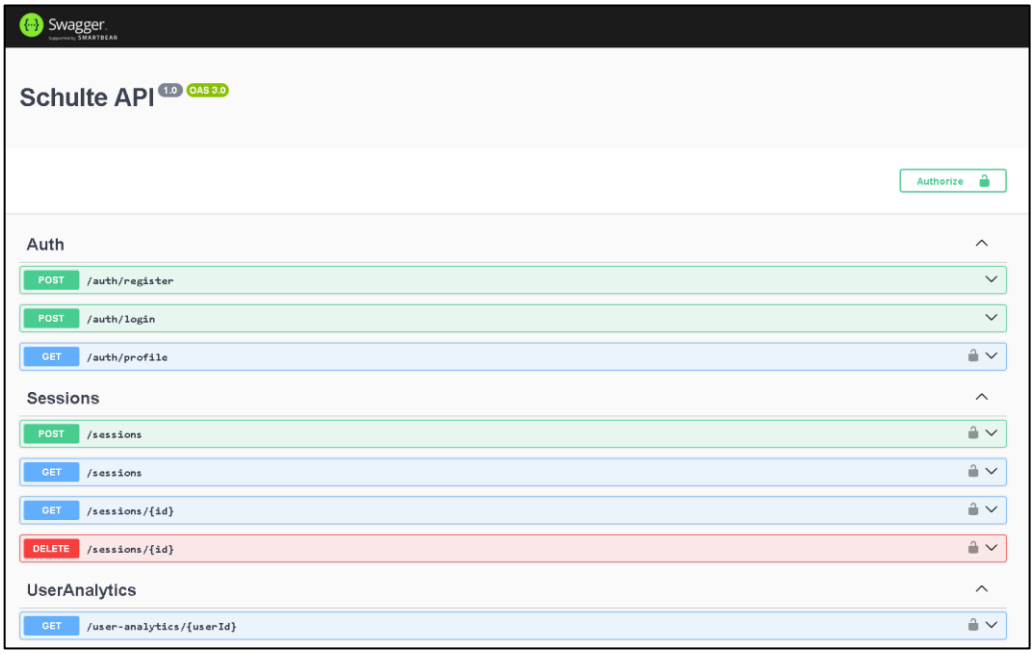


Рисунок 3.28 – Загальний вигляд інтерфейсу Swagger

Як можна помітити, інтерфейс Swagger UI містить кілька логічно згрупованих наборів маршрутів, кожен із яких відповідає певному функціональному блоку серверної частини системи. Зважаючи на це, варто провести тестування в кожній з цих груп.

Першою групою є Auth, який забезпечує функціональність реєстрації та авторизації користувачів. Група включає маршрути для створення нового облікового запису, входу до системи та отримання профілю авторизованого користувача.

Перш за все необхідно здійснити перевірку маршруту реєстрації користувача. У Swagger UI для цього слід розгорнути відповідний маршрут POST /auth/register, після чого відкривається інтерфейс з полем Request body та кнопкою Execute. У полі Request body відображається JSON-структура, яку потрібно заповнити актуальними даними для створення нового облікового запису, зокрема email, password, а також інші можливі поля залежно від конфігурації. Після заповнення структури потрібно натиснути кнопку Execute, що призведе до надсилання запиту на сервер (рис. 3.29).

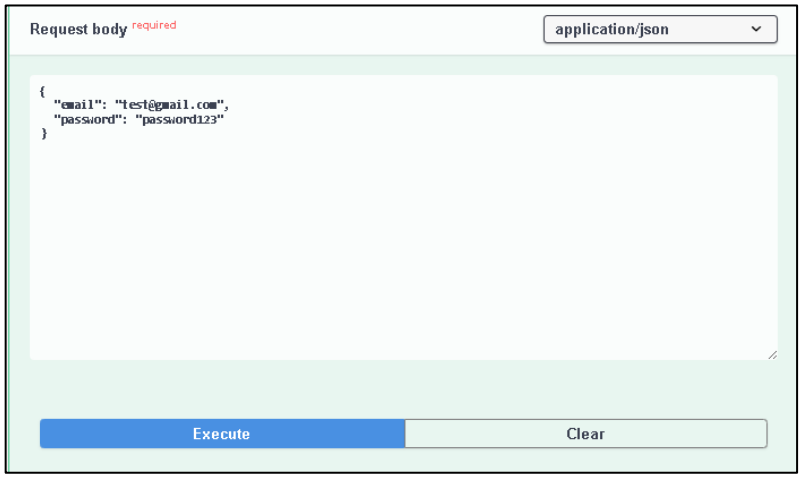


Рисунок 3.29 – Приклад заповнення тіла POST запиту

Після надсилання у нижній частині вікна відображається відповідь сервера у вигляді HTTP-коду (в даному випадку 201 Created) та тіла відповіді у форматі JSON, яке містить дані створеного облікового запису (рис. 3.30).

Рисунок 3.30 – Відповідь сервера на POST запит реєстрації

За допомогою цього можна переконатися, що сервер коректно обробляє вхідні дані та створює нового користувача згідно з очікуваною логікою.

Після успішної реєстрації, так само як і під час входу в систему через маршрут POST /auth/login, користувач отримує токен доступу (JWT). Цей токен необхідний для здійснення запитів до захищених маршрутів, які містять дані, пов'язані з конкретним користувачем. Для подальшого тестування авторизованих запитів у Swagger UI цей токен слід вручну передати до системи. Для цього в інтерфейсі Swagger необхідно натиснути кнопку Authorize у верхньому правому куті. У вікні, що відкриється, ввести отриманий токен у відповідне поле (рис. 3.31).

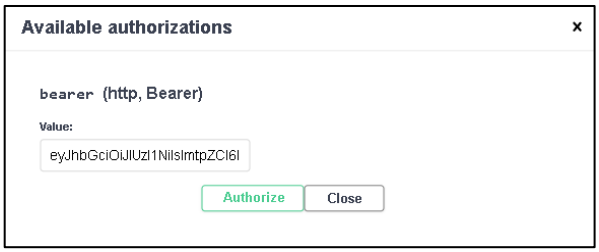


Рисунок 3.31 – Авторизація за допомогою JWT в Swagger

Наступна група маршрутів, яку варто протестувати після успішної авторизації, це Sessions, оскільки саме вона відповідає за створення, перегляд та видалення сесій проходження тесту Шульте. Так як токен авторизації вже було передано через кнопку Authorize, можна безперешкодно виконувати запити до захищених маршрутів цієї групи.

Наприклад, для перевірки коректності отримання даних про створені сесії, достатньо розгорнути маршрут GET /sessions та натиснути кнопку Execute. У відповідь сервер повертає список сесій, пов’язаних із поточним користувачем. Якщо токен є валідним, відповідь міститиме масив об’єктів із детальною інформацією про кожну сесію. Цей запит дозволяє переконатися, що система правильно фільтрує сесії відповідно до користувача та успішно зчитує інформацію з бази даних Supabase (рис. 3.32).

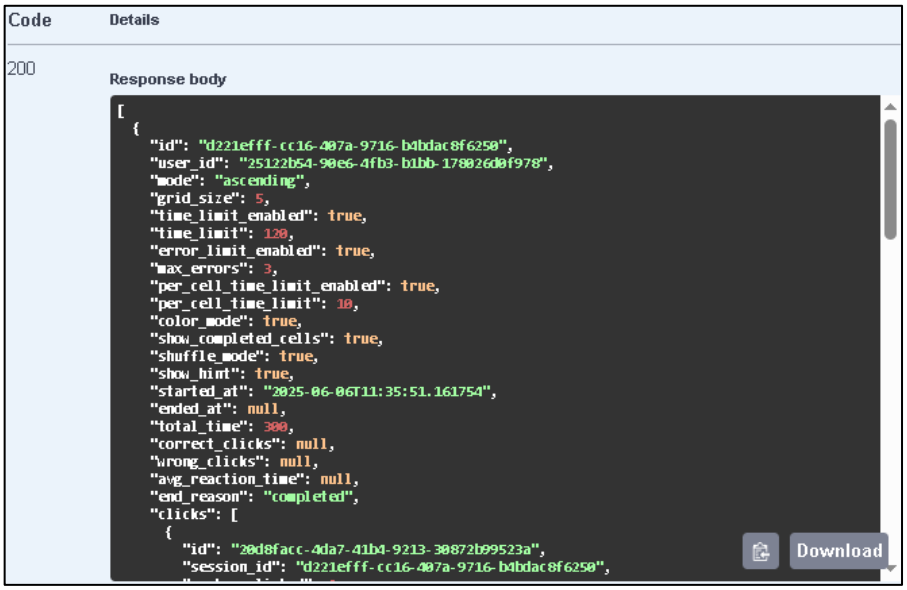


Рисунок 3.32 – Відповідь сервера на GET запит сесій

Крім перегляду списку сесій, також можна протестувати функціональність видалення сесії, яка реалізована через маршрут DELETE /sessions/{id}. Для цього необхідно вказати ідентифікатор сесії, яку потрібно видалити, у відповідному полі, що з’являється після розгортання маршруту (рис. 3.33).

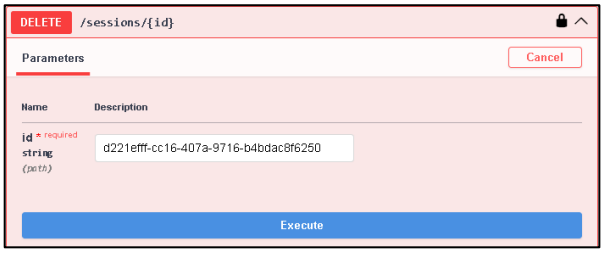


Рисунок 3.33 – Приклад тестування DELETE запиту

Після натискання кнопки Execute сервер виконає запит і, у разі успішної авторизації та наявності сесії з таким ідентифікатором, повертає відповідь зі статусом 200 OK свідчить про успішне видалення.

Також важливо протестувати маршрут отримання аналітичних даних користувача, який реалізовано через GET /user-analytics/{userId}. Цей запит дозволяє отримати узагальнену статистику на основі усіх завершених сесій, пов'язаних із конкретним користувачем. Для тестування достатньо ввести у відповідне поле значення userId, яке можна отримати після авторизації (наприклад, із відповіді на POST /auth/login або GET /auth/profile), та натиснути Execute (рис. 3.34).

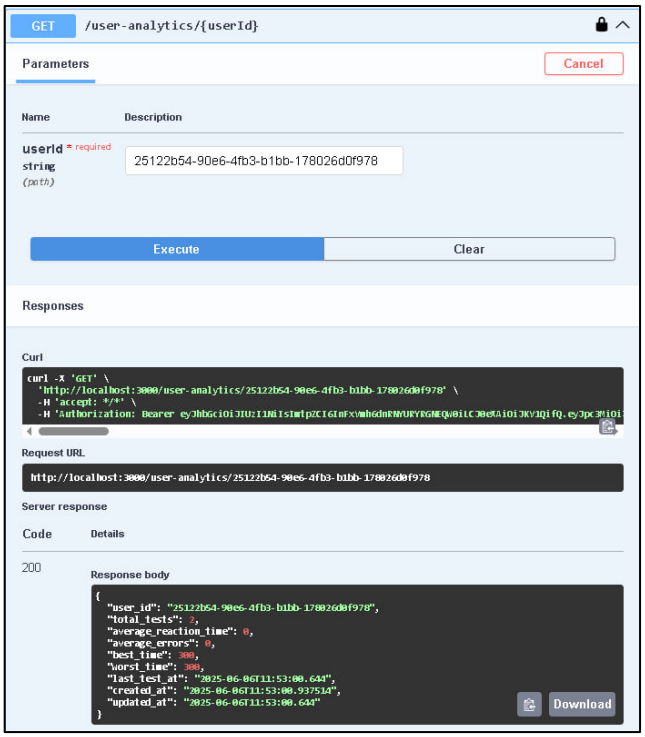


Рисунок 3.34 – Тестування запиту на отримання аналітики

При наявності токена авторизації та наявності даних у базі, у відповідь сервер повертає структуру, що містить потрібну інформацію.

Крім тестування серверної частини, важливо також провести тестування вебзастосунку, оскільки саме він забезпечує інтерактивну взаємодію користувача з системою.

Тестування вебзастосунку проводитиметься мануально, тобто вручну, без використання автоматизованих інструментів.

Мануальне тестування – це процес перевірки функціональності, зручності та стабільності роботи програмного забезпечення шляхом безпосередньої взаємодії з інтерфейсом користувача. Під час такого тестування тестувальник самостійно виконує певні дії в системі, спостерігає за її поведінкою та фіксує всі виявлені помилки або відхилення від очікуваного результату.

Перш за все потрібно протестувати інтерфейс тестування. Необхідно протестувати коректність відображення ключових елементів тесту Шульте, а саме формування сітки з числами відповідно до вибраного розміру, реєстрацію помилок та правильних натискань та середнього часу реакціх і коректність роботи таймерів.

Після запуску тестування був заданий розмір сітки 5x5, режим тестування за зростанням, відображення вже натиснутих чисел, обмеження часу в 250 секунд, ліміт помилок 5 та час на натискання однії клітинки 15 секунд. Як можна помітити (рис. 3.35), інтерфейс коректно відображає всі елементи, та реєструє у відповідних елементах дії користувача.

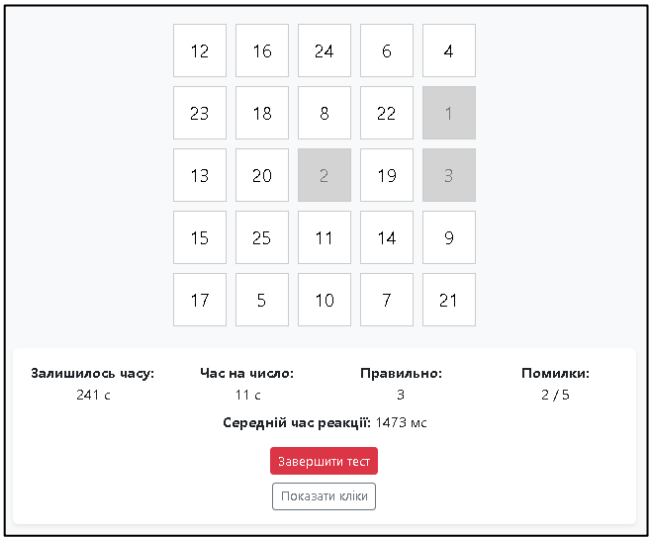


Рисунок 3.35 – Тестування інтерфейсу проходження тесту

Також важливо провести тестування механізму авторизації у вебзастосунку, оскільки він є критичною частиною системи доступу до персоналізованого

функціоналу. Зокрема, необхідно перевірити наявність і коректність валідації полів введення під час входу в систему, а також реакцію інтерфейсу у випадку помилкових чи неповних даних.

Для цього в поле електронної адреси буде введено некоректні дані, а в поле паролю – пароль що складається з кількох символів. В результаті інтерфейс відреагував відображенням підказок, що свідчить про коректну роботу валідації (рис. 3.36).

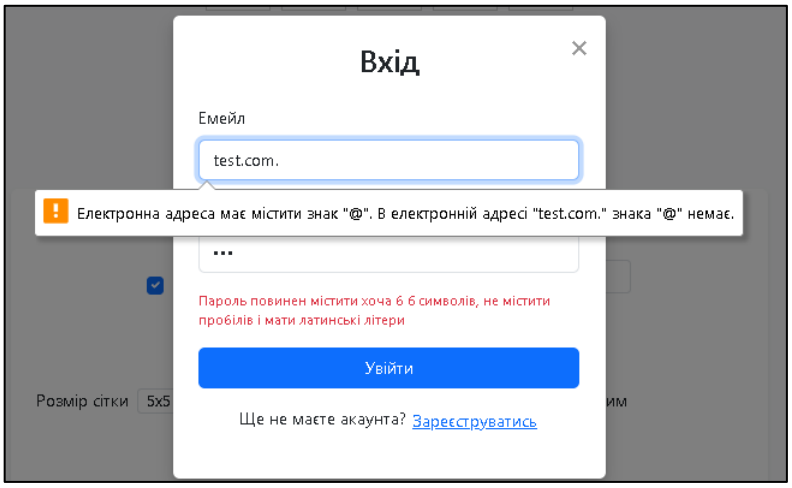


Рисунок 3.36 – Тестування інтерфейсу проходження тесту

Загалом у процесі мануального тестування також було перевірено роботу інших елементів навігації та взаємодії з інтерфейсом вебзастосунку. Усі елементи інтерфейсу реагували на дії користувача згідно з очікуваною логікою, без збоїв, затримок чи візуальних помилок. Жодних критичних або функціональних помилок у роботі клієнтської частини виявлено не було, що свідчить про стабільність та надійність реалізованого інтерфейсу.

3.5 Висновки до третього розділу

У третьому розділі було здійснено практичну реалізацію інформаційної системи. Перш за все було створено структуру бази даних на платформі Supabase, яка забезпечує зберігання даних про користувачів, сесії проходження тесту та

супутні дані. Важливою частиною реалізації стало налаштування системи авторизації з використанням JWT-токенів, що дозволяє гарантувати контрольований доступ до персоналізованих даних.

На серверному рівні розроблено повноцінне API за допомогою фреймворку NestJS, яке включає основні маршрути для створення та отримання сесій тестування, отримання аналітичних даних, а також авторизації. Архітектура серверної частини розроблена у модульному стилі, що сприяє зручному масштабуванню та підтримці коду.

В клієнтській частині створено сучасний вебзастосунок, який дозволяє користувачу проходити тест Шульте в інтерактивному середовищі з гнучким налаштуванням параметрів тесту. Інтерфейс побудований із дотриманням принципів UX/UI-дизайну і забезпечує зручність у користуванні. Реалізовано повну інтеграцію з API, завдяки чому тестові результати передаються на сервер.

Завершальним етапом стало тестування системи, за допомогою якого було перевірено роботу всіх її частин.

ВИСНОВКИ

У дипломні роботі, за результатами виконаних теоретичних та практичних досліджень, було розроблено повнофункціональну інформаційну систему для психодіагностики з використанням тесту Шульте, що охоплює аналіз предметної області та реалізацію вебзастосунку з клієнт-серверною архітектурою.

У першому розділі проведено аналіз методики таблиці Шульте, визначено її місце в сучасній психодіагностиці та можливості застосування в різних сферах. Був проведений огляд існуючих цифрових реалізацій цього тесту, на основі чого було сформовано вимоги до функціоналу та структури майбутньої системи. Це дозволило чітко визначити завдання, які має вирішувати програмний продукт.

У другому розділі здійснено поетапне проєктування інформаційної системи. Було обґрунтовано вибір трирівневої клієнт-серверної архітектури, SPA-підходу на основі фреймворку React, а також модульної структури серверної частини, реалізованої засобами NestJS. Проведено вибір технологічного стеку, побудовано необхідні діаграми, що дозволило створити основу для реалізації системи.

У третьому розділі реалізовано серверну частину з повноцінним API, організовано взаємодію з базою даних Supabase, впроваджено систему авторизації через JWT, а також створено клієнтський вебзастосунок для проведення тесту Шульте. Система підтримує налаштування тесту, обробку результатів, збереження даних та їх подальший аналіз. Завершальним етапом стало тестування роботи системи, що підтвердило її працездатність і відповідність поставленим вимогам.

Виконання дипломної роботи дало змогу поглибити практичні навички в галузі сучасної веброзробки, архітектурного проєктування інформаційних систем і взаємодії з хмарними сервісами. Цей досвід став цінним етапом професійного становлення та надав чудову основу для подальшого розвитку у сфері інформаційних технологій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Агапова С. Внесок Еріка Берна у розвиток сучасної психології. *Видатні постаті психології: історія, сучасність і перспективи*. 2022. № 4. С. 18–21. URL: https://sci.ldubgd.edu.ua/bitstream/123456789/10415/1/Наукові%20праці%20курсантів%20та%20студентів%202022.pdf (дата звернення:14.04.2025).
2. UCL has a racist legacy, but can it move on? URL: https://www.theguardian.com/education/2020/aug/02/ucl-has-a-racist-legacy-but-can-it-move-on (дата звернення: 14.04.2025).
3. The Army Alpha: How WWI Shaped Intelligence Testing. URL: https://www.cogn-iq.org/blog/the-army-alpha-how-wwi-shaped-intelligence-testing (дата звернення: 15.04.2025).
4. Group Examination Alpha. URL: https://americanhistory.si.edu/collections/object/nmah_1213725 (дата звернення: 15.04.2025).
5. Srivastava, A., Farahmand, V., Babaei, A. et al. A benchmark dataset for virtual machine placement in cloud computing. *Scientific Data*. 2024. Vol. 11. Article 372.
6. Shulte Tables. URL: https://brainapps.io/science/task/shulte (дата звернення: 16.04.2025).
7. Фото класичної таблиці Шульте. URL: https://vikna.tv/wp-content/uploads/2022/04/15/tablyczya-shulte.jpg (дата звернення: 16.04.2025).
8. Schulte table: brain training 1.1.23. URL: https://schulte-tables-speed-reading-attention-training.soft112.com (дата звернення: 17.04.2025).
9. Open Schulte table3. URL: https://open-schulte-table.soft112.com (дата звернення: 17.04.2025).
10. Kowalski, K., Wierzbicki, T. Drivers' Psychomotor Reaction Times Tested with a Test Station Method. *Sensors*. 2021. Vol. 21(5). P. 1749. URL https://www.researchgate.net/publication/349939982_Drivers%27_Psychomotor_Reaction_Times_Tested_with_a_Test_Station_Method (дата звернення: 17.04.2025).

11. Lachowicz M., Żurek A., Jamro D., et al. Changes in concentration performance and alternating attention after short-term virtual reality training in E-athletes: a pilot study. *Scientific Reports*. 2024. Vol. 14. Article 8904. URL: https://www.nature.com/articles/s41598-024-59539-w (дата звернення: 18.04.2025).

12. Maggio, M. G., De Bartolo, D., Calabrò, R. S., et al. Computer-assisted cognitive rehabilitation in neurological patients: state-of-art and future perspectives. *Frontiers in Neurology*. 2023. Vol. 14. Article 1255319. URL: https://www.frontiersin.org/journals/neurology/articles/10.3389/fneur.2023.1255319/full (дата звернення: 18.04.2025).

13. Що таке десктопні додатки: визначення, переваги та тренди. URL: https://wezom.com.ua/ua/blog/desktop-dodatok (дата звернення: 19.04.2025).

14. Різниця між гібридними та нативними мобільними додатками. URL: https://wezom.com.ua/ua/blog/chem-otlichajutsja-nativnoe-i-gibridnoe-mobilnye-prilozhenija (дата звернення: 19.04.2025).

15. Advantages and Disadvantages of Web Applications. URL: https://highzeal.com/blog/advantages-and-disadvantages-of-web-applications (дата звернення: 19.04.2025).

16. Таблиці Шульте – Швидкочитання. URL: https://play.google.com/store/apps/details?id=com.yurkap.fastreading&hl=uk (дата звернення: 20.04.2025).

17. Фото десктопної програми Open Schulte Table. URL: https://windows-cdn.softpedia.com/screenshots/Open-Schulte-table_1.png (дата звернення: 20.04.2025).

18. Functional vs. Non Functional Requirements. URL: http://geeksforgeeks.org/functional-vs-non-functional-requirements (дата звернення: 21.04.2025).

19. A Guide to Functional Requirements (with Examples). URL: https://www.nuclino.com/articles/functional-requirements (дата звернення: 21.04.2025).

20. Нефункціональні вимоги: приклади, типи, підходи. URL: https://training.qatestlab.com/blog/technical-articles/non-functional-requirements-examples-types-approaches (дата звернення: 22.04.2025).

21. The Software Architect: Demystifying 18 Software Architecture Patterns. URL: https://medium.com/%40amitvsolutions/the-software-architect-deep-dive-into-17-essential-design-patterns-a0fb5a4726ab (дата звернення: 23.04.2025).

22. Monolithic Architecture – System Design. URL: https://www.geeksforgeeks.org/monolithic-architecture-system-design (дата звернення: 24.04.2025).

23. Understanding Monolithic Architecture: Software Design Architecture. URL: https://medium.com/%40madhurajayashanka/understanding-monolithic-architecture-software-design-architecture-ce3be4e2146d (дата звернення: 24.04.2025).

24. Моноліт проти Мікросервісів: виявлення, яка архітектура програмного забезпечення найкраще підходить для вашого рішення та бізнесу. URL: https://stfalcon.com/uk/blog/post/monolith-vs-microservices (дата звернення: 25.04.2025).

25. Monoliths vs. Microservices: Breaking Down Software Architectures. URL: https://dev.to/documatic/monoliths-vs-microservices-breaking-down-software-architectures-1keh (дата звернення: 27.04.2025).

26. Про мікросервісну архітектуру. URL: https://foxminded.ua/mikroservisna-arkhitektura (дата звернення: 27.04.2025).

27. What Are the Benefits of a Microservices Architecture? URL: https://www.akamai.com/blog/cloud/benefits-of-a-microservices-architecture (дата звернення: 30.04.2025).

28. 10 disadvantages of microservices you'll need to overcome. URL: https://www.theserverside.com/answer/What-are-some-of-the-disadvantages-of-microservices (дата звернення: 02.05.2025).

29. Client-Server Model. URL: https://www.geeksforgeeks.org/client-server-model (дата звернення: 06.05.2025).

30. Клієнт-серверна архітектура. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture (дата звернення: 07.05.2025).

31. What are HTTP methods? URL: https://blog.postman.com/what-are-http-methods (дата звернення: 07.05.2025).

32. Client Side MVC. URL: https://medium.com/@alishapal/client-side-mvc-89c1e1af8ade (дата звернення: 09.05.2025).

33. MVC Architecture: Simplifying Web Application Development. URL: https://www.ramotion.com/blog/mvc-architecture-in-web-application (дата звернення: 10.05.2025).

34. MVC in Frontend is Dead. URL: https://dev.to/daelmaak/mvc-in-frontend-is-dead-19ff (дата звернення: 10.05.2025).

35. A Comprehensive Guide to Micro Frontend Architecture: Benefits, Terminology, and Best Practices. URL: https://medium.com/@ravisharma50063/a-comprehensive-guide-to-micro-frontend-development-benefits-terminology-and-best-practices-5e61511736dc (дата звернення: 10.05.2025).

36. Exploring Micro Frontend Architecture: A Comprehensive Guide to Benefits and Best Practices. URL: https://softjourn.com/insights/micro-frontend-architecture (дата звернення: 10.05.2025).

37. Microfrontend architecture: Advantages and disadvantages. URL: https://rasidagac.medium.com/micro-frontend-architecture-advantages-and-disadvantages-acf45e20c6c9 (дата звернення: 10.05.2025).

38. Understanding Web Application Architecture. URL: https://medium.com/@prashant.bismani/what-is-web-application-architecture-4f58d57aad36 (дата звернення: 11.05.2025).

39. What are Single Page Apps? A Complete Guide. URL: https://techtinium.com/what-are-single-page-apps (дата звернення: 11.05.2025).

40. A guide to single-page application performance. URL: https://techtinium.com/what-are-single-page-apps (дата звернення: 12.05.2025).

41. Backend: Layered Architecture. URL: https://dev.to/blindkai/backend-layered-architecture-514h (дата звернення: 14.05.2025).

42. Layers in software architecture. URL: https://medium.com/%40sagar.hudge/layers-in-software-architecture-c8cc16329ff6 (дата звернення: 14.05.2025).

43. Developing modular software: Top strategies and best practices. URL: https://vfunction.com/blog/modular-software (дата звернення: 14.05.2025).

44. What is a relational database? URL: https://www.ibm.com/think/topics/relational-databases (дата звернення: 15.05.2025).

45. Normal Forms in DBMS. URL: https://www.geeksforgeeks.org/normal-forms-in-dbms (дата звернення: 15.05.2025).

46. What is NoSQL? URL: https://www.mongodb.com/resources/basics/databases/nosql-explained (дата звернення: 15.05.2025).

47. Relational vs. Non-Relational Databases: Features and Benefits. URL: https://www.couchbase.com/blog/relational-vs-non-relational-database (дата звернення: 18.05.2025).

48. Building an app? These are the best JavaScript frameworks in 2025. URL: https://www.contentful.com/blog/best-javascript-frameworks (дата звернення: 19.05.2025).

49. Designing Modular and Scalable APIs with NestJS. URL: https://medium.com/%40sachetacharya19/designing-modular-and-scalable-apis-with-nestjs-2c81d617d4d0 (дата звернення: 20.05.2025).

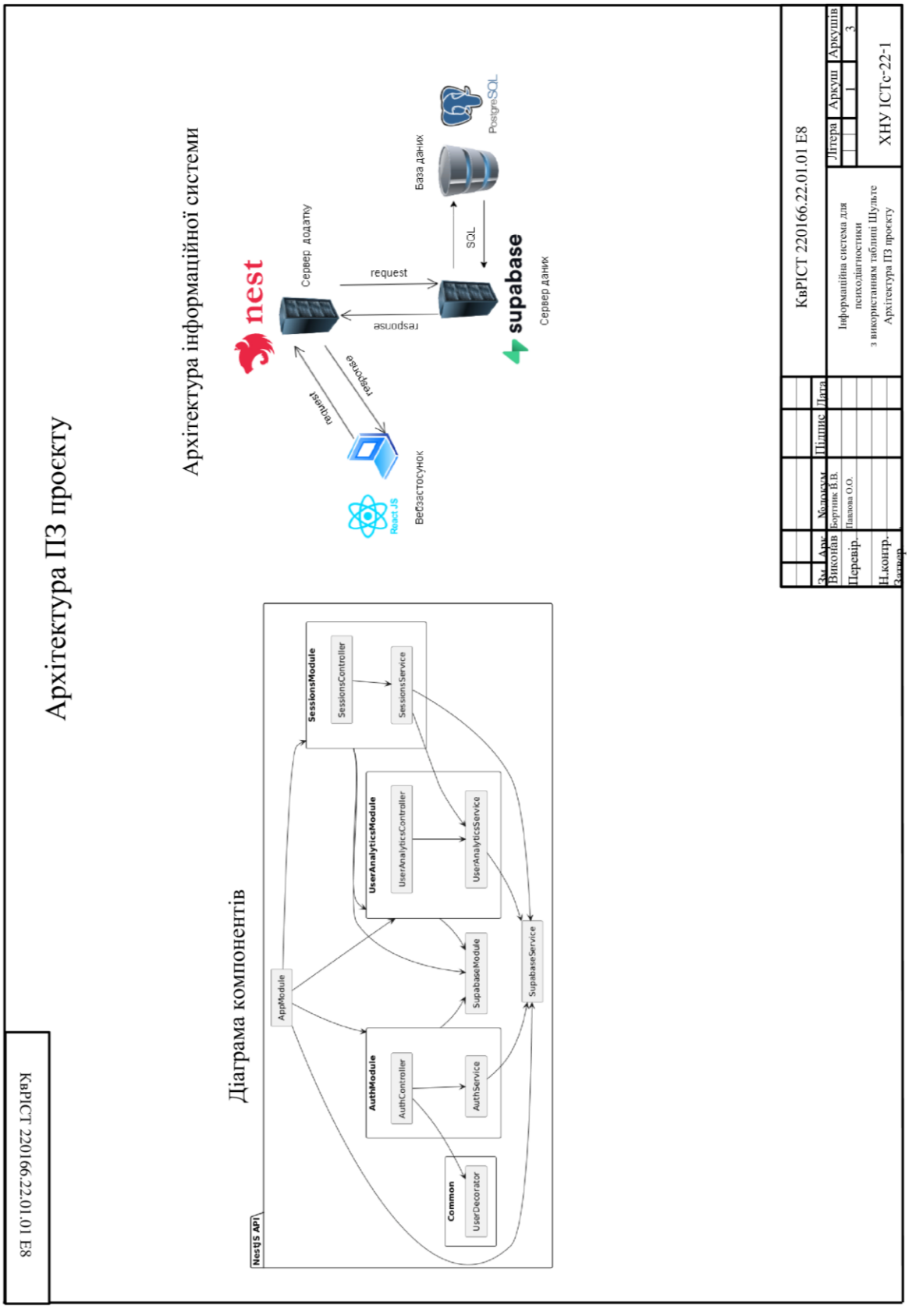
50. Supabase Docs. URL: https://supabase.com/docs (дата звернення: 20.05.2025).

51. 7 Tips for a Good ER Diagram Layout. URL: https://vertabelo.com/blog/vertabelo-tips-good-er-diagram-layout (дата звернення: 20.05.2025).

Додаток А

(обов’язковий)

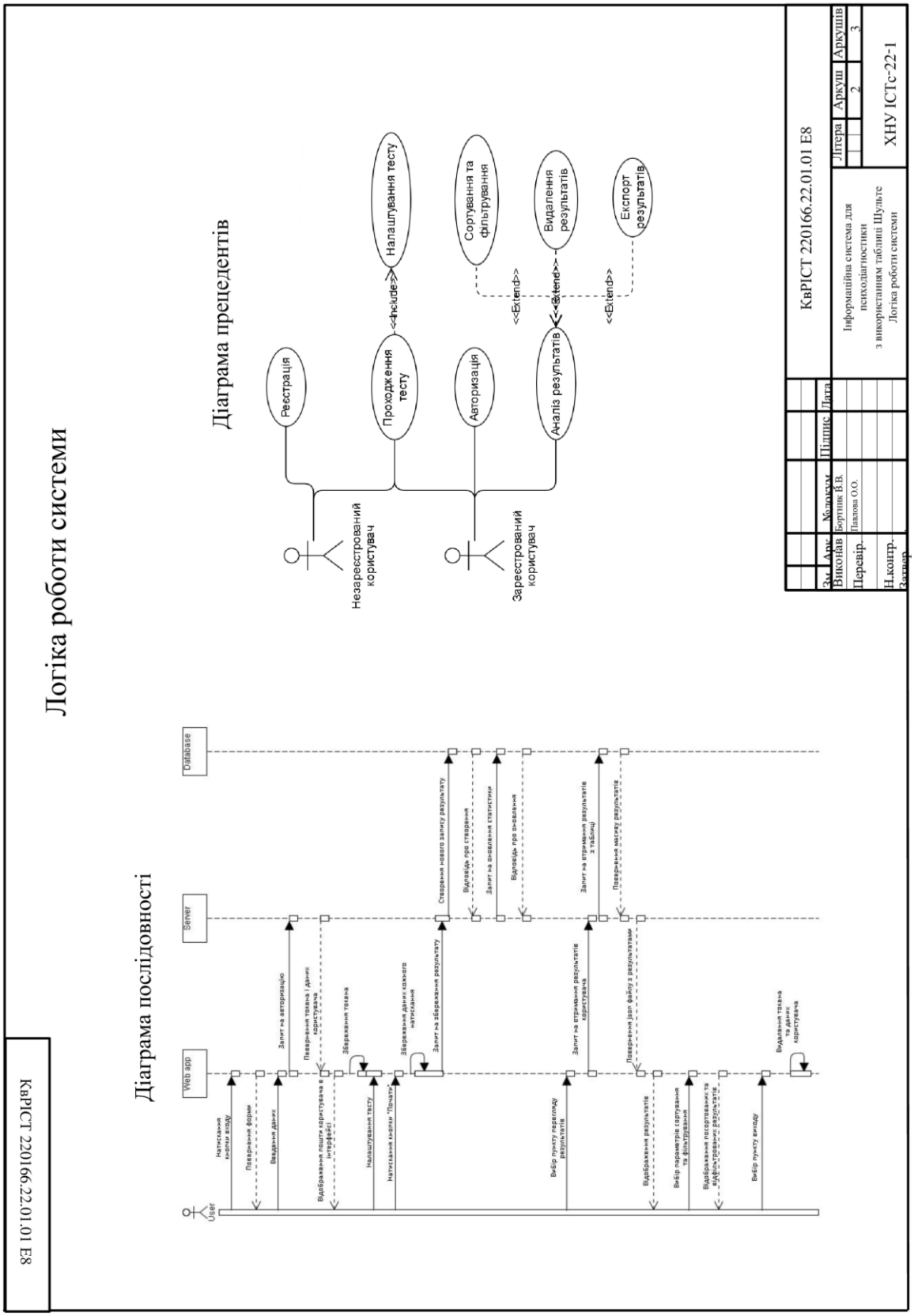
КОПІЯ КРЕСЛЕННЯ «АРХІТЕКТУРА ПЗ ПРОЄКТУ»



Додаток Б

(обов'язковий)

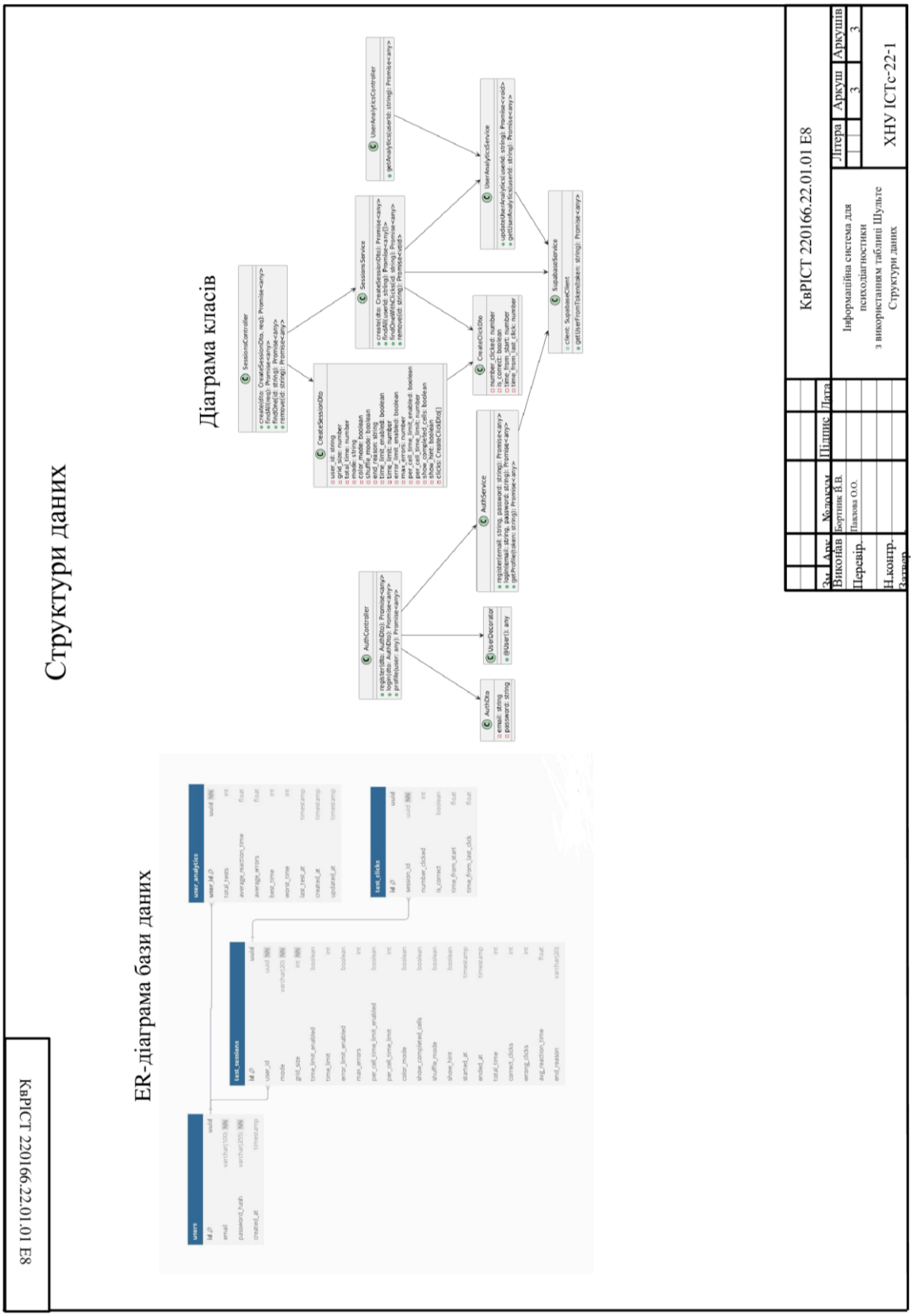
КОПІЯ КРЕСЛЕННЯ «ЛОГІКА РОБОТИ СИСТЕМИ»



Додаток В

(обов’язковий)

КОПІЯ КРЕСЛЕННЯ «СТРУКТУРИ ДАНИХ»



Додаток Г

(обов'язковий)

КОД ПРОГРАМНОГО ЗАБЕЗПЕЧЕНЯ

Код сервісу бази даних

```
import { Injectable } from '@nestjs/common';
import { createClient, SupabaseClient } from '@supabase/supabase-
js';
@Injectable()
export class SupabaseService {
  public client: SupabaseClient;
  constructor() {
    const supabaseUrl = process.env.SUPABASE_URL!;
    const supabaseKey = process.env.SUPABASE_SERVICE_ROLE_KEY!;
    this.client = createClient(supabaseUrl, supabaseKey);
  }
  async getUserFromToken(token: string) {
    return this.client.auth.getUser(token);
  }
}
```

Код сервісу авторизації

```
import { Injectable } from '@nestjs/common';
import { SupabaseService } from '../supabase/supabase.service';
@Injectable()
export class AuthService {
  constructor(private readonly supabaseService: SupabaseService) {}
  async register(email: string, password: string) {
    const { data, error } = await
this.supabaseService.client.auth.signUp({ email, password });
    return { data, error };
  }
  async login(email: string, password: string) {
    const { data, error } = await
this.supabaseService.client.auth.signInWithPassword({ email,
password });
    return { data, error };
  }
  async getProfile(token: string) {
    return this.supabaseService.getUserFromToken(token);
  }
}
```

Код DTO результату тестування

```
import { ApiProperty } from '@nestjs/swagger';
```

```
import { CreateClickDto } from './create-click.dto';
export class CreateSessionDto {
  @ApiProperty() user_id: string;
  @ApiProperty() grid_size: number;
  @ApiProperty() total_time: number;
  @ApiProperty({ enum: ['ascending', 'descending'] }) mode: string;
  @ApiProperty() color_mode: boolean;
  @ApiProperty() shuffle_mode: boolean;
  @ApiProperty({ enum: ['completed', 'time_limit', 'error_limit',
'manual'] }) end_reason: string;
  @ApiProperty() time_limit_enabled: boolean;
  @ApiProperty() time_limit: number;
  @ApiProperty() error_limit_enabled: boolean;
  @ApiProperty() max_errors: number;
  @ApiProperty() per_cell_time_limit_enabled: boolean;
  @ApiProperty() per_cell_time_limit: number;
  @ApiProperty() show_completed_cells: boolean;
  @ApiProperty() show_hint: boolean;
  @ApiProperty({ type: [CreateClickDto] }) clicks: CreateClickDto[];
}
```

Код сервісу результатів

```
import { Injectable } from '@nestjs/common';
import { SupabaseService } from '../supabase/supabase.service';
import { CreateSessionDto } from './dto/create-session.dto';
import { CreateClickDto } from './dto/create-click.dto';
import { UserAnalyticsService } from '../user-analytics/user-
analytics.service';
@Injectable()
export class SessionsService {
  constructor(
    private readonly supabase: SupabaseService,
    private readonly userAnalyticsService: UserAnalyticsService,
  ) {}
  async create(dto: CreateSessionDto): Promise<any> {
    const { clicks, ...sessionData } = dto;
    const { data: session, error: sessionError } = await
this.supabase.client
      .from('test_sessions')
      .insert(sessionData)
      .select()
      .single();
    if (sessionError) throw sessionError;
    const clickInserts = clicks.map((click: CreateClickDto) => ({
      ...click,
      session_id: session.id,
    }));
    const { data: clicksData, error: clicksError } = await
this.supabase.client
      .from('test_clicks')
      .insert(clickInserts);
```

```
    if (clicksError) throw clicksError;
    await
this.userAnalyticsService.updateUserAnalytics(session.user_id);
    return { ...session, clicks: clicksData };
  }
  async findAll(userId: string): Promise<any[]> {
    const { data: sessions, error } = await this.supabase.client
      .from('test_sessions')
      .select('*')
      .eq('user_id', userId);
    if (error) throw error;
    const sessionIds = sessions.map(s => s.id);
    const { data: clicks, error: clicksError } = await
this.supabase.client
      .from('test_clicks')
      .select('*')
      .in('session_id', sessionIds);
    if (clicksError) throw clicksError;
    return sessions.map(session => ({
      ...session,
      clicks: clicks.filter(c => c.session_id === session.id),
    }));
  }
  async findOneWithClicks(id: string): Promise<any> {
    const { data: session, error: err1 } = await
this.supabase.client
      .from('test_sessions')
      .select('*')
      .eq('id', id)
      .single();
    if (err1) throw err1;
    const { data: clicks, error: err2 } = await this.supabase.client
      .from('test_clicks')
      .select('*')
      .eq('session_id', id)
      .order('time_from_start', { ascending: true });
    if (err2) throw err2;
    return { ...session, clicks };
  }
  async remove(id: string): Promise<void> {
    const { error } = await this.supabase.client
      .from('test_sessions')
      .delete()
      .eq('id', id);
    if (error) throw error;
  }
}
```

Код генерації таблиці шульте

```
import React from 'react';
import './Grid.css'; // стилі для сітки
```

```
interface Props {
  grid: number[];
  gridSize: number;
  current: number;
  onCellClick: (number: number) => void;
  showCompletedCells?: boolean;
  highlightCurrent?: boolean;
  colors?: { [num: number]: { bg: string; text: string } };
}

const Grid: React.FC<Props> = ({
  grid,
  gridSize,
  current,
  onCellClick,
  showCompletedCells = true,
  highlightCurrent = true,
  colors = {},
}) => {
  return (
    <div
      className="grid-container"
      style={{
        display: 'grid',
        gridTemplateColumns: `repeat(${gridSize}, 1fr)`,
        gap: '8px',
      }}
    >
      {grid.map((num) => {
        const isCompleted = showCompletedCells && (
          (current > num && current > 1) || (current < num &&
current < grid.length)
        );

        const cellStyle = {
          backgroundColor: colors[num]?.bg || '#ffffff',
          color: colors[num]?.text || '#000000',
          border: '1px solid #ccc',
          padding: '10px',
          textAlign: 'center' as const,
          fontWeight: highlightCurrent && num === current ? 'bold' :
'normal',
          opacity: isCompleted ? 0.4 : 1,
          cursor: 'pointer',
        };

        return (
          <div
            key={num}
            style={cellStyle}
            onClick={() => onCellClick(num)}
          >
```

```
            {num}
          </div>
        );
      })}
    </div>
  );
};

export default Grid;
```

Код сервісу в вебзастосунку

```
export const saveTestResults = async (data: any, token: string) => {
  try {
    const response = await fetch('http://localhost:3000/sessions', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      alert('Помилка при збереженні результату.');
    } else {
      const result = await response.json();
      console.log('Збережено:', result);
    }
  } catch (error) {
    alert('Сервер недоступний або сталася помилка мережі.');
  }
};
```

Завідувачу кафедри КІІС
д-р. філософії, доц. Ользі ПАВЛОВІЙ

Віталія БОРТНИКА
ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи ІСТс-22-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Strike-Plagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

12.06. 2025 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Бортник Віталій Валерійович

Тема: Інформаційна система для психодіагностики з використанням таблиці Шульте

Спеціальність: 126 «Інформаційні системи та технології»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 Кількість сторінок записки 72

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є проєктування та розробка інформаційної системи, що дозволяє проводити психодіагностичне тестування з використанням таблиці Шульте у вебсередовищі.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню і охоплює всі етапи розробки інформаційної системи, від аналізу предметної області до реалізації та тестування.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі детально проаналізовано історичні та сучасні підходи до психодіагностики з використанням таблиці Шульте. Виявлено проблеми існуючих програмних реалізацій. Сформовано функціональні та нефункціональні вимоги до інформаційної системи.

В другому розділі проведено аналіз та обґрунтований вибір трирівневої клієнт-серверної архітектури, односторінкового підходу до розробки вебзастосунку, модульної архітектури серверної частини, а також бази даних Supabase. Спроєктовано ER-діаграму бази даних, діаграму варіантів використання та розроблено прототип майбутнього інтерфейсу.

В третьому розділі реалізовано програмну частину системи, а саме розгорнуто базу даних та налаштовано доступ, розроблено REST API, авторизацію, логіку проходження тесту, а також створено адаптивний і зручний інтерфейс для користувача. Проведено тестування працездатності кожного компонента системи.

4. Позитивні сторони роботи: високий рівень реалізації за допомогою сучасного технологічного стеку, чітка структура та логіка побудови системи, висока практична цінність роботи.

5. Негативні сторони роботи: негативних сторін не виявлено.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена відповідно до чинних вимог ДСТУ. Містить рисунки, а також усі необхідні елементи, такі як зміст, анотацію, висновки, список джерел. Графічна частина структурована.

7. Відгук про роботу в цілому: Робота виконана на високому науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Макаренкін Денис Анатолійович, доцент, к.т.н. кафедри АКІТ&Р

"12" червня 2025 р. _____ (підпис)

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ
КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Інформаційна система для психодіагностики з використанням таблиці Шульте

Автор: Віталій БОРТНИК

Спеціальність: 126 – Інформаційні системи та технології

Освітня програма: освітньо-професійна

Науковий керівник: Дмитро МЕДЗАТИЙ, к.т.н., доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10-40 джерелами на один фрагмент речення;
4) в якості запозичень в окремих місцях системою зафіксовано послідовності чотирьохрозряних двійкових кодів, які є вхідними даними до великої кількості задач і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
5) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 1.5% і адресується до 10 першоджерел; та системою Anti-Plagiarism складає 17%, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи — Дмитро МЕДЗАТИЙ

Гарант ОП — Андрій НІЧЕПОРУК

Завідувач кафедри КІС — Ольга ПАВЛОВА

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Віталій БОРТНИК

Співавтор:

Назва: Бортник_Інформаційна система для психодіагностики з використанням таблиці Шульте

Експерт:

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:1.5%

Коефіцієнт подібності 2:0.4%

Мікропробіли: 6

Заміна букв: 1

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-11 11:07:03.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2025-06-11 Доцент Андрій Нічепорук

Дата експерт

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 17.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 12%**

	Document		Sum coincidence on the DB	
ID: 244917 Title: БКР Інформаційна система для психодіагностики з використанням таблиці Шульте Added in a DB: 2025-06-11 Authors: Віталій БОРТНИК Heads: Дмитро МЕДЗАТИЙ Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	96535	783	17346 (18%)	140 (18%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes
240796	Title: Звіт з ПДП Інформаційна система для психодіагностики з використанням таблиці Шульте Added in a DB: 2025-05-04 Authors: В. В. Бортника Heads: Т.О. Говорущенко Consultants: Opponents:	16264 (17.0%)	129 (16.0%)