Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка інклюзивного вебзастосунку для тайм-менеджменту нейровідмінних людей

Виконав: студент ___4___ курсу
групи ___5ПІ-21б__
спеціальності
__121 – Інженерія програмного забезпечення_____
(шифр і назва напряму підготовки, спеціальності)
Глоба Анна Русланівна
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Бабюк Н. П.
(прізвище та ініціали)

Рецензент:. к.т.н., доц. каф. ОТ Колесник І.С.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ __Романюк_О. Н._
«09» __06_____ 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Глобі Анні Русланівні

1. Тема роботи – розробка інклюзивного вебзастосунку для тайм-менеджменту нейровідмінних людей.

Керівник роботи: Бабюк Н. П., к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи: 2 червня 2025

3. Вихідні дані до роботи: особисті параметри користувача; інформація про події; дані нотаток та планів; обрана кількість «ложок» на день, як показник енергії; режим перегляду календаря; алгоритм взаємодії інтерактивного тамагочі; фокус-сесії за допомогою таймера; середовище розробки – Visual Studio Code; мова розробки – JavaScript, React; база даних – MySQL; платформа для реалізації серверної логіки – Node.js; вихідні дані: збережені події, календар користувача, нотатки, плани, інтерактивний тамагочі, контролювання стану ресурсу, рекомендації по користуванню сервісом, результати виконання тестових сценаріїв.

4. Перелік ілюстративного матеріалу: загальна архітектура системи; блок-схема алгоритму взаємодії користувача із застосунком; UML-діаграма діяльності системи; інтерфейс головної сторінки з індикатором ложок; інтерфейс календаря з подіями та планами; форма створення події з перевіркою ресурсу; вигляд розділу нотаток з кольоровим маркуванням; форма у вигляді модального вікна для створення та редагування нотатки; інтерфейс для отримання та взаємодії із віртуальним тамагочі; інтерфейс із рекомендаціями по використанню тайм-менеджменту, а також

ознайомлення із нейровідмінністю; візуальний дизайн, адаптованого під когнітивні особливості користувачів.

5. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Бабюк Н. П., к.т.н. доцент кафедри ПЗ	24.03.25	01.06.25

6. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного стану систем для тайм-менеджменту та їх можливості використання для нейровідмінних людей	25.03.25 – 01.04.25	вик
2	Розробка інклюзивного інтерфейсу для нейровідмінних людей	01.04.25 – 08.04.25	вик
3	Розробка алгоритмів вебзастосунку	08.04.25 – 22.04.25	вик
4	Розробка модулів вебзастосунку	22.04.25 – 16.05.25	вик
5	Тестування програмного застосунку	16.05.25 – 23.05.25	вик
6	Оформлення матеріалів до захисту БКР	23.05.25 – 30.05.25	вик

Студент _____ **А. Р. Глоба**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **Н. П. Бабюк**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.4’42

Глоба А. Р. Розробка інклюзивного вебзастосунку для тайм-менеджменту нейровідмінних людей, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 71 с.

На укр. мові. Бібліогр. : назв. 31; рис. 44; табл. 6.

Бакалаврська кваліфікаційна робота присвячена розробці інтерактивного вебзастосунку для тайм-менеджменту нейровідмінних людей. У процесі дослідження проаналізовано актуальність теми в контексті зростання обізнаності про нейровідмінність, проведено порівняльний аналіз існуючих систем планування та виявлено їх недоліки. На основі цього сформульовано ключові завдання системи.

Спроєктовано архітектуру вебсистеми з урахуванням адаптивного інтерфейсу та моделі «ресурсних ложок». Розроблено модуль клієнтської частини, що дозволяють створювати, редагувати та переглядати події з різними типами повторюваності, також модуль, що відповідає за створення заміток, модуль, для взаємодії із інтерактивним тамагочі та системою «монет» для покупки аксесуарів для тваринки.

Обґрунтовано вибір технологій для реалізації застосунку — JavaScript, React для фронтенду, Node.js для серверної частини, MySQL як система керування базою даних. У якості середовища розробки використано Visual Studio Code.

Система протестована на відповідність функціональним вимогам.

Ключові слова: тайм-менеджмент, нейровідмінність, РДУГ, вебзастосунок, календар подій, нотатки, енергетичний ресурс, ложки, інтерфейс користувача, тамагочі, інклюзивність, гейміфікація.

ABSTRACT

UDC 004.4’42

Hloba A. R. Development of an inclusive web application for time management of neurodiverse people: bachelor's thesis in the specialty 121 Software Engineering, educational program – software engineering. Vinnytsia: VNTU, 2025. 71 p.

In Ukrainian. Bibliography: 31 titles; Figures: 44; Table 6.

The bachelor’s qualification project is dedicated to the development of an interactive web application for time management tailored to neurodivergent individuals. The research includes an analysis of the topic’s relevance in the context of increasing awareness of neurodiversity, as well as a comparative review of existing planning systems, identifying their limitations. Based on this analysis, the core requirements for the system were formulated.

The architecture of the web application was designed with a focus on an adaptive user interface and the «spoon theory» model of energy management. The client-side includes modules for creating, editing, and viewing events with various repetition types, a module for note-taking, and another for interacting with a virtual pet (Tamagotchi), featuring a coin-based system for purchasing accessories.

The choice of technologies was justified: JavaScript and React were used for the frontend, Node.js for the backend, and MySQL as the database management system. Visual Studio Code was chosen as the primary development environment.

The system was tested for compliance with the specified functional requirements.

Keywords: time management, neurodiversity, ADHD, web application, event calendar, notes, energy resource, spoons, user interface, Tamagotchi, inclusivity, gamification.

ЗМІСТ

ВСТУП

Протягом останніх десяти років у суспільстві спостерігається зростання розуміння важливості ментального здоров’я, нейропсихологічного розмаїття та потреби адаптації цифрових технологій до індивідуальних особливостей користувачів. Особливу увагу почали приділяти людям із нейровідмінністю — зокрема тим, хто має розлад дефіциту уваги та гіперактивністю (РДУГ) [1], розлади аутистичного спектра [2], тривожні розлади [3] та інші когнітивні особливості. Все більше підлітків і дорослих починають сприймати свій нейропрофіль не як діагноз у традиційному сенсі, а як унікальний спосіб сприйняття й взаємодії з навколишнім світом. Багато з них із полегшенням усвідомлюють, що їхні труднощі з організацією часу, плануванням і збереженням енергії не свідчать про лінощі чи характер, а є наслідком нейровідмінності, що потребує відповідних умов і підтримуючих інструментів.

Одним із найпоширеніших викликів для нейровідмінних людей є складнощі з тайм-менеджментом. Труднощі з оцінкою витрат енергії на виконання завдань, часті «зависання» між діями створюють значні бар’єри в повсякденному житті, навчанні та професійній діяльності. І хоча на ринку існує велика кількість планувальників — від Google Calendar до мобільних застосунків для трекінгу задач — більшість із них орієнтовані на нейротипових користувачів.

Такі інструменти часто мають надмірну функціональність, можуть спричиняти сенсорне перенавантаження (наприклад, через яскраві кольори чи зайву візуальну інформацію), не підтримують планування за ресурсом людини («теорія ложок»), замість зменшення навантаження, лише його підсилюють, що приводить до повного вигорання.

Також у наявних нейротипових аналогах відсутня система гейміфікації, яка могла б допомогти із невеликими викидами дофаміну, які так важливі для нейровідмінних людей, оскільки їхній мозок не виробляє достатню кількість цієї речовини, так званого «гормона щастя».

Саме тому розробка вебсистеми для тайм-менеджменту нейровідмінних людей є актуальною та матиме перевагу над конкурентами на ринку. Запропонована система покликана не лише допомогти користувачам у плануванні часу, а й створити підтримувальне цифрове середовище, адаптоване до їхнього стилю сприйняття, мотивації та енергетичних обмежень. Вона інтегрує концепцію «теорії ложок», що дозволяє враховувати обмежений щоденний ресурс користувача, і попереджає про ризик перевантаження ще до того, як воно призведе до вигорання.

Окрему увагу приділено функції саморефлексії — користувач може щодня оцінювати свій стан та кількість ресурсу на день. Ці дані дозволяють системі адаптуватися до користувача в динаміці, пропонуючи релевантні поради або зміни у планів. Таким чином, система виступає не лише інструментом для організації часу, а й партнером у турботі про добробут користувача.

Завдяки комплексному підходу до проектування, який включає досвід нейровідмінних користувачів, сучасні UX-практики та принципи інклюзивного дизайну, розроблений продукт здатен заповнити важливу нішу на ринку цифрових планувальників. Його використання може істотно підвищити якість життя нейровідмінних людей, сприяючи їхній автономності, впевненості у власних силах та стабільності щоденного ритму.

Метою бакалаврської кваліфікаційної роботи є вдосконалення процесу тайм-менеджменту та щоденного планування шляхом розробки адаптивної вебсистеми, що забезпечує нейровідмінним користувачам можливість краще управляти своїм часом з урахуванням їхніх унікальних когнітивних, емоційних і ресурсних потреб. Для досягнення поставленої мети необхідно виконати такі завдання:

- аналіз існуючих аналогів та визначення їхніх обмежень щодо потреб нейровідмінних користувачів;

- проєктування архітектури вебсистеми з урахуванням адаптивного інтерфейсу та моделі «ресурсних ложок»;

- реалізація серверної частини з використанням Node.js для забезпечення гнучкої обробки подій і збереження даних;

- розробка модулю клієнтської частини, що дозволяють створювати, редагувати та переглядати події з різними типами повторюваності;

- розробка модулю клієнтської частини, що відповідає за створення заміток;

- розробка модулю клієнтської частини для взаємодії із інтерактивним тамагочі та системою «монет» для покупки аксесуарів для тваринки;

- створення бази даних для зберігання інформації про користувачів, події та ресурси, з використанням MySQL;

- тестування роботи системи та оцінка її зручність для користувачів з нейровідмінністю.

Об'єктом дослідження є процес розробки модулів вебсистеми для організації часу та тайм-менеджменту для людей із когнітивними особливостями.

Предметом дослідження виступають методи та засоби, що забезпечують ефективне адаптивне планування з урахуванням індивідуальних потреб користувачів.

Новизна роботи полягає у наступному: покращено тайм-менеджмент для нейровідмінних користувачів, шляхом розробки адаптивної вебсистеми, яка враховує принципи інклюзивного дизайну та когнітивні особливості користувачів, що забезпечує зручне, ресурсно-орієнтоване планування. Реалізація механізмів, таких як «теорія ложок» та система гейміфікації, суттєво покращує користувацький досвід у порівнянні з існуючими аналогів, орієнтованими переважно на нейротипових користувачів.

Практична цінність системи полягає в можливості застосування розробленої системи як засобу щоденного планування для нейровідмінних осіб, що сприяє більш кращому управлінню особистими ресурсами, зниженню перевантаження та підтримці емоційної рівноваги. Окрім того, система має потенціал для використання в освітніх установах, психологічній практиці та

інклюзивних середовищах, що загалом сприяє покращенню якості життя користувачів і їхній кращій адаптації до повсякденних потреб.

Апробація та публікація результатів бакалаврської кваліфікаційної роботи. Результати кваліфікаційної роботи доповідалися на Всеукраїнській науково-методичній конференції «Освіта, наука та виробництво: розвиток та перспективи», а також на Міжнародній міждисциплінарній науково-практичній конференції «Актуальні питання, проблеми та перспективи розвитку науки і освіти», і опубліковані в тезах доповіді цих конференцій [4,5].

1 АНАЛІЗ СУЧАСНОГО СТАНУ СИСТЕМ ДЛЯ ТАЙМ-МЕНЕДЖМЕНТУ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану технологій для тайм-менеджменту нейровідмінних людей

У сучасному цифровому середовищі існує велика кількість інструментів для планування часу та управління завданнями — від простих списків справ до складних систем із календарями, нагадуваннями та аналітикою. Ці інструменти надають широкий функціонал, що дозволяє структурувати завдання, контролювати дедлайни, ставити нагадування та працювати над проєктами в команді. Проте, незважаючи на багатофункціональність, більшість таких систем проєктуються з урахуванням потреб нейротипових користувачів, які мають стабільну здатність до концентрації, гнучко адаптуються до інтерфейсів та можуть добре взаємодіяти з абстрактними концепціями планування.

Для нейровідмінних людей, зокрема з РДУГ [1] (розладом дефіциту уваги і гіперактивністю), аутизмом [2] чи тривожними розладами [3], класичні інструменти тайм-менеджменту часто виявляються неефективними. Причинами цього є надмірна когнітивна складність інтерфейсу, відсутність візуальної гнучкості та обмежені можливості для індивідуалізації досвіду, а також відсутність врахування внутрішнього стану користувача — зокрема рівня енергії, емоційної стійкості та сенсорних навантажень.

На відповідь цим викликам почали з’являтися спеціалізовані системи, орієнтовані саме на нейровідмінних користувачів. Також поширеними є паперові системи на основі метафори «ложок», яка описує обмежений обсяг енергії, що доступний людині з нейровідмінністю протягом дня. Однак такі системи залишаються розрізненими, фрагментованими та, здебільшого, не інтегрованими з цифровими технологіями в достатньому обсязі.

Ще одним недоліком є відсутність підтримки для гнучкої повторюваності подій, відображення візуальної структури дня з акцентом на енерговитрати, а також зручних інструментів для планування з урахуванням когнітивних

обмежень. Багато застосунків створюються без врахування сенсорної чутливості користувачів (наприклад, агресивні кольори, дрібний текст, надлишок анімацій), що ускладнює їхнє використання.

Отже, попри наявність окремих спроб адаптації традиційних інструментів тайм-менеджменту до потреб нейровідмінних користувачів, на ринку все ще бракує комплексного цифрової системи, що дозволяє поєднати адаптивний інтерфейс, систему енергетичного планування, гнучку логіку повторення подій, процесу та зручну візуалізацію структури дня. Це створює передумови для подальшого дослідження та розробки спеціалізованого програмного забезпечення, що відповідає реальним потребам нейровідмінної аудиторії.

1.2 Порівняльний аналіз аналогів

На сучасному ринку програмного забезпечення існує чимало інструментів, які позиціонуються як засоби для планування, підвищення продуктивності та особистого тайм-менеджменту. Проте лише одиниці з них враховують потреби нейровідмінних користувачів. Для здійснення порівняльного аналізу було обрано чотири популярних застосунку: Habitica, Microsoft To Do, Todoist: Planner & Calendar та Trello. Вони аналізуються за набором критеріїв, які є критично важливими саме для користувачів з РДУГ, аутизмом або тривожними розладами, а також у контексті порівняння із власною програмною системою, назвемо його «ResourseHub».

Habitica [6] (рис. 1.1) — це застосунок, який поєднує функціонал списку завдань з ігровими механіками. Користувачі можуть створювати завдання, отримувати за їх виконання віртуальні нагороди та прокачувати персонажа. Такий підхід стимулює до дій, але має певні недоліки: велика кількість дрібних елементів, візуальна навантаженість, складність у структурованому плануванні подій на день або тиждень.

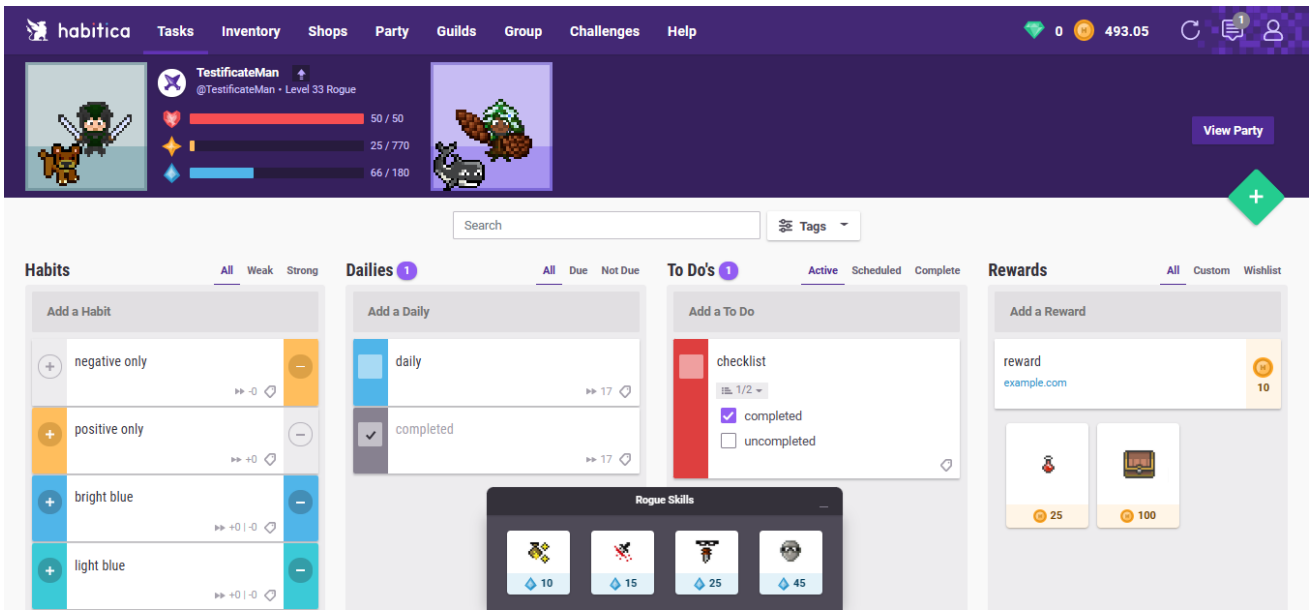

Рисунок 1.1 – Представлення інтерфейсу програми Habitica

Microsoft To Do [7] (рис. 1.2) — це зручний та інтуїтивно зрозумілий застосунок від Microsoft, який дає змогу формувати списки справ, встановлювати дедлайни, нагадування та сортувати завдання за категоріями. Водночас у застосунку відсутня підтримка моделі обліку ресурсів, таких як рівень енергії, немає візуального представлення розкладу дня, а його інтерфейс, попри простоту, більше підходить для нейротипових користувачів і не враховує специфічних когнітивних потреб нейровідмінної аудиторії.

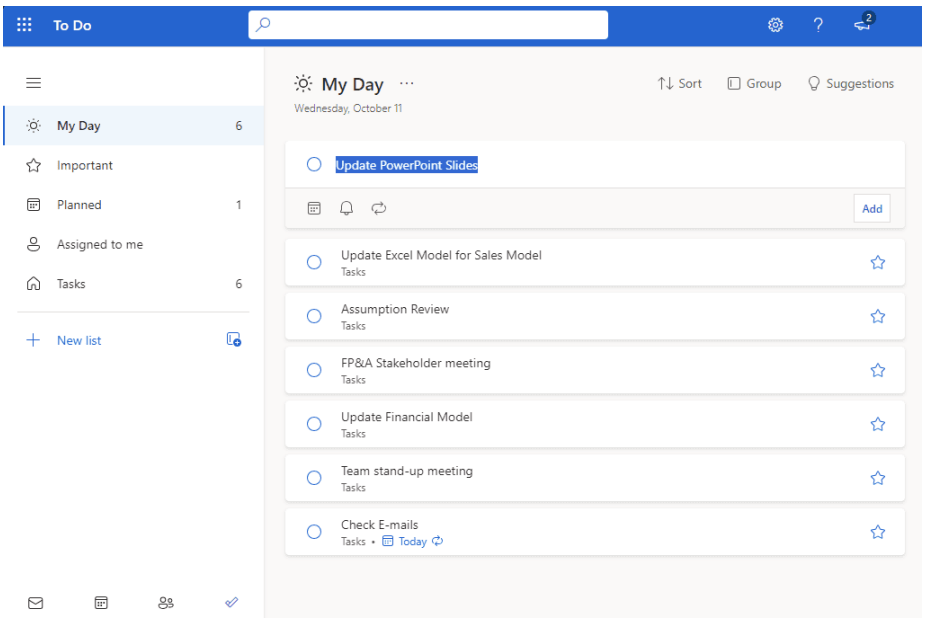

Рисунок 1.2 – Представлення інтерфейсу програми Microsoft To Do

Todoist: Planner & Calendar [8] (рис. 1.3) — один із найкращих інструментів для особистого тайм-менеджменту, що підтримує встановлення пріоритетів, фільтрацію завдань, дедлайни, повторювані події та інтеграцію з календарями. Незважаючи на багатий функціонал, його інтерфейс є досить складним і вимагає високого рівня зосередженості та навичок системного мислення для повноцінного використання. У застосунку відсутні механізми обліку енерговитрат користувача, а також немає інструментів для адаптації під сенсорні або когнітивні особливості.

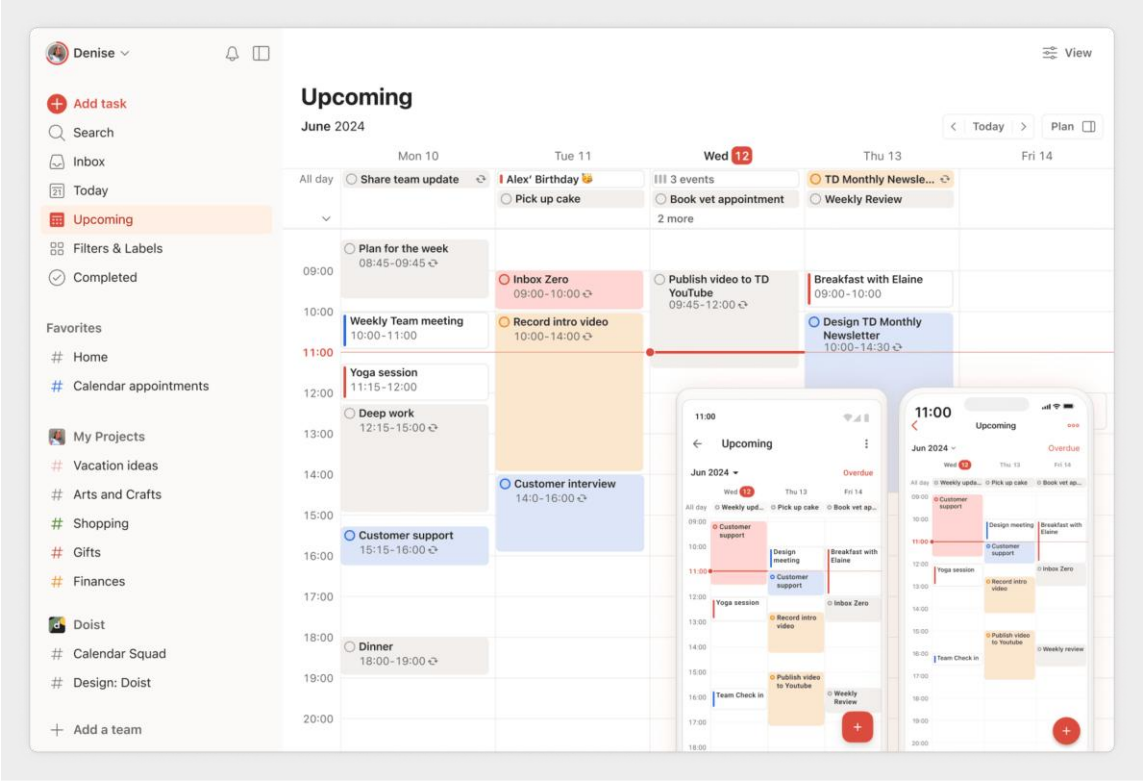

Рисунок 1.3 – Представлення інтерфейсу програми Todoist: Planner & Calendar

Trello [9] (рис. 1.4) — це платформа візуального керування завданнями, побудована на основі дошок, колонок і карток, яка забезпечує значну гнучкість і зручна для командної співпраці. Водночас її функціональна насиченість, велика кількість візуальних елементів і складна система автоматизацій можуть стати перепоною для нейровідмінних користувачів. Особливо складним є орієнтування у великій кількості вбудованих можливостей, що ускладнює щоденне використання без перевантаження.

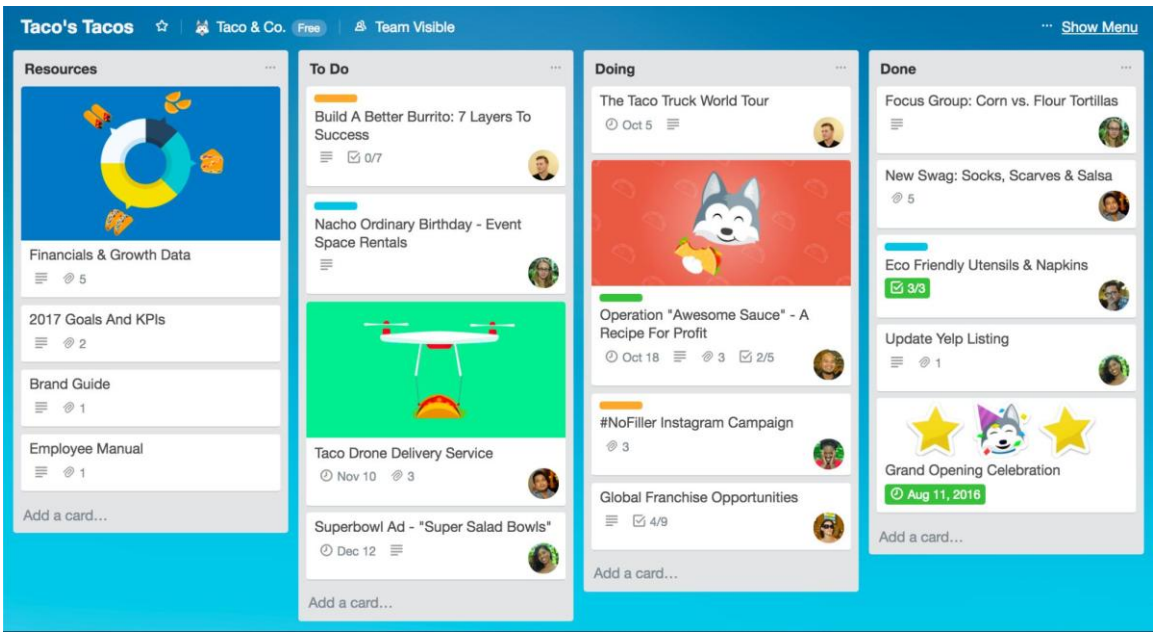

Рисунок 1.4 – Представлення інтерфейсу програми Trello

Перш ніж приступати до створення власного програмного продукту, доцільно провести аналіз наявних аналогів у сфері тайм-менеджменту. Такий аналіз дає змогу не лише окреслити переваги найпопулярніших застосунків, а й виявити їхні недоліки з точки зору зручності та доступності для нейровідмінних користувачів. Визначення обмежень існуючих систем допомагає сформувати обґрунтовані вимоги до майбутнього застосунку, уникнути повторення поширених помилок і зосередити увагу на функціоналі, що справді відповідає потребам цільової аудиторії та сприяє підвищенню якості користувацького досвіду.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 — Порівняльна характеристика аналогів

Критерії	Habitica	Microsoft To-Do	Todoist	Trello	ResourseHub
Система для організації власного енергового ресурсу	-	-	-	-	+

Продовження таблиці 1.1

Критерії	Habitica	Microsoft To-Do	Todoist	Trello	ResourseHub
Особистий кабінет користувача	+	-	+	+	+
Календар для подій прив’язаний до часу або повторюваності	-	-	+	-	+
Нотатки без прив’язки до дня	-	+	+	+	+
Фокус-таймер для повного зосередження користувача	-	-	-	-	+
Гейміфікація процесу планування	+	-	-	-	+
Загальна оцінка	2	1	3	2	6

Проведений аналіз порівняльної таблиці свідчить про очевидну перевагу програмної системи ResourceHub у контексті задоволення потреб нейровідмінних користувачів. На відміну від популярних аналогів, лише цей застосунок пропонує механізм обліку власного енергетичного ресурсу, що є критично важливим для людей із РДУГ, аутизмом чи тривожними станами. У той час як інші сервіси реалізують лише окремі функції — як-от базове планування, роботу з нотатками або гейміфікацію, — ResourceHub об’єднує їх у цілісну, логічно структуровану систему, орієнтовану на зменшення когнітивного навантаження та підвищення зручності користування. У результаті саме

ResourceHub отримує найвищу загальну оцінку серед представлених аналогів, що підтверджує його доцільність як спеціалізованого інструменту для нейровідмінної аудиторії.

Отже, результати порівняльного аналізу свідчать про те, що жоден із досліджених аналогових застосунків не охоплює повною мірою функціональні потреби нейровідмінних користувачів. Попри наявність окремих переваг, кожен із них або має надмірну складність, або не враховує специфіку когнітивного сприйняття та роботи мозку людей з нейровідмінністю. Натомість розроблене в рамках роботи застосунок, вирізняється впровадженням механізмів обліку енергетичного ресурсу користувача (зокрема, на основі «Теорії ложок»), адаптивним інтерфейсом та рядом інших функцій, що робить його більш придатним і зручним для повсякденного використання представниками нейровідмінної аудиторії.

1.3 Аналіз методів розробки

Створення програмного забезпечення для нейровідмінної аудиторії потребує уважного підходу до вибору технологій, побудови інтерфейсу, організації взаємодії та структурування даних. Важливо, щоб така система не лише виконувала необхідні функції, але й враховувала когнітивні та сенсорні особливості користувачів, що суттєво визначає характер її архітектури та методів реалізації.

Для реалізації вебзастосунку тайм-менеджменту, орієнтованого на нейровідмінних користувачів, було обрано мову JavaScript. Такий вибір зумовлений насамперед її універсальністю, широкою підтримкою браузерами та здатністю забезпечити високий рівень інтерактивності інтерфейсу без потреби в додатковому програмному забезпеченні. Це суттєво спрощує доступ до системи для кінцевого користувача, що є критично важливим у контексті зниження когнітивного навантаження.

JavaScript [10] має розвинену екосистему сучасних інструментів і бібліотек, серед яких особливо вирізняється React [11] — фреймворк, що

дозволяє реалізувати адаптивний, гнучкий і доступний інтерфейс. Завдяки компонентному підходу можна покращено структурувати взаємодію та адаптувати візуальні елементи до індивідуальних потреб користувачів: змінювати кольори, розміри, анімації та логіку відображення залежно від когнітивних або сенсорних особливостей.

Angular і Vue.js — це два популярні фреймворки для розробки вебінтерфейсів, кожен із яких має свої переваги та застосовується залежно від потреб системи. Angular [12], створений компанією Google, є комплексним фреймворком, що базується на TypeScript і надає всі необхідні інструменти для створення масштабованих застосунків. Його структура ґрунтується на суворому дотриманні архітектурних принципів, підтримці залежностей, модульності, двосторонньому зв’язуванню даних та розвиненій системі форм і маршрутизації. Такий підхід забезпечує надійність і стабільність у великих корпоративних проєктах, але водночас вимагає від розробника високого рівня підготовки та часу на вивчення численних концепцій.

На противагу Angular, Vue.js [13] — це легкий і гнучкий фреймворк з простим синтаксисом і низьким порогом входу, що дозволяє швидко створювати інтерактивні інтерфейси. Завдяки реактивній системі він автоматично оновлює дані в UI, а структура в одному файлі спрощує розробку. Vue особливо підходить для невеликих і середніх систем, де важлива швидкість і зручність.

В таблиці 1.2 представлено результати аналізу та порівняння цих фреймворків.

Таблиця 1.2 — Порівняння фреймворків для розробки вебінтерфейсів

Критерій	React	Angular	Vue.js
Гнучкість і кастомізація	+	+/-	+
Реактивність інтерфейсу	+	+/-	+
Адаптивність до UI-дизайну	+	+/-	+
Компонентний підхід	+	+	+
Підтримка доступності (a11y)	+	-	+/-
Контроль над структурою UI	+	-	+/-
Підходить для інклюзивного дизайну	+	+/-	+/-

На основі проведеного порівняння можна зробити висновок, що React є найкращим вибором для створення інклюзивного вебзастосунку тайм-менеджменту, орієнтованого на нейровідмінних користувачів. Його висока гнучкість, підтримка численних бібліотек для доступності, можливість точного контролю інтерфейсу та потужна спільнота розробників створюють сприятливі умови для реалізації адаптивного та чутливого до потреб користувача дизайну. На відміну від Angular, який вимагає складної архітектури та великої кількості ресурсу на впровадження, React дозволяє сфокусуватися саме на досвіді користувача. Vue.js, хоч і простіший у використанні, має менш розвинену екосистему для доступності, що обмежує його потенціал у спеціалізованих інклюзивних проєктах.

Проаналізуємо та оберемо оптимальний інструмент для реалізації серверу: Node.js, PHP, .Net.

Node.js [14] — це середовище виконання JavaScript на сервері, побудоване на рушії V8 від Google Chrome. Його головна особливість — неблокуюча, подієво-орієнтована архітектура, що дозволяє швидко обробляти велику кількість одночасних запитів без втрати продуктивності. Завдяки цій властивості Node.js особливо добре підходить для створення застосунків реального часу, таких як чати, стрімінг або інтерактивні панелі керування.

PHP (Hypertext Preprocessor) [15] — мова програмування, створена спеціально для веброзробки. Вона працює за синхронною моделлю та виконується на сервері кожного разу, коли користувач надсилає запит. PHP широко використовується для створення динамічних вебсторінок і підтримується багатьма CMS, такими як WordPress, Joomla чи Drupal. Простота у використанні та широка підтримка на хостингах зробили PHP однією з найрозповсюдженіших серверних технологій. Однак у контексті сучасних високонавантажених застосунків PHP поступається Node.js і .NET за продуктивністю та гнучкістю архітектури.

.NET [16] — це серверна платформа від Microsoft, яка дозволяє створювати масштабовані, надійні та продуктивні вебзастосунки за допомогою мов C# або

VB.NET. Однією з ключових технологій є ASP.NET Core — кросплатформений фреймворк для створення веб API та серверної логіки. .NET вирізняється високою продуктивністю, строгим підходом до типізації, вбудованими засобами безпеки та підтримкою багатопотокової обробки даних. Його часто обирають для корпоративних систем, де важлива стабільність, масштабованість та інтеграція з іншими сервісами Microsoft.

Результати аналізу, за допомогою критеріїв, важливих для розробки інклюзивного вебзастосунку для тайм-менеджменту нейровідмінних людей, подано у таблиці 1.3.

Таблиця 1.3 — Порівняння інструментів для реалізації серверу

Критерій	Node.js	PHP	.NET
Підтримка реального часу	+	+/-	+/-
Продуктивність при високій кількості запитів	+	+/-	+
Гнучкість архітектури	+	-	+
Інтеграція з фронтендом	+	+/-	+/-
Масштабованість	+	+/-	+
Гнучкість під потреби нейровідмінних	+	-	+/-

В результаті аналізу, для реалізації серверної логіки було обрано Node.js. У зв’язці з Express.js вона дозволяє гнучко будувати REST API для створення, редагування та перегляду подій, ведення обліку енергетичних ресурсів користувача та керування повторюваністю завдань. Така архітектура забезпечує простоту масштабування та можливість подальшого розширення функціоналу.

Вибір бази даних є критично важливим етапом розробки вебзастосунку для тайм-менеджменту нейровідмінних користувачів, оскільки саме вона визначає спосіб зберігання, доступу, обробки та структурування інформації, яка лежить в основі взаємодії користувача із системою. У такому застосунку необхідно зберігати події, повторювані завдання, щоденні ресурси («ложки»), особисті налаштування, нотатки, дані про стан користувача, історію змін тощо — усе це потребує гнучкої та надійної моделі даних. Розглянемо такі бази даних як: MySQL, PostgreSQL, MangoDB.

MySQL [17] — це одна з найпопулярніших реляційних систем управління базами даних, яка зберігає інформацію у вигляді таблиць і використовує мову SQL для створення, оновлення та вибірки даних. Вона вирізняється простотою використання, стабільністю та широкою підтримкою серед хостинг-провайдерів. Завдяки своїй легкості та добрій продуктивності MySQL часто використовується для створення вебзастосунків.

PostgreSQL [18] — це потужна об'єктно-реляційна система управління базами даних з відкритим кодом, яка підтримує не лише класичні SQL-операції, а й складні типи даних, розширення, тригери та процедури. Вона відома своєю надійністю, високою відповідністю стандартам та гнучкістю. PostgreSQL дозволяє зберігати неструктуровані дані, працює з JSON, має розвинену систему прав доступу й транзакцій, що робить її ідеальним вибором для застосунків з ускладненою логікою й високими вимогами до цілісності даних.

MongoDB [19] — це нереляційна (NoSQL) база даних, яка зберігає інформацію у вигляді гнучких документів у форматі BSON (розширений JSON). На відміну від реляційних СУБД, MongoDB не вимагає суворої схеми даних, що дозволяє швидко розробляти та змінювати структуру збережених об’єктів. Вона особливо підходить для систем із динамічно змінюваною структурою, для зберігання подій, профілів користувачів, журналів, а також тоді, коли важлива висока швидкість читання і запису.

Результати аналізу баз даних для системи подано у таблиці 1.4.

Таблиця 1.4 — Порівняння баз даних

Критерій	MySQL	PostgreSQL	MongoDB
Простота впровадження	+	+/-	+
Продуктивність у типових CRUD-операціях	+	+	+/-
Структура даних для повторюваних подій	+	+/-	+/-
Облік ресурсів користувача (ложки)	+	+/-	+/-
Масштабованість	+	+	+
Підтримка доступності та UX-функцій	+	+/-	+/-
Стабільність та надійність	+	+	+/-

З урахуванням потреб розробки інклюзивного вебзастосунку для тайм-менеджменту нейровідмінних користувачів, MySQL є найбільш збалансованим вибором. Вона забезпечує стабільну роботу, чітку структуру для зберігання повторюваних подій, енергетичних ресурсів користувача та нотаток, а також легко інтегрується з бекендом на Node.js. Простота в налаштуванні, висока продуктивність при базових операціях та підтримка транзакцій дають змогу уникати помилок, які можуть дезорієнтувати або перевантажити користувача. Завдяки цьому MySQL дозволяє створити надійну і чутливу до когнітивних потреб основу для роботи всього застосунку.

Важливою складовою системи є UX-дизайн, адаптований до нейропсихологічних особливостей користувачів [20]. При створенні інтерфейсу застосовується мінімалістичний підхід, що спрямований на зниження когнітивного навантаження: використовуються прості кольори, достатній контраст, збільшені шрифти та відсутність зайвих анімацій.

Гнучкість у налаштуванні повторюваних подій є одним із ключових елементів системи. Реалізація здійснена за допомогою алгоритмів, що дозволяють задавати повторення за різними шаблонами — «щодня», «щотижня», «у вибрані дні» або «щомісяця». Такий підхід забезпечує адаптивність до змінного та нестабільного розпорядку, який характерний для нейровідмінних користувачів.

Крім того, в систему інтегровано модель обліку енергетичного ресурсу, засновану на концепції Spoon Theory. Вона ґрунтується на фіксації кількості доступної енергії у вигляді умовних «ложок» та сповіщає користувача у разі перевищення встановленого ліміту. Механізм працює шляхом підрахунку загальної витрати ресурсу протягом дня і порівняння її з заданим обмеженням, що дозволяє своєчасно виявляти ризик перевантаження та регулювати навантаження відповідно до індивідуальних можливостей.

Теорія «ложок» (Spoon Theory) [21] — це метафоричний підхід до розуміння обмеженого енергетичного ресурсу, яким щоденно володіє людина для виконання повсякденних дій. Її авторкою є Крістін Мізерандо, активістка з

вовчаком, яка використала цю концепцію, щоб пояснити своє оточення, як саме хронічна хвороба впливає на її здатність функціонувати. Згодом теорія набула популярності серед людей з різними нейровідмінностями — зокрема, РДУГ та аутизмом.

В основі метафори лежить уявлення про те, що кожен день людина починає з обмеженого запасу умовних одиниць енергії — «ложок». Наприклад, прокинутися, прийняти душ або поїсти — це вже витрати ложок, а складніші дії на кшталт навчання чи спілкування можуть вимагати ще більше. На відміну від нейротипових людей, нейровідмінні особи часто мають суттєво менший резерв енергії, тому потребують більш зваженого планування, щоб уникнути виснаження та вигорання.

У контексті тайм-менеджменту ця модель допомагає користувачам краще розуміти свої можливості, розставляти пріоритети та адаптувати розпорядок дня не лише за часом, а й за рівнем витрат енергії. Саме тому концепція Spoon Theory є основою для реалізації функціоналу в системі: кожна подія має визначену «вартість» у ложках, а система відстежує витрати і попереджає про наближення або перевищення встановленого ліміту.

Отже, поєднання React для клієнтського інтерфейсу, Node.js для реалізації серверної логіки, MySQL як системи керування базою даних, а також адаптивного UX-дизайну створює стійку й чутливу до потреб користувача технічну основу вебсистеми. Обрані технології дозволяють досягти оптимального балансу між функціональністю, гнучкістю у налаштуванні, доступністю інтерфейсу, що є ключовими чинниками для розробки системи тайм-менеджменту, орієнтованої на нейровідмінних людей.

1.4 Постановка задач для розробки інклюзивного вебзастосунку для тайм-менеджменту нейровідмінних людей

З огляду на проведений аналіз стану технологій у сфері тайм-менеджменту для нейровідмінних користувачів, а також вивчення методів створення адаптивного програмного забезпечення, сформульовано перелік основних задач,

які необхідно реалізувати в межах цієї системи. Кожна із задач орієнтована на покращення системи тайм-менеджменту та викорінення недоліків, що були виявлені в процесі аналізу аналогів і потреб цільової аудиторії.

До основних задач роботи належать:

- спроєктувати архітектуру вебсистеми з урахуванням адаптивного інтерфейсу та моделі «ресурсних ложок»;

- реалізувати серверну частину з використанням Node.js для забезпечення гнучкої обробки подій і збереження даних;

- розробити модулі клієнтської частини, що дозволяють створювати, редагувати та переглядати події з різними типами повторюваності у календарі;

- розробити модулі клієнтської частини, що відповідають за створення заміток;

- розробити модулі клієнтської частини, що відповідають за інтерактивного тамагочі;

- створення бази даних для зберігання інформації про користувачів, події та ресурси, з використанням MySQL;

- провести тестування роботи системи та оцінити її зручність для користувачів з нейровідмінністю.

Отже, у цьому підрозділі було поставлено задачі бакалаврської кваліфікаційної роботи.

1.5 Висновки

У першому розділі було проведено комплексний аналіз сучасних програмних аналогів у сфері тайм-менеджменту, з акцентом на їхню придатність для використання нейровідмінними людьми. Розглянуто такі популярні інструменти, як Habitica, Microsoft To Do, Todoist та Trello. Проаналізовано їхні функціональні можливості, а також недоліки у контексті потреб користувачів із РДУГ, аутизмом, тривожними розладами та іншими формами нейровідмінності. Особливу увагу приділено таким аспектам, як система для контролю енергії, наявність особистого кабінету користувача, календаря для подій прив’язаного до

часу або повторюваності, наявність системи нотаток без прив’язки до часу, наявність фокус-таймера для повного зосередження користувача на задачі та гейміфікація процесу планування.

На основі виявлених недоліків було обґрунтовано необхідність створення спеціалізованої програмної системи, орієнтованого на нейровідмінних користувачів. У ході аналізу методів розробки визначено найбільш ефективні технології для реалізації проєкту — зокрема, використання React для клієнтської частини, Node.js для побудови серверної логіки, та бази даних MySQL для зберігання інформації. Враховано також підходи до UX-дизайну, адаптивного до сенсорних і когнітивних особливостей користувачів.

Зважаючи на результати аналізу, було сформульовано перелік задач, які мають бути реалізовані в межах курсового проєкту.

Отже, було доведено актуальність і доцільність розробки власного спеціалізованого програмної системи, здатного задовольнити специфічні потреби нейровідмінної аудиторії в сфері щоденного планування та тайм-менеджменту.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Розробка моделі системи

Для розуміння логіки роботи вебсистеми та взаємозв’язків між її ключовими компонентами було побудовано модель, що відображає основні процеси взаємодії користувача з системою. У цьому підрозділі модель представлено у вигляді UML-діаграми діяльності (рис. 2.1), яка наочно демонструє послідовність дій, переходів між станами, а також точки вибору дій під час взаємодії з інтерфейсом.

На основі функціональності системи користувач проходить кілька ключових етапів у роботі з застосунком. Після авторизації або входу в систему користувач потрапляє на головний екран, де він може оцінити свій енергетичний стан за шкалою «ложок». Наступною можливою дією є перехід до одного з функціональних розділів: «календар», «нотатник», «тамагочі» та «селфхелп».

У взаємодії з календарем користувач може переглядати поточний день або обирати будь-яку іншу дату. В межах вибраного дня доступні функції перегляду списку запланованих подій, додавання нової події через спеціальну форму, а також редагування чи видалення вже існуючих записів.

Діаграма діяльності також враховує введення даних у форму події: назви, часу початку й завершення, кількості ложок, а також вибір іконки та варіанту повторюваності. Після натискання кнопки підтвердження подія зберігається в базу даних, а загальний список подій оновлюється. У разі перевищення денного ліміту ложок система повідомляє користувача попереджувальним повідомленням.

Важливими елементами діаграми є умовні розгалуження: наприклад, якщо подія перевищує доступний енергетичний ресурс — користувач має вибір або зменшити значення ложок, або все ж таки зберегти подію попри попередження. Це демонструє гнучкість системи й орієнтацію на свідоме планування, а не жорстке обмеження.

Завдяки моделі у вигляді UML-діаграми діяльності вдалось структуровано відобразити основні сценарії використання, типові дії користувача, можливі варіанти поведінки та наслідки їх виконання. Такий підхід полегшує розробку та подальше тестування системи, дозволяє виявити вузькі місця в логіці до початку реалізації програмної частини.

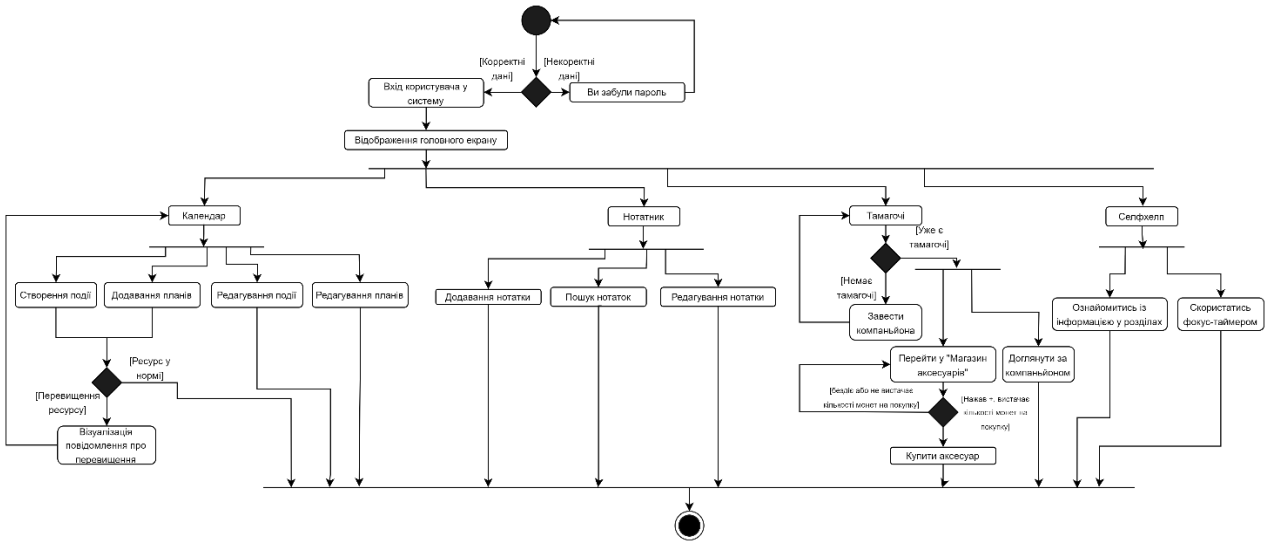

Рисунок 2.1 – Діаграма діяльності програмних засобів

Отже, створення моделі системи у вигляді діаграми діяльності дозволило глибше опрацювати сценарії взаємодії користувача із системою та забезпечити цілісність та узгодженість усіх функціональних компонентів вебзастосунку. Модель стала основою для реалізації логіки інтерфейсу та бекенд-архітектури. Розробка діаграми діяльності була виконана у програмному вебзастосунку Draw.io.

2.2 Розробка алгоритмів роботи системи

Для реалізації функціональності вебсистеми тайм-менеджменту нейровідмінних користувачів було спроєктовано загальний алгоритм, який відповідає за ключові процеси взаємодії.

Алгоритм роботи вебсистеми тайм-менеджменту для нейровідмінних користувачів, зображений на блок-схемі на рисунку 2.2, відображає основну логіку взаємодії користувача із застосунком від моменту входу до завершення сеансу. Після запуску користувач проходить процес автентифікації або реєстрації, після чого система завантажує головний екран із вітальним повідомленням і основними індикаторами. Далі перевіряється, чи є вже заплановані події або пункти плану на поточний день. Якщо вони є, користувач бачить список завдань, інакше — порожній інтерфейс із можливістю швидкого додавання інформації.

Після цього користувач встановлює кількість доступних ложок — тобто свій енергетичний ресурс на день.

Потім обирає потрібний розділ: календар, нотатник, тамагочі або селфхелп. У календарі реалізовано вибір формату перегляду (день, тиждень або місяць), після чого відбувається вибір дати та завантаження подій, запланованих на цей день.

Якщо користувач хоче додати нову подію чи пункт плану, відкривається відповідна форма введення. У неї можна додати час початку та час закінчення події, її назву або короткий опис, відмітити скільки «ложок» віднімає подія у користувача та призначити їй іконку, для швидшого розпізнавання. А також призначити повторюваність події.

Після визначення кількості ресурсу на цю подію система перевіряє, чи задана кількість ложок не перевищує доступний ліміт. У разі перевищення користувач отримує попередження, але має можливість переформулювати подію або вийти із цього меню. Якщо подія проходить перевірку по «ресурсності», кнопка «Зберегти» стає активною. Користувач її натискає і подія записується до бази даних та після цього відображається в календарі.

У випадку вибору нотатника користувач переглядає раніше створені нотатки або бачить повідомлення про їх відсутність. За бажанням можна створити нову нотатку, після чого вона також зберігається в базі даних і виводиться на екран. Для додавання нової нотатки виводиться окреме модальне

вікно, де можна вказати назву нотатки та її опис, обрати колір і також призначити мітки для організування порядку у всій системі.

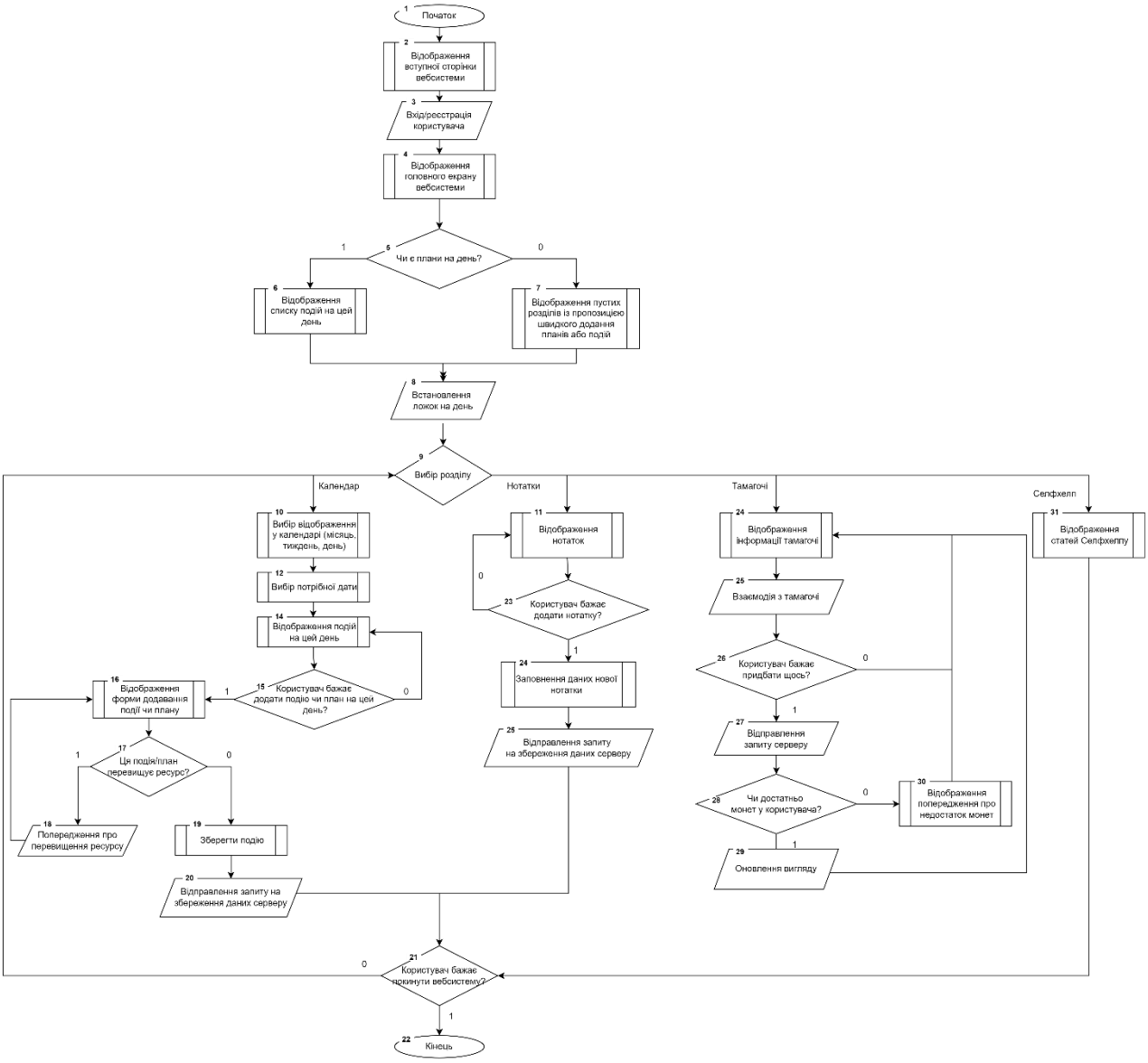

Рисунок 2.2 – Алгоритм роботи програмного застосунку

У розділі «Тамагочі» користувач може взаємодіяти із інтерактивним улюбленцем, підвищуючи його рівень щастя за допомогою різних дій. Це розвиває відповідальність користувача та додає стимулу для користування планувальним сервісом. Також є можливість купити аксесуари для тамагочі у спеціальному підрозділі «Магазин аксесуарів» за допомогою монет, які заробляються завдяки взаємодії із вебзастосунком.

У розділі «Селфхелп» користувач може ознайомитись із поняттям РДУГ, різними методами планування, використати влаштований таймер для фокус-сесії та ознайомитись з можливими видами втоми та як відпочивати при них, що важливо для нейровідмінних.

У користувача завжди є можливість покинути сервіс та завершити роботу з ним або повернутись на головний екран для подальшої роботи із вебзастосунком.

Отже, розроблені алгоритми забезпечують надійне та логічне функціонування системи: від обробки подій і ресурсів до повторюваності та зворотного зв’язку з користувачем. Вони є фундаментом адаптивного планування, що базується на енергетичному стані та індивідуальному ритмі життя нейровідмінних людей.

2.3 Розробка інтерфейсу програмного застосунку

Інтерфейс користувача є одним із ключових компонентів вебсистеми для тайм-менеджменту, особливо коли йдеться про аудиторію з нейропсихологічними особливостями. Від того, наскільки інтерфейс є доступним, інтуїтивним і ненавантажувальним, залежить ефективність взаємодії користувача з системою, його готовність планувати день та повернутись до використання застосунку знову. Саме тому розробка інтерфейсу у цій роботі базується на принципах адаптивного UX-дизайну [22], мінімалізму та врахування когнітивного навантаження.

Графічна частина реалізована з використанням бібліотеки React, що дозволяє будувати інтерфейс із незалежних компонентів. Такий підхід забезпечує гнучкість, можливість повторного використання коду та полегшує масштабування інтерфейсу в майбутньому.

У розробці інтерфейсу вебсистеми було використано спеціально підібрану палітру кольорів. Палітра містить нейтральні та базові кольори інтерфейсу, які використовуються для фону, тексту, заголовків та панелей. На рис 2.3 зображена палітра використаних кольорів для розробки інтерфейсу застосунку.

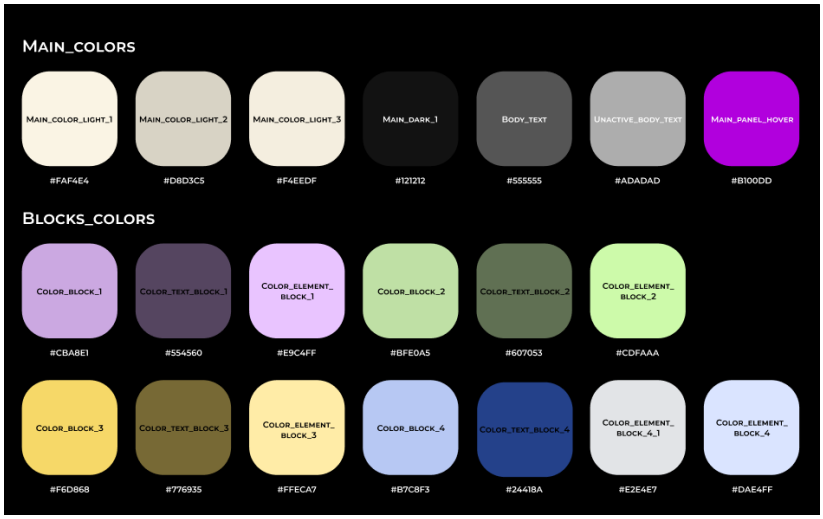

Рисунок 2.3 – Палітра, використана для інтерфейсу застосунку

Світлі відтінки (#FAF4E4, #D8D3C5, #F4EEDF) формують фон інтерфейсу, створюючи приємну, ненав'язливу візуальну атмосферу, яка не перевантажує зір. Темніші тони (#121212, #555555) використовуються для тексту та акцентів, забезпечуючи достатній контраст для читабельності.

Колір #B100DD (Main_panel_hover) обрано як яскравий акцент, що сигналізує про активну або наведено курсорну взаємодію. Його помітність дозволяє користувачеві легко орієнтуватися в інтерфейсі, не загублюючи фокус.

Кольорова палітра блоків призначена для візуального розділення типів контенту (наприклад, подій, сповіщень, категорій задач). Кожен блок має свою палітру: базовий колір, текстовий колір і колір елементів, а саме:

- Block 1 (#CBA8E1, #554560, #E9C4FF) використовує м’який фіолетовий — колір, який асоціюється з творчістю, спокоєм і безпекою. Він особливо корисний для маркування нетермінових або надихаючих завдань;

- Block 2 (#BFE0A5, #607053, #CDFAAA) має природний зелений відтінок, що часто асоціюється зі стабільністю, балансом та відпочинком — корисно для завдань, пов’язаних із самодопомогою або здоров’ям;

- Block 3 (#F6D868, #776935, #FFECA7) — теплі жовті тони, що активізують увагу, тому можуть використовуватись для позначення термінових або нагадувальних подій;

- Block 4 (#B7CBF3, #24418A, #E2E4E7, #DAE4FF) — холодні сині відтінки, які навіюють спокій і чіткість, що зручно для відображення інформаційних або структурних подій.

Причини вибору саме такої кольорової палітри зумовлені потребами цільової аудиторії — нейровідмінних користувачів [23], для яких надмірна візуальна насиченість може стати джерелом сенсорного перевантаження. У палітрі переважають приглушені, м’які кольори, які не дратують зір і не викликають втоми під час тривалої взаємодії з інтерфейсом. Поєднання світлих фонових відтінків з темними шрифтами забезпечує добру читабельність, а чітко виділені акцентні кольори дозволяють легко орієнтуватися у структурі подій. Різні кольорові блоки створюють асоціації з певними типами завдань, що полегшує сприйняття без додаткових підписів. Такий підхід дозволяє інтерфейсу залишатися одночасно функціональним, інклюзивним і приємним у використанні.

Інтерфейсна логіка побудована таким чином, щоб усі основні дії могли виконуватись за 2-3 кліки без потреби довгих переходів між сторінками. Це дає змогу зменшити когнітивне навантаження і зробити використання системи більш комфортним для людей, які швидко втомлюються від багаторівневої навігації.

Усі компоненти інтерфейсу розміщено з урахуванням принципів доступності: налаштування масштабування, можливість навігації з клавіатури та сумісність із програмами екранного озвучення. Отже, вебзастосунок є не лише зручним, але й інклюзивним.

Головний екран застосунку (рис. 2.4) спроєктовано з урахуванням доступності, зручності та структурованості для нейровідмінних користувачів. Ліворуч розташоване навігаційне меню з основними розділами (Головна, Календар, Нотатник, Тамагочі, Селфхелп), кожен з яких має іконку, активний розділ виділяється фіолетовим кольором. У центрі — вітальний блок, коловий індикатор тижневого прогресу та повзунок для встановлення кількості «ложок» — одиниць енергії. Праворуч — інтерактивний календар і список подій на день із чекбоксами. Унизу розміщено панель швидких дій для додавання нотаток,

подій або пунктів у план. Така структура забезпечує зручний доступ до ключових функцій і дозволяє оцінити стан справ та ресурсів.

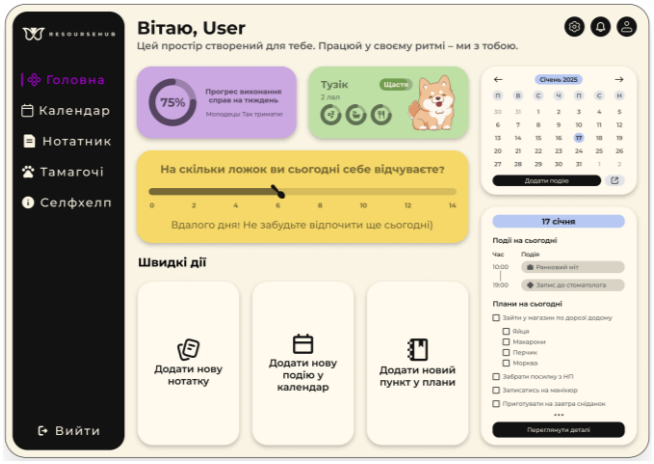

Рисунок 2.4 – Головний екран ResourseHub

Інтерфейс календаря (рис. 2.5) реалізовано у вигляді зручного місячного перегляду з можливістю перемикання між режимами «День», «Тиждень» і «Місяць». У центрі розташовано сітку з подіями по днях, праворуч — структуровані плани на день, тиждень і місяць із чекбоксами. Нижче відображено список подій на вибраний день. Кнопки додавання й редагування подій розташовані знизу для швидкого доступу.

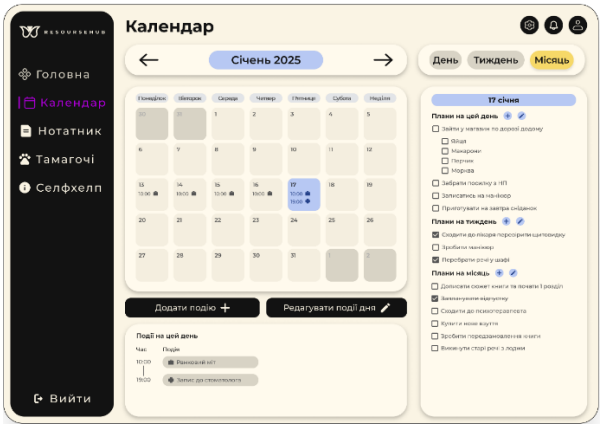

Рисунок 2.5 – Календар ResourseHub

Розділ «Нотатник» (рис. 2.6) призначений для збереження та організації коротких текстових записів. Інтерфейс містить кольорові картки нотаток із

назвами, мітками, описом і часом створення. Для зручності реалізовано функції фільтрації за мітками, пошуку та швидкого редагування або видалення нотаток. Завдяки кольоровому кодуванню та компактному розміщенню, розділ є зручним і візуально комфортним для перегляду великої кількості записів.

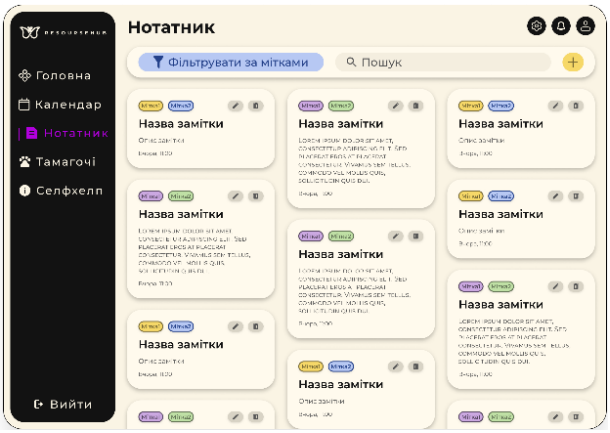

Рисунок 2.6 – Нотатник ResourseHub

Форма додавання подій у календар (рис. 2.7) дозволяє користувачу вказати час початку і завершення події, її назву або опис, кількість ресурсу (ложок), а також вибрати іконку для візуального позначення. Реалізовано зручну систему вибору повторюваності: одноразово, щодня, щотижня або із власним значенням. Інтерфейс форми мінімалістичний і зрозумілий, що дозволяє швидко створити подію без зайвого навантаження.

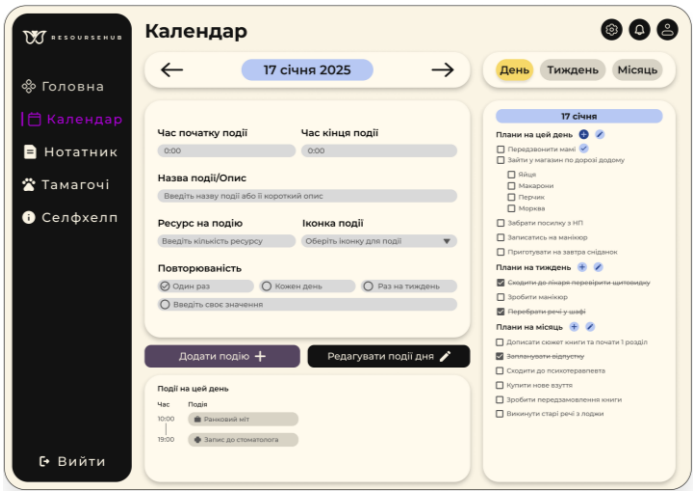

Рисунок 2.7 – Форма додавання полів у календар

Розділ «Тамагочі» (рис. 2.8) вебзастосунку дозволяє користувачеві взаємодіяти з віртуальним улюбленцем. У центрі екрана розміщено зображення тваринки на фоні кімнати з елементами інтер’єру. У верхній частині відображається кількість наявних монет, ім’я тамагочі та кнопка переходу до магазину аксесуарів. Знизу розташовано три дії для догляду: «Спатки», «Помити» та «Погодувати», кожна з піктограмою та кольоровим оформленням. Інтерфейс має теплу гаму, великі кнопки й інтуїтивну структуру, що робить його зручним і доступним для нейровідмінних користувачів.

Рисунок 2.8 – Інтерактивний тамагочі

Підрозділ під назвою «Магазин аксесуарів» (рис. 2.9) розділу «Тамагочі» Цей розділ дозволяє користувачеві придбати віртуальні предмети за внутрішню валюту — монети, які відображаються у верхній частині інтерфейсу. Основна частина екрана складається з візуальної сітки товарів. Кожен елемент представлено у вигляді окремого блоку з ілюстрацією аксесуара або тваринки, кнопкою з іконкою «+» для вибору товару та вказаною вартістю в монетах. У нижньому рядку відображено три варіанти віртуальних тваринок, серед яких обраний песик, а також інші можливі улюбленці – котик та папуга. У нижній частині екрана розміщено велику жовту кнопку «Купити обраний товар», яка активується після вибору предмета. Завдяки візуальній формі та гейміфікації, магазин підвищує інтерес до догляду за тамагочі та виконує мотиваційну функцію, важливу для підтримки нейровідмінних користувачів.

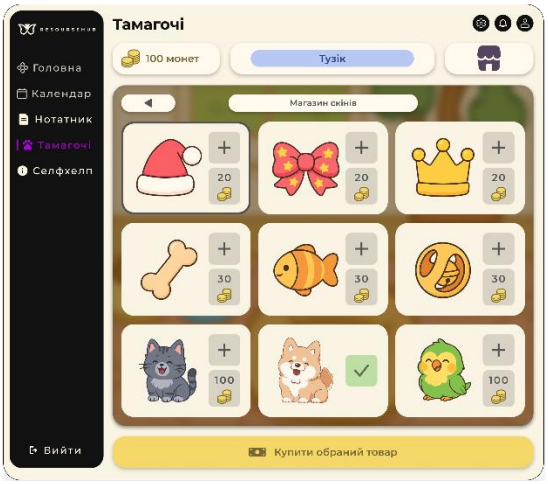

Рисунок 2.9 – Магазин аксесуарів

Отже, розроблений інтерфейс вебсистеми враховує як загальні вимоги до сучасного UX/UI-дизайну, так і специфічні потреби нейровідмінних користувачів. Завдяки використанню React, адаптивного компонування, спрощеної логіки взаємодії та візуальної гнучкості, інтерфейс забезпечує комфортне й ефективне використання функціоналу системи з мінімальним навантаженням на користувача.

2.4 Висновки

У другому розділі детально розглянуто процес розробки користувацького інтерфейсу з урахуванням потреб нейровідмінних користувачів. Описано основні алгоритми функціонування системи, зокрема механізми роботи з подіями та ресурсами. Також побудовано модель поведінки застосунку у вигляді UML-діаграми діяльності, що ілюструє ключові сценарії взаємодії користувача з функціоналом системи. Крім цього, побудовано загальний алгоритм роботи системи.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ВЕБЗАСТОСУНКУ ДЛЯ ТАЙМ-МЕНЕДЖМЕНТУ НЕЙРОВІДМІННИХ ЛЮДЕЙ

3.1 Обґрунтування вибору засобів та середовища для реалізації програмного забезпечення

У процесі розробки вебсистеми для тайм-менеджменту нейровідмінних користувачів постала потреба у виборі таких інструментів і технологій, які б дозволили забезпечити гнучку, масштабовану та доступну архітектуру програмного забезпечення. При цьому особливу увагу було приділено швидкодії, зручності інтеграції між клієнтською та серверною частинами, а також можливостям візуального оформлення інтерфейсу з урахуванням особливостей сприйняття користувачами.

На етапі аналізу було розглянуто кілька варіантів для реалізації клієнтської частини: використання фреймворків Angular, Vue.js або React. Серед них React було обрано як найбільш гнучкий, легкий у вивченні та широко підтримуваний інструмент з великою спільнотою розробників. Його компонентна структура дозволяє зручніше організувати логіку інтерфейсу, спрощує повторне використання елементів і добре підходить для створення адаптивних та доступних інтерфейсів — що є критично важливим у контексті роботи з нейровідмінними користувачами.

React [11] — це бібліотека JavaScript з відкритим вихідним кодом, призначена для створення користувацьких інтерфейсів, зокрема односторінкових вебзастосунків. React дозволяє будувати інтерфейси у вигляді окремих компонентів, що спрощує повторне використання коду, підтримку та масштабування проєктів. Основною перевагою React є реактивність: зміни в даних миттєво відображаються в інтерфейсі без потреби в перезавантаженні сторінки.

Для реалізації серверної частини, у процесі аналізу, було розглянуто можливість використання платформ PHP, .Net, та Node.js. Обрано було Node.js у поєднанні з Express, оскільки ця технологія дозволяє використовувати JavaScript

як на клієнтській, так і на серверній стороні, що спрощує обмін даними між компонентами системи. Крім того, Node.js забезпечує високу швидкість обробки запитів, що є важливим для інтерактивної роботи користувача в реальному часі, наприклад, при оновленні календаря чи енергетичного індикатора без перезавантаження сторінки.

Node.js [14] — це серверна платформа, яка дозволяє запускати JavaScript-процеси поза межами браузера. Вона побудована на рушії V8 від Google і забезпечує асинхронну, подієво-орієнтовану модель обробки, що робить її особливо доброю для вебзастосунків, які працюють у режимі реального часу. Node.js часто використовується для створення бекендів, REST API та мікросервісів.

У якості системи керування базами даних було розглянуто три основні варіанти: реляційна MySQL, об'єктно-реляційна PostgreSQL та документно-орієнтована MongoDB. З огляду на структурованість даних у системі (таблиці подій, користувачів, ресурсів, повторюваності тощо), вибір було зроблено на користь MySQL, яка краще підходить для зберігання строго визначених записів з чіткими зв’язками. Для зручної роботи з базою даних під час розробки використовувалось середовище MySQL Workbench, яке забезпечує візуальне моделювання структури, виконання запитів та контроль над схемою даних.

MySQL [17] — це система керування реляційними базами даних, яка дозволяє створювати, зберігати, змінювати та обробляти структуровані дані. Вона базується на мові SQL (Structured Query Language) і широко використовується у веброзробці для зберігання даних користувачів, подій, налаштувань тощо.

І на останок, розглянемо та обґрунтуємо вибір середовища програмування для реалізації проекту. Розглянемо такі середовища програмування як: Visual Studio Code, WebStorm, Atom.

Visual Studio Code [24] — це безкоштовне, кросплатформене середовище розробки від компанії Microsoft, орієнтоване на роботу з JavaScript, TypeScript, HTML, CSS та іншими мовами програмування. Воно підтримує розширення, які

дозволяють налаштовувати середовище під будь-який технологічний стек, зокрема React, Node.js і MySQL. Visual Studio Code має вбудований термінал, інтеграцію з Git, потужну систему автодоповнення та дебагінгу, що робить його зручним для повноцінної веброзробки, включаючи системи з акцентом на адаптивність і UX-дизайн.

WebStorm [25] — це комерційне інтегроване середовище розробки від компанії JetBrains, спеціально створене для JavaScript і суміжних технологій. Воно надає глибоку інтеграцію з фреймворками типу React, Angular і Vue, має потужний механізм аналізу коду, вбудований дебагер, інструменти тестування, роботу з базами даних і зручне підсвічування помилок. WebStorm відрізняється високою продуктивністю, але потребує ліцензії та більше ресурсів системи.

Atom [26] — це текстовий редактор з відкритим кодом, розроблений GitHub, який був популярним серед веброзробників завдяки своїй гнучкості та можливості глибокої кастомізації. Він підтримує плагіни для роботи з JavaScript, Node.js і React, проте з часом поступився в популярності більш продуктивним і стабільним середовищам програмування на зразок Visual Studio Code. Через припинення активної розробки він сьогодні рідше використовується у професійних системах.

Порівняння за критеріями цих середовищ програмування продемонстровано у таблиці 3.1.

Таблиця 3.1 – Порівняння середовищ програмування

Критерій	Visual Studio Code	WebStorm	Atom
Підтримка React, Node.js, MySQL	+	+	+/-
Швидкість роботи та стабільність	+	+/-	+
Доступність (вартість)	+	-	+
UX і зручність розробки	+	+/-	+
Інтеграція з Git, терміналом, дебагінгом	+	+	+/-
Актуальність і розвиток	+	+	-

Зробимо висновок, Visual Studio Code — легкий та гнучкий редакторі коду з підтримкою розширень, автодоповнення, вбудованого терміналу та інтеграції з системами керування версіями. Це середовище дозволяє зручно працювати як з фронтендом, так і з бекендом у межах одного інтерфейсу, через що є найкращим для цього проекту.

Отже, вибір React, Node.js, Express, MySQL і Visual Studio Code був зумовлений прагненням забезпечити просту в підтримці, продуктивну, гнучку та доступну для користувача систему. Обрані інструменти дозволили не лише реалізувати необхідний функціонал, а й адаптувати інтерфейс та логіку роботи до потреб нейровідмінної цільової аудиторії.

3.2 Розробка модуля системи ложок

При розробці модуля системи ложок було враховано усі особливості реалізації концепції обмеженої кількості ресурсів користувача згідно з підходом «Теорії ложок», який особливо актуальний для нейровідмінних осіб. Цей підхід дає змогу користувачам планувати свою діяльність, орієнтуючись на власний енергетичний ресурс протягом дня, що сприяє уникненню перевантаження та формуванню здорового розпорядку.

Розроблений модуль реалізує можливість встановлення максимальної кількості «ложок» — одиниць умовної енергії — на один день. Користувач може задати власне значення ресурсу, яке зберігається у базі даних та використовується для контролю навантаження. Особливістю реалізації є автоматичне обмеження частоти оновлення значення ложок: дана операція дозволяється лише один раз на добу, що запобігає частим змінам і підтримує стабільність обліку.

Додатково система виконує підрахунок кількості вже використаних ложок на основі створених подій у календарі. Для цього здійснюється обробка кожної події з прив’язкою до дати та ресурсу (поля resource), що вона споживає. У результаті визначається залишок енергії користувача, який також візуалізується

у вигляді відсоткового співвідношення відносно максимальної встановленої кількості.

З технічного боку, реалізацію виконано за допомогою функціонального компоненту SpoonSystem.jsx, що використовує React-хуки useState та useEffect для керування станами. Основними змінними стану є spoons — кількість доступних ложок, usedToday — кількість вже витрачених ресурсів, dbSpoons — значення, збережене у базі, та alreadySetToday — індикатор того, чи оновлював користувач значення ложок поточної доби.

При завантаженні компонента відбувається перевірка локального сховища на наявність збережених даних користувача. У випадку, якщо дані знайдено, відправляються запити до серверу для отримання актуального значення ложок (/api/spoons/:userId) та подій на поточну дату (/events?user_id=...). Після обробки відповіді від API відбувається фільтрація подій за поточною датою та обчислення сумарного використаного ресурсу, який присвоюється змінній usedToday. Усі обчислення та перевірки виконуються автоматично при кожному оновленні або повторному вході користувача в систему (рис. 3.1).

```
48  const fetchUsedToday = async () => {
49    try {
50      const today = new Date();
51      const todayYMD = formatDateForSQL(today);
52
53      const res = await axios.get(`http://localhost:5000/events?user_id=${user.id}`);
54      const total = res.data
55        .filter((ev) => {
56          const evDate = new Date(ev.event_date);
57          return formatDateForSQL(evDate) === todayYMD;
58        })
59        .reduce((sum, ev) => sum + (parseInt(ev.resource) || 0), 0);
60
61      setUsedToday(total);
62    } catch (err) {
63      console.error("Помилка завантаження подій:", err);
64    }
65  };
```

Рисунок 3.1 – Реалізація отримання загального ресурса користувача

А у ситуації, якщо користувач намагається встановити меншу кількість ложок, ніж уже використано — система видає попередження, що зображено на рисунку 3.2.

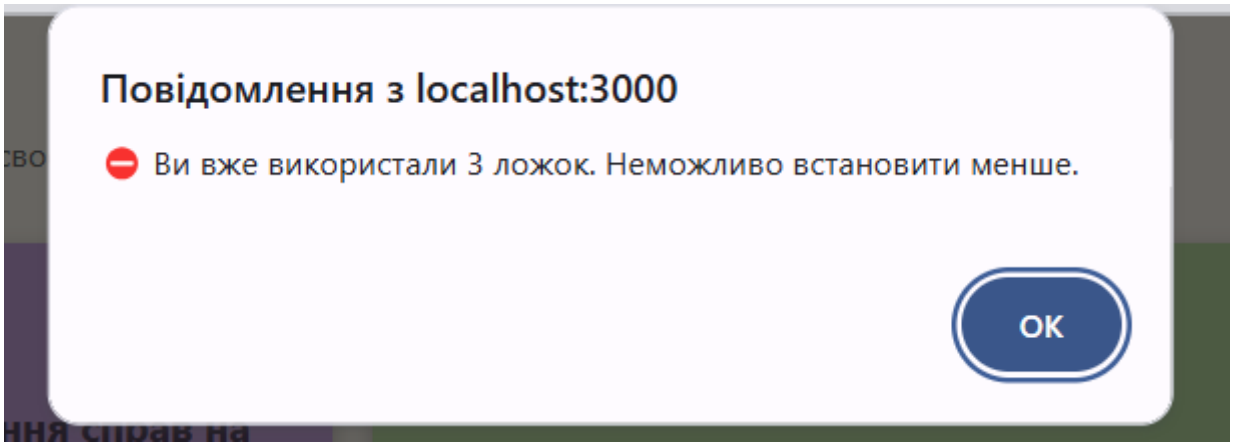

Рисунок 3.2 – Попередження про встановлення меншої кількості ложок, ніж вже використано

Лістинг перевірки на зміну кількості ложок та як описано попередження про це зображено на рисунку 3.3.

```
79  if (spoons < usedToday) {
80    alert(`⛔ Ви вже використали ${usedToday} ложок. Неможливо встановити менше.`);
81    return;
82  }
```

Рисунок 3.3 – Лістинг перевірки на зміну кількості ложок

З метою наочного відображення доступного енергетичного ресурсу для користувача у модулі реалізовано візуальну шкалу заповнення на головній сторінці. Вона формується на основі обчислення відсоткового співвідношення між встановленою кількістю ложок на день та максимально допустимим значенням, яке за замовчуванням може дорівнювати 14. Змінна fillPercentage визначає це співвідношення та використовується для побудови відповідного графічного індикатора. На рисунку 3.4 зображено шматок лістингу програми, де показано як позначене значення користувачем відправляється на сервер.

```
76  const handleSubmit = async () => {
77    if (!user) return;
78
79    if (spoons < usedToday) {
80      alert(`⛔ Ви вже використали ${usedToday} ложок. Неможливо встановити менше.`);
81      return;
82    }
83
84    try {
85      const res = await axios.post("http://localhost:5000/api/spoons", {
86        userId: user.id,
87        spoons: Number(spoons),
88      });
89      setSubmitted(true);
90      setAlreadySetToday(true);
91      setDbSpoons(Number(spoons));
92      setMessage("Ложки оновлено ✅");
93    } catch (error) {
94      alert(error.response?.data?.message || "Помилка при збереженні");
95    }
96  };
```

Рисунок 3.4 – Лістинг надсилання ресурсу на сервер

Такий підхід дозволяє користувачу візуально і швидко оцінити залишок доступного ресурсу, що сприяє кращому плануванню справ та раціональному розподілу зусиль протягом дня. Відображення відсоткового заповнення (рис. 3.5) є інтуїтивно зрозумілим і підсилює загальну зручність інтерфейсу системи.

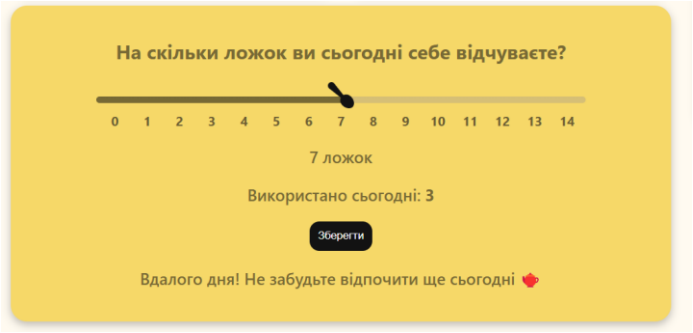

Рисунок 3.5 – Повідомлення якщо користувач вкаже менше ресурсу, ніж запланував

Отже, можна зазначити, що модуль системи ложок є важливою складовою програмного застосунку, орієнтованого на підтримку користувачів у повсякденному тайм-менеджменті.

3.3 Розробка модуля нотаток користувача

Модуль нотаток у складі програмного забезпечення реалізовано з метою забезпечення користувачів зручним інструментом для збереження особистих думок, нагадувань або корисної інформації у структурованому форматі. Основна ідея полягає у створенні простого, але функціонального середовища, де кожна нотатка може бути позначена відповідними мітками (тегами), а також легко знайдена за ключовими словами.

Реалізований функціонал модуля включає створення нових нотаток, їх редагування та видалення. Окрім базового зберігання тексту, зроблена можливість додавання до кожної нотатки одного або кількох тегів, що у подальшому дозволяє здійснювати фільтрацію за категоріями. Також реалізовано механізм пошуку нотаток за назвою або вмістом, що підвищує швидкість роботи з великим обсягом збережених записів. Усі дані зберігаються на сервері з чіткою прив’язкою до ідентифікатора користувача, що гарантує конфіденційність та персоналізований доступ.

За реалізацію логіки та інтерфейсу взаємодії з нотатками відповідає компонент Notes.jsx (рис. 3.6), який використовує сучасний підхід до керування станом у React. Основними змінними стану є: allNotes — повний список нотаток користувача, notes — список, який може бути відфільтрованим відповідно до заданих параметрів, tags — доступні теги, selectedTags — список обраних тегів для фільтрації, а також змінні modalNote, modalOpen, isAddModalOpen, які відповідають за керування модальними вікнами, що відкриваються під час створення чи редагування нотаток.

Для взаємодії з серверною частиною застосунку використовуються HTTP-запити до API. При завантаженні компоненту відправляється запит на отримання списку нотаток за відповідним ідентифікатором користувача, після чого кожна нотатка додатково доповнюється пов’язаними тегами. Аналогічно, за допомогою окремого запиту завантажується повний список доступних тегів, що може використовуватись у подальшому для редагування або створення нових записів.

```
const fetchNotes = async () => {
  const res = await axios.get(`http://localhost:5000/notes?user_id=${user.id}`);
  const notesData = res.data;

  const notesWithTags = await Promise.all(
    notesData.map(async (note) => {
      const tagsRes = await axios.get(`http://localhost:5000/note_tags?note_id=${note.id}`);
      return { ...note, tags: tagsRes.data };
    })
  );

```

Рисунок 3.6 — Лістинг отримання нотаток та міток з серверу

У рамках реалізації модуля нотаток важливу роль відіграють модальні вікна (рис. 3.7), які забезпечують зручний інтерфейс взаємодії користувача з функціоналом створення та редагування записів. Їх реалізовано у вигляді окремого компоненту AddNoteModal.jsx, що дозволяє зберігати логіку керування уніфікованою, а інтерфейс — компактним і узгодженим зі структурою програми.

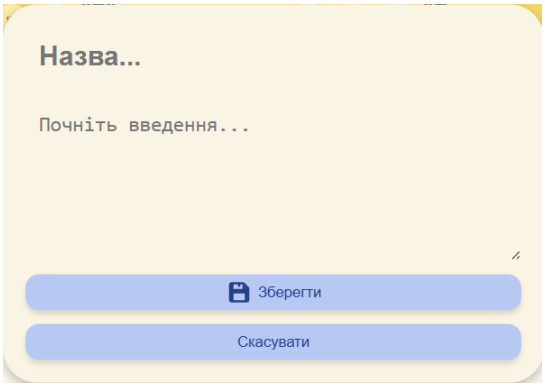

Рисунок 3.7 – Вигляд модального вікна для створення нотатки

Модальні вікна використовуються у двох основних сценаріях: перший — додавання нової нотатки, активація якого здійснюється за допомогою булевого стану isAddModalOpen; другий — редагування існуючої нотатки, керування яким відбувається за допомогою змінних modalOpen та modalNote, що містять відповідно стан відкриття вікна та інформацію про редагований запис.

При ініціалізації модального вікна редагування викликається функція openModal(note), яка виконує асинхронний запит до серверної частини системи з метою отримання повного списку тегів, прикріплених до вибраної нотатки.

Після цього форма заповнюється актуальними даними, що дозволяє користувачу без затримок внести зміни та зберегти оновлення. Лістинг відкриття модального вікна зображено на рисунку 3.8.

```
92  const openModal = async (note) => {
93    try {
94      const tagsRes = await axios.get(`http://localhost:5000/note_tags?note_id=${note.id}`);
95      const noteWithTags = { ...note, tags: tagsRes.data };
96      setModalNote(noteWithTags);
97      setModalOpen(true);
98    } catch (error) {
99      console.error("Помилка при завантаженні міток:", error);
100   }
101 };
```

Рисунок 3.8 – Лістинг відкриття модального вікна для створення або редагування нотатки

Модальне вікно автоматично закривається після оновлення або скасування дій, що і продемонстровано у лістингу 3.9.

```
21  const addNote = async () => {
22    if (!newNote.content.trim()) {
23      alert("Текст нотатки не може бути порожнім!");
24      return;
25    }
26    try {
27      await axios.post(`http://localhost:5000/notes`, {
28        title: newNote.title.trim() || "",
29        content: newNote.content.trim(),
30        user_id: user.id,
31      });
32      onNoteAdded(); // Оновлюємо список нотаток
33      setNewNote({ title: "", content: "" });
34      onClose();
35    } catch (error) {
36      console.error("Помилка при додаванні нотатки:", error);
37    }
38  };
```

Рисунок 3.9 — Лістинг збереження нотатки

Користувач може фільтрувати нотатки за тегами, які зручно вибираються через спеціальний фільтр. Також зроблено пошук за ключовими словами в заголовку або тексті нотатки. Цей функціонал зображено на рисунку 3.10.

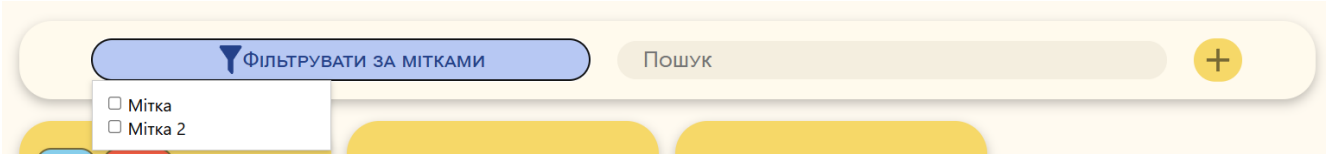

Рисунок 3.10 – Демонстрація функціоналу пошуку і фільтрування

Отже, можна зробити висновок, що модуль нотаток користувача є важливою складовою програмного застосунку, спрямованою на підвищення продуктивності та зручності роботи з особистою інформацією. Завдяки підтримці тегів, пошуку та фільтрації, користувачі можуть швидко структурувати та знаходити потрібні записи. Реалізація модальних вікон у вигляді окремого компонента забезпечує зручність взаємодії та дозволяє створювати або редагувати нотатки без виходу з основного інтерфейсу.

3.4 Розробка модулю календаря

Одним із ключових функціональних компонентів програмного застосунку є модуль календаря, який забезпечує зручне візуальне планування та перегляд подій користувача в різних режимах. Реалізація календаря є центральною частиною інтерфейсу, оскільки саме через нього здійснюється основна взаємодія з особистим розкладом, подіями, а також системою енергетичних ресурсів (ложок).

З технічного боку календар реалізовано на основі бібліотеки React із поділом інтерфейсів на три основні компоненти відображення: DayView, WeekView та MonthView. Кожен із них представлений окремим функціональним компонентом (DayView.jsx, WeekView.jsx, MonthView.jsx) і відповідає за рендеринг подій відповідно до обраного режиму перегляду.

Компонент Calendar.jsx виступає центральним обгортуючим елементом, який керує перемиканням між режимами перегляду та ініціалізує завантаження відповідних даних із сервера. Усі події, пов’язані з користувачем, запитуються через API і фільтруються відповідно до обраної дати чи періоду. Для зручності реалізовано спеціальну обгортку DayViewWrapper.jsx, яка відповідає за передачу

даних у відповідні підкомпоненти та взаємодію з іншими модулями системи, зокрема системою ложок.

На прикладі WeekView.jsx (рис. 3.11), у рамках компонента реалізовано розбиття на дні тижня з можливістю точного позиціонування подій за часом, що є особливо корисним для щоденного або тижневого планування. Кожна подія містить інформацію про дату, назву, час початку й завершення, а також кількість ресурсів (ложок), які вона споживає. Усі ці дані використовуються також у модулі системи ложок для підрахунку вже використаного ресурсу на поточну дату. Схожим чином працюють і DayView.jsx та MonthView.jsx.

```
return (
  <div className="week-view-container">
    <div className="weekly-timeline-header">
      <div className="cell hour-label">Година</div>
      {daysArray.map((day, i) => (
        <div className="cell day-header" key={i}>
          <div className="week-day-number">{day.dateNumber}</div>
          <div className="week-day-name">{day.label}</div>
        </div>
      ))}
    </div>

    <div className="weekly-timeline-body">
      {HOURS.map((hour) => {
        const hourLabel = `${String(hour).padStart(2, "0")}:00`;
        return (
          <div className="weekly-timeline-row" key={hour}>
            <div className="cell hour-cell">{hourLabel}</div>
            {daysArray.map((day, idx) => {
              const dayEvents = events.filter((ev) => {
                const evHour = parseInt(ev.start_time?.split(":" )[0], 10);
                return isEventOnDate(ev, day.date) && evHour === hour;
              });
              return (
                <div className="cell day-hour-cell" key={idx}>
                  {dayEvents.map((ev) => (
                    <div
                      className="event-block"
                      key={`${ev.id}-${day.ymd}`}
                      onClick={() => onEdit && onEdit(ev)}
                    >
                      <span className="event-title">{ev.title}</span>
                      <br />
                      <span className="event-time">
                        {ev.start_time.slice(0, 5)} - {ev.end_time.slice(0, 5)}
                      </span>
                    </div>
                  ))}
```

Рисунок 3.11 – Лістинг розбиття на дні тижня

Календар підтримує інтерактивність: користувач може відкривати вікна створення або редагування подій, які відкриваються у відповідних компонентах (NewEventPage.jsx, EventEditPage.jsx). Дані, створені в цих вікнах, передаються через API до серверної частини, де зберігаються в базі даних. Оновлення подій миттєво відображається в інтерфейсі без потреби оновлення сторінки.

Реалізація компоненту NewEventPage.jsx відображена на рисунку 3.12, а саме додавання нової події. А на рисунку 3.13 подано програмну реалізацію EventEditPage.jsx, а саме алгоритм редагування події.

```
const NewEventPage = () => {
  const navigate = useNavigate();
  const location = useLocation();

  const handleSave = () => {
    navigate("/calendar");
  };

const handleCancel = () => {
  const params = new URLSearchParams(location.search);
  const date = params.get("date");

  console.log("🔴 handleCancel:");
  console.log("URLSearchParams date:", date);
  console.log("location.state?.from:", location.state?.from);

  if (date) {
    console.log("➡️ Navigating to:", `/calendar/day/${date}`);
    navigate(`/calendar/day/${date}`);
  } else if (location.state?.from) {
    console.log("➡️ Navigating to state.from:", location.state.from);
    navigate(location.state.from);
  } else {
    console.log("➡️ Navigating to fallback: /calendar/home");
    navigate("/calendar/home");
  }
};
```

Рисунок 3.12 –Лістинг додавання нової події

```
7   const EventEditPage = () => {
8     const { id } = useParams(); // подія id з URL
9     const navigate = useNavigate();
10    const [event, setEvent] = useState(null);
11
12    useEffect(() => {
13      const user = JSON.parse(localStorage.getItem("user"));
14      if (!user?.id) return;
15
16      axios
17        .get(`http://localhost:5000/events?user_id=${user.id}`)
18        .then((res) => {
19          const all = res.data || [];
20          const found = all.find((ev) => ev.id.toString() === id);
21          if (found) setEvent(found);
22          else navigate("/calendar"); // якщо не знайдено — назад
23        })
24        .catch((err) => {
25          console.error("Помилка завантаження події:", err);
26          navigate("/calendar");
27        });
28    }, [id, navigate]);
29
30    const handleSave = () => {
31      navigate("/calendar");
32    };
```

Рисунок 3.13 – Лістинг редагування події

Візуальна частина календаря реалізована з урахуванням принципів доступності та зручності користування. Використання адаптивних CSS-файлів (DayView.css, MonthView.css, WeekView.css) дозволяє забезпечити коректне відображення як на великих екранах, так і на мобільних пристроях.

Отже, модуль календаря є ядром системи планування, який не лише забезпечує гнучке візуальне представлення розкладу, але й тісно інтегрується з іншими модулями системи, зокрема нотатками та системою ложок, створюючи цілісну інфраструктуру персонального тайм-менеджменту.

3.5 Розробка модулю інтерактивного тамагочі

Модуль інтерактивного тамагочі розроблено як самостійний функціональний блок у складі вебзастосунку, призначений для створення віртуального улюбленця та подальшого догляду за ним. Його головна ідея полягає в тому, щоб за допомогою ігрового підходу мотивувати користувача до тайм-менеджменту, розвитку відповідальності та підтримки власного емоційного балансу. Такий формат взаємодії сприяє формуванню позитивної поведінки в умовах стресу або перевантаження, що є особливо актуальним для нейровідмінної аудиторії.

З технічної точки зору, модуль реалізовано на основі бібліотеки React шляхом створення кількох спеціалізованих компонентів, кожен з яких виконує окрему роль у загальній логіці взаємодії з користувачем. Центральним елементом є компонент TamagotchiPage.jsx, який виступає в ролі керуючого елемента модуля та визначає, яка саме частина інтерфейсу має бути відображена у відповідний момент.

Після завантаження сторінки компонент ініціалізує отримання інформації про поточного користувача з локального кешу. У разі відсутності даних виконується запит до серверного API для перевірки наявності вже створеного віртуального улюбленця. В залежності від результатів перевірки, TamagotchiPage перемикається на відповідний режим візуалізації. Якщо дані ще не завантажені, відображається стан «loading». Якщо тваринка ще не створена,

система пропонує вступний екран «intro» або перехід до етапу вибору улюбленця — режим «select». Якщо улюбленець уже створений, користувачу відкривається основний екран «main» із можливістю взаємодії, догляду та перегляду його стану. У разі звернення до магазину, що є додатковою функцією, активується режим «shop», де можна переглядати й купувати аксесуари для персоналізації віртуального персонажа.

На рисунку 3.14 продемонстровано шматок коду, який відповідає за реалізацію режимів відображення компонента.

```
22    const loadPet = async (userId) => {
23    try {
24      const res = await axios.get(`http://localhost:5000/api/pets/${userId}`
25      if (res.data && res.data.name) {
26        setPet(res.data);
27        setStep("main"); // показує головний екран
28      } else {
29        setStep("intro"); // показує вступ
30      }
31    } catch (error) {
32      console.log("🐾 Помилка завантаження улюбленця:", error);
33      setStep("intro"); // якщо помилка – все одно показує вступ
34    }
35  };
36
```

Рисунок 3.14 – Лістинг режимів відображення компонента

Другий компонент, PetIntroScreen.jsx, який являє собою вступний екран. Мета цього компонента — ознайомити користувача з концепцією віртуального улюбленця. Він містить коротку текстову інформацію та кнопку «Я хочу додати собі компаньйона!», яка перемикає стан на «select». На рисунку 3.15 представлено скріншот відображення компоненту у застосунку.

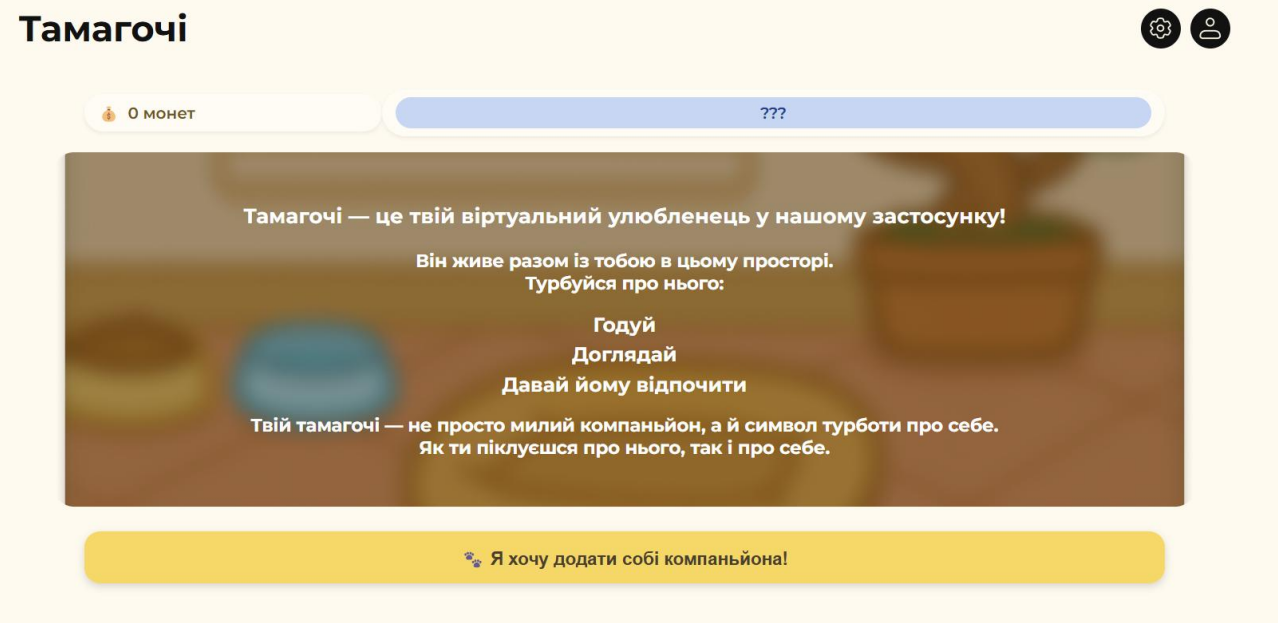

Рисунок 3.15 – Відображення компонента PetIntroScreen.jsx у застосунку

Третій компонент PetSelectionScreen.jsx дозволяє користувачу вибрати тваринку та ввести ім’я улюбленця. Після підвердження дій надсилається POST-запит на створення. На рисунку 3.16 продемонстровано шматок коду реалізації надсилання запиту до серверу на створення тамагочі.

```
41  const handlePetCreated = async (type, name) => {
42    try {
43      console.log("🐾 Надсилаємо:", { user_id: user.id, name, type });
44      const res = await axios.post("http://localhost:5000/api/pets", {
45        user_id: user.id,
46        name,
47        type,
48      });
49      setPet({ ...res.data, type, name });
50      setStep("main");
51    } catch (error) {
52      console.error("❌ Помилка створення улюбленця:", error);
53    }
54  };
55
```

Рисунок 3.16 – Лістинг надсилання запиту до серверу на створення

Основний екран MainPetScreen.jsx є головним місцем взаємодії з тамагочі. Дозволяється виконувати три дії «Спати», «Помити» та «Погодувати», що змінюють значення відповідно. Також виводиться ім’я та зображення улюбленця, враховуючи обраний аксесуар. Можливе редагування імені з live-зміною. Для редагування імені створено окремий об’єкт PetNameEditor.jsx. На рисунку 3.17 зображено шматок коду реалізації редагування імені улюбленця.

```
5   const [editing, setEditing] = useState(false);
6   const [newName, setNewName] = useState(name);
7
8   const handleSave = () => {
9     if (newName.trim().length < 2) {
10      alert("Ім’я має бути не коротше 2 символів");
11      return;
12    }
13    onSave(newName.trim());
14    setEditing(false);
15  };
```

Рисунок 3.17 – Лістинг редагування імені тамагочі

Також наявний магазин аксесуарів в компоненті AccessoryShopScreen.jsx. Магазин дозволяє отримати список доступних аксесуарів, побачити активний аксесуар, купити аксесуар та активувати потрібний аксесуар. На рисунку 3.18 представлено інтерфейс цього магазину.

Рисунок 3.18 – Інтерфейс AccessoryShopScreen.jsx

Окрім цього створено картку стану тамагочі в комопненті PetSummaryCard.jsx. Використовується поза основним екраном для короткого огляду стану улюбленця на головному екрані/екрані профілю. Містить такі дані як ім’я, зображення, рівні трьох потреб, які візуалізуються через PieChart з бібліотеки Recharts, підрахунок щастя на основі середнього значення показників та кнопку створення тваринки, якщо її не існує. На рисунку 3.19 представлено інтерфейс картки стану улюбленця.

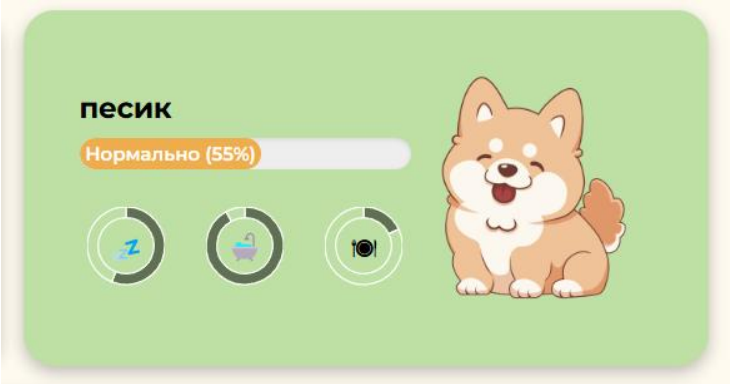

Рисунок 3.19 – Інтерфейс PetSummaryCard.jsx

Взаємодія з модулем інтерактивного тамагочі здійснюється з урахуванням індивідуального профілю користувача, ідентифікованого за допомогою унікального параметра user_id. Уся інформація, пов’язана з профілем користувача та станом віртуального улюбленця, зберігається в кеші, що забезпечує оперативну реакцію інтерфейсу, мінімізує затримки у взаємодії та покращує загальну продуктивність системи.

Отже, модуль поєднує елементи ігрової механіки з практичним впливом на поведінку користувача, створюючи сприятливе середовище для формування здорових звичок. Завдяки регулярній взаємодії з віртуальним улюбленцем, користувач отримує дозу дофаміну в процесі, що дуже важливо для нейровідмінних людей, а також переводить планування в ігрову форму.

Візуальна складова модуля, разом з інтерактивним інтерфейсом, сприяє позитивному емоційному зворотному зв’язку: піклування про цифрового компаньйона асоціюється з турботою про себе, що закріплюється у свідомості

користувача як корисна та приємна дія. Такий підхід дозволяє не лише підвищити рівень залучення до застосунку, але й зробити його цінним інструментом для підтримки емоційного балансу та стабільності в повсякденному житті.

3.6 Висновки

У третьому розділі було обґрунтовано вибір технологій та інструментів, що використовувались для розробки програмних модулів. Виконавши аналіз програмного продукту та його задач було обрано комбінацію мов програмування JavaScript та бібліотеки React з Node.js Також було детально розглянуто процес реалізації основних програмних модулів системи, зокрема модуля системи ложок та модуля нотаток користувача. У межах розділу було описано структуру програмних компонентів, механізми керування станом та обробки користувацьких даних, а також реалізацію логіки обробки інформації через модальні вікна. Особливу увагу було приділено інтеграції функціональних частин між собою та забезпеченню зручності користування, персоналізації та адаптивності інтерфейсу.

4 ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ ТАЙМ-МЕНЕДЖМЕНТУ НЕЙРОВІДМІННИХ ЛЮДЕЙ

4.1 Тестування системи

Тестування програмного забезпечення [27] є ключовим етапом у процесі розробки будь-якого застосунку, оскільки воно дозволяє виявити помилки, неточності у логіці та забезпечити відповідність функціоналу вимогам, визначеним на етапі проєктування. Під тестуванням розуміють процес перевірки якості роботи програмного продукту з метою виявлення невідповідностей між фактичною поведінкою системи та очікуваними результатами. Це дозволяє підвищити надійність програмного забезпечення, покращити роботу його модулів і гарантувати коректність взаємодії між усіма складовими системи.

Для початку тестування необхідно встановити тестову версію програми на комп'ютер з операційною системою Windows, яка буде використовуватись під час тестування. Перед початком тестування також важливо забезпечити належну документацію, включаючи специфікації вимог та план тестування. Проведення тестування включає кілька етапів та видів тестів.

Початковим етапом процесу тестування виступає функціональна перевірка програмного забезпечення [28], метою якої є підтвердження правильності реалізації основного функціоналу. Вона охоплює тестування операцій введення та виведення даних, а також аналіз реакції системи на помилкові або некоректні вхідні значення. Особливу увагу приділяють відповідності дій програми заданій бізнес-логіці, що дозволяє переконатися у правильності виконання закладених сценаріїв.

Другим етапом виступає тестування відмовостійкості [29], в межах якого визначається здатність системи протистояти збоям та стабільно функціонувати в нестандартних умовах. Під час цього етапу моделюються ситуації, такі як: введення некоректних або непередбачуваних даних, імітуються порушення у взаємодії з зовнішніми модулями чи системними компонентами. Такий підхід

дозволяє оцінити, наскільки система здатна повернутися до нормального функціонування після виникнення критичних помилок.

Після завершення основного функціонального тестування та перевірки відмовостійкості виконується оцінювання продуктивності програмного забезпечення [30]. На цьому етапі проводяться тести швидкодії, які дозволяють встановити час виконання ключових операцій, а також виміряти швидкість реагування системи на користувацькі запити. Це дає змогу оцінити швидкість роботи програми за умови навантаження або інтенсивної взаємодії.

Окрему роль у комплексі перевірок відіграє тестування сумісності [31], яке охоплює перевірку коректності функціонування програмного продукту на різних платформах, версіях операційних систем та апаратному забезпеченні. Завдяки цьому виявляються можливі конфлікти з іншими програмами або обмеження у роботі за певних технічних умов. Такий підхід дозволяє переконатися в універсальності розробленого програмного забезпечення та його готовності до експлуатації у реальному середовищі.

Проведемо спочатку функціональне тестування та тестування відмовостійкості.

Було перевірено реакцію вебзастосунку на введення неправильних даних для входу в систему. Якщо користувач вводить некоректну електронну пошту або пароль, програма виявляє помилку автентифікації та виводить відповідне повідомлення про неправильні облікові дані. Така перевірка дозволяє впевнитися, що система забезпечує базовий рівень захисту та не допускає несанкціонованого доступу.

У процесі тестування перевірялися різні варіанти помилкових введень, зокрема порожні поля, неправильний формат email-адреси та неправильне поєднання email/пароль. В усіх випадках система успішно ідентифікувала помилку й інформувала користувача про те, що введено неправильну електронну пошту або пароль.

На рисунку 4.1 зображено приклад інтерфейсу з повідомленням про помилку входу, що виникає внаслідок введення недійсних облікових даних. Така

функціональність є необхідною для забезпечення зручності користування та гарантує контроль доступу до персоналізованих даних користувача.

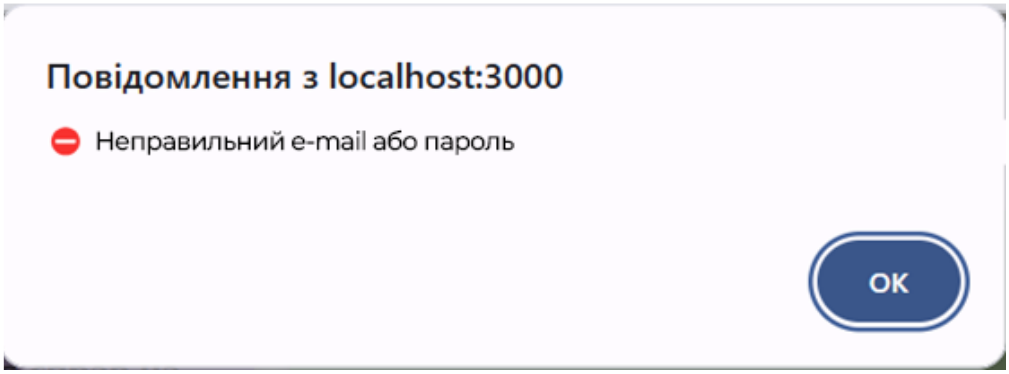

Рисунок 4.1 – Невдалий вхід у застосунок (неправильний пароль чи електронна пошта)

Було перевірено поведінку модуля системи ложок у ситуації, коли користувач намагається встановити меншу кількість доступних ресурсів (ложок), ніж уже було витрачено протягом поточного дня. Такий сценарій є критично важливим для збереження коректного балансу енергетичного ресурсу та недопущення порушення логіки обліку витрат.

Під час тестування спершу в календар було додано одну або кілька подій із зазначеним ресурсом (наприклад, по 3 ложки), внаслідок чого загальне використання за день склало певне значення. Далі користувач спробував вручну встановити загальну кількість ложок на день менше, ніж уже витрачено. У цьому випадку система правильно зафіксувала конфлікт даних та вивела повідомлення з попередженням про те, що встановлення меншої кількості ложок є неможливим. Водночас зміна значення не збереглася, що забезпечило збереження цілісності даних.

На рисунку 4.2 зображено інтерфейс із повідомленням про помилку встановлення меншої кількості ложок. Така перевірка засвідчує, що система успішно контролює допустимі межі редагування ресурсу та запобігає потенційним логічним помилкам, пов’язаним із перепланування навантаження.

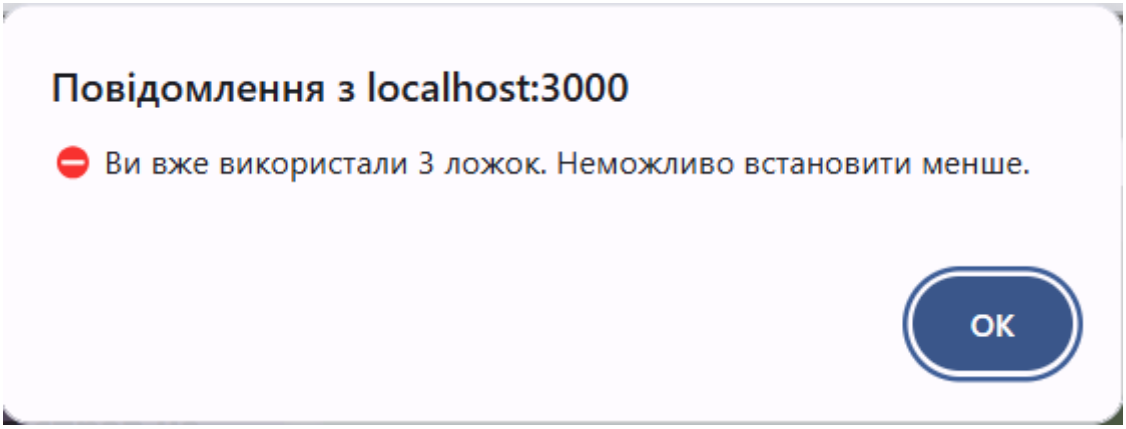

Рисунок 4.2 – Попередження про встановлення меншої кількості ложок, ніж вже використано

У рамках наступного тестового сценарію було перевірено реакцію системи на ситуацію, коли користувач намагається додати нову подію в календар у той день, коли вже було використано всі доступні ложки. Такий механізм є принципово важливим для підтримки концепції Spoon Theory, оскільки запобігає перевантаженню користувача понад визначений ліміт ресурсу.

На початку тестування було встановлено певну кількість доступних ложок на поточний день (наприклад, 10 одиниць). Далі в календар було внесено серію подій із сумарною витратою ложок, яка дорівнює максимальному ліміту. Після цього користувач спробував створити ще одну подію на цей самий день, зазначивши додаткову кількість ресурсів.

Система відреагувала відповідно до закладеної логіки: вивела повідомлення з попередженням — «Ви використали всі ложки на сьогодні. Спробуйте перепланувати день або утримайтесь від нових завдань — бережіть свій ресурс!». При цьому нову подію не було додано, і користувач не зміг перевищити встановлений щоденний ліміт.

На рисунку 4.3 зображено приклад повідомлення про перевищення ресурсу та заблоковану можливість додавання нової події. Це свідчить про успішну реалізацію контролю навантаження та підкреслює адаптивність системи до потреб користувачів, які дотримуються обмеженого планування впродовж дня.

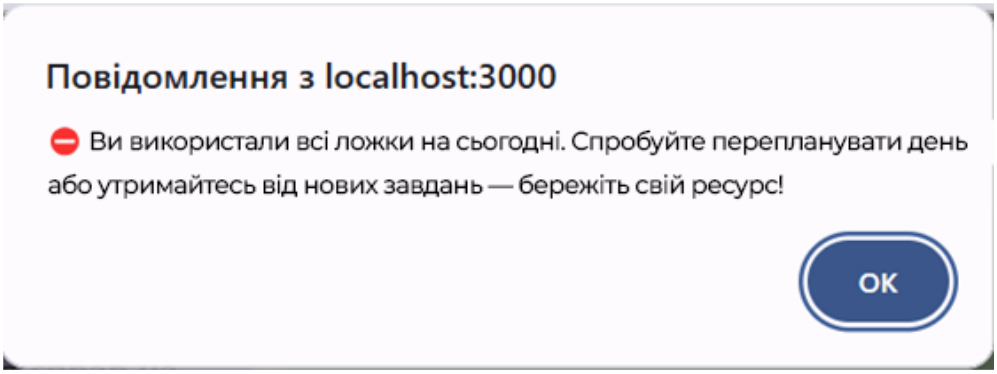

Рисунок 4.3 – Спроба використати більше ложок, ніж зазначили на день

Також на останок було перевірено коректність роботи модуля нотаток у ситуації, коли користувач намагається створити нову нотатку без вмісту. Такий тип перевірки дозволяє оцінити, наскільки система здатна запобігти збереженню неповноцінних або порожніх записів, що можуть негативно впливати на загальну структурованість інформації користувача.

Під час тестування користувач відкривав модальне вікно створення нотатки, залишав поля заголовка або вмісту порожніми та натискав кнопку збереження. У результаті система виявляла помилку в даних і виводила повідомлення про те, що текст нотатки не може бути порожнім. При цьому створення нотатки блокувалося до моменту внесення необхідної інформації.

На рисунку 4.4 зображено приклад валідації введення в модальному вікні та повідомлення, що з’являється у випадку порожнього поля. Така поведінка підтверджує реалізацію механізмів перевірки обов’язкових даних і підвищує надійність збереження змістовної інформації в системі нотаток.

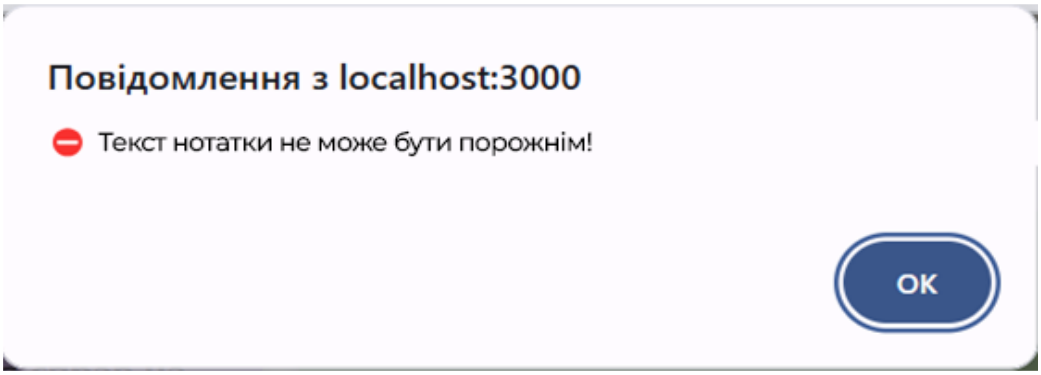

Рисунок 4.4 – Спроба додати пусту нотатку

Оцінимо продуктивність програмного забезпечення за допомогою допоміжних інструментів: Chrome DevTools, Lighthouse та React Profiler.

Chrome DevTools є невід'ємним інструментом сучасної веброзробки, інтегрованим безпосередньо в браузер Google Chrome. Цей набір засобів надає розробнику широкі можливості для аналізу, тестування та налагодження вебінтерфейсів у реальному часі. Завдяки DevTools можна безпосередньо редагувати HTML, CSS і JavaScript, переглядати стан елементів, аналізувати помилки, а також контролювати поведінку мережевих запитів, що пришвидшує розробку і покращує якість кінцевого продукту.

Особливу роль у процесі покращення продуктивності вебзастосунку відіграють інструменти Performance та Network, що входять до складу Chrome DevTools. Зокрема, панель Network дає змогу відстежувати усі запити до серверу, час їхньої обробки, обсяг переданих даних та можливі затримки у мережевій взаємодії. Результати, представлені на рисунку 4.5, свідчать про стабільну роботу API: спостерігається швидке завантаження ресурсів та відсутність критичних мережевих затримок, що є важливим показником якості інтеграції клієнтської та серверної частин застосунку.

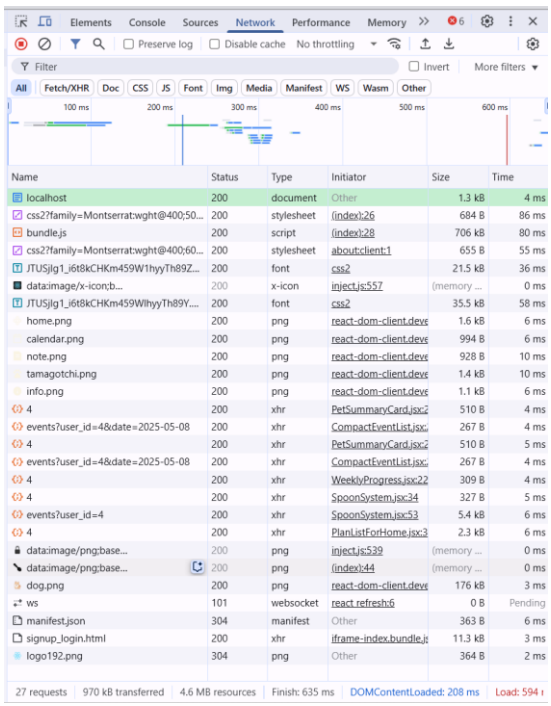

Рисунок 4.5 – Результати інструменту Network

Інструмент Performance дозволяє проаналізувати загальну швидкодію сторінки, зокрема час реакції інтерфейсу на дії користувача, швидкість рендерингу компонентів та навантаження на основний потік. Як показано на рисунку 4.6, вебінтерфейс демонструє високу продуктивність навіть під час взаємодії з динамічними компонентами. Зокрема, показник «Interaction to Next Paint», який відповідає за вимірювання часу між дією користувача і відображенням візуальних змін, підтверджує швидку реакцію системи. Це є критично важливим у контексті створення комфортного та доступного середовища для нейровідмінних користувачів, які можуть бути чутливими до затримок або перевантаженого інтерфейсу.

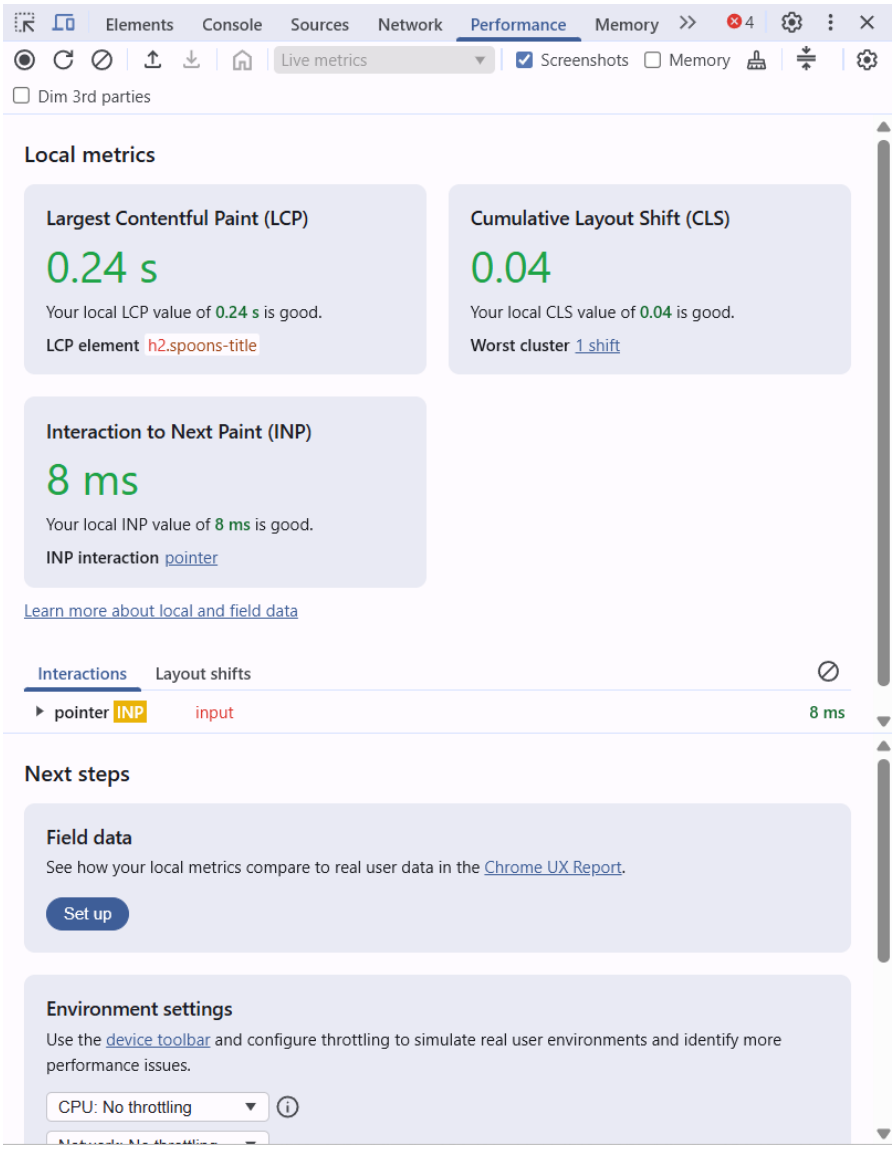

Рисунок 4.6 – Результати інструменту Performance

Lighthouse — це автоматизований інструмент аудиту вебсторінок, розроблений компанією Google з відкритим вихідним кодом. Його основне призначення — всебічна оцінка якості вебзастосунків за кількома ключовими критеріями, зокрема продуктивності, доступності, відповідності сучасним стандартам прогресивних вебзастосунків (PWA), а також покращення для пошукових систем (SEO). Lighthouse може бути запущений для будь-якої сторінки, як публічної, так і захищеної автентифікацією, що робить його універсальним засобом аналізу як на етапі розробки, так і під час фінального тестування.

У межах реалізації системи за допомогою Lighthouse було проведено аудит продуктивності кількох ключових сторінок вебзастосунку. Зокрема, результати тестування головної сторінки подано на рисунку 4.7 та 4.8.

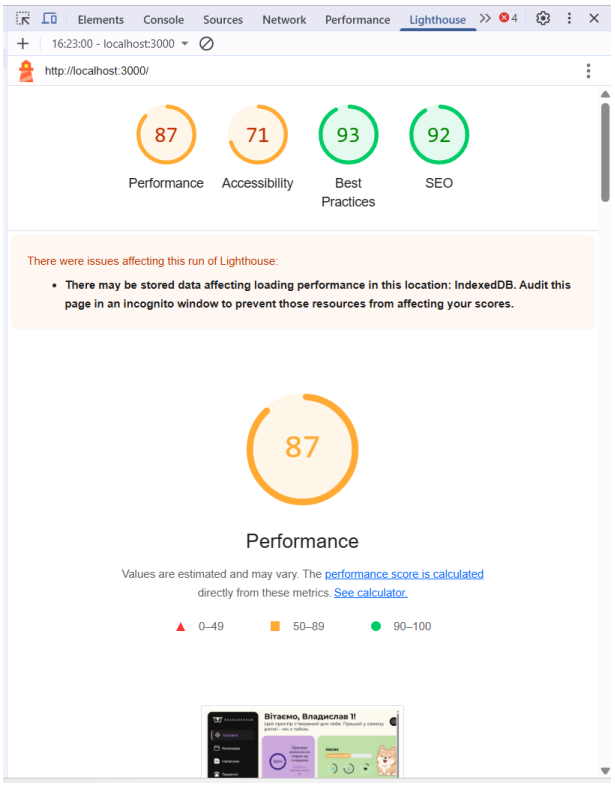

Рисунок 4.7 – Результати інструменту Lighthouse на головній сторінці

Попри наявність складних компонентів інтерфейсу та взаємодій, загальний час завантаження та відповідь на дії користувача залишаються в межах допустимих норм. Хоча фінальна оцінка не є максимальною, з огляду на

специфіку системи та інтерактивний характер її елементів, показники вважаються прийнятними і демонструють достатні показники для комфортного користування.

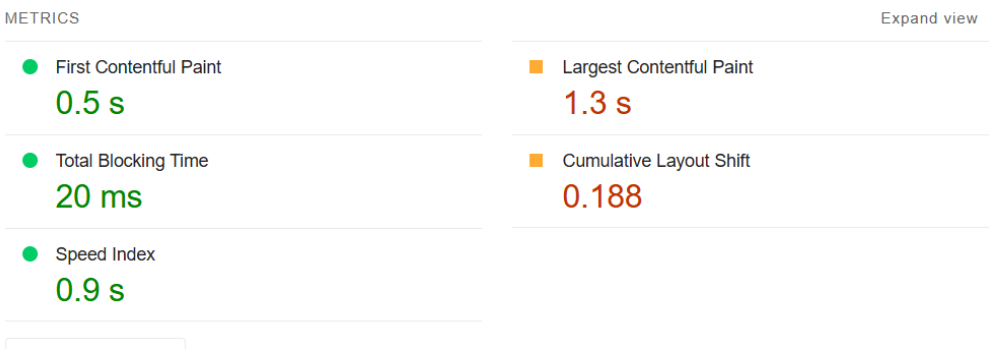

Рисунок 4.8 – Метрики інструменту Lighthouse на головній сторінці

Аналіз продуктивності окремих модулів, зокрема модуля нотаток, представлений на рисунку 4.9 та 4.10.

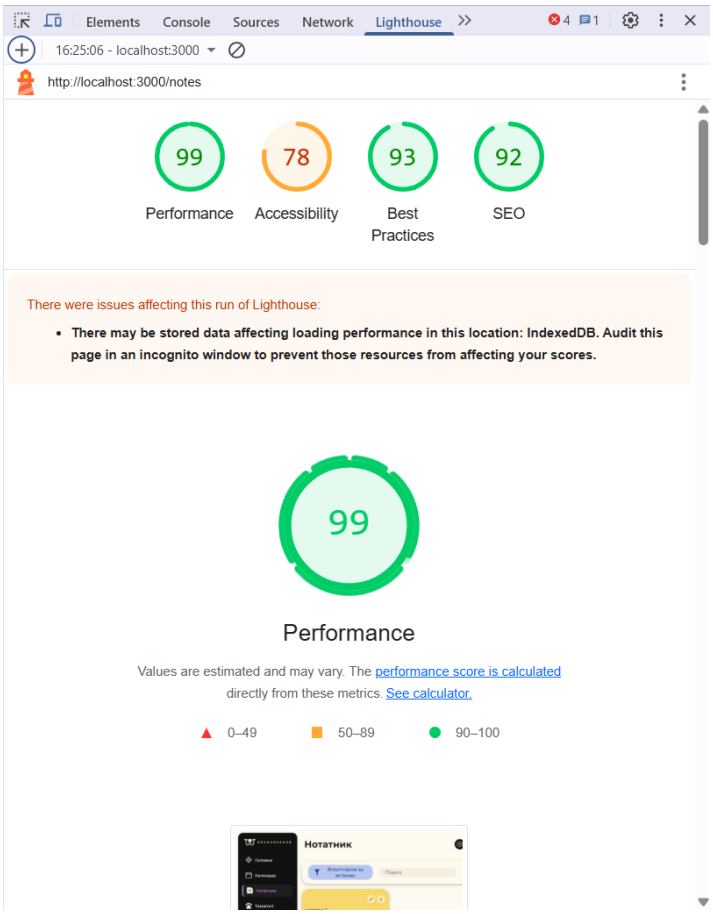

Рисунок 4.9 – Результати інструменту Lighthouse на сторінці нотаток

У цьому випадку система показала вищу швидкість, що свідчить про раціональну реалізацію логіки та оптимальну побудову користувацького інтерфейсу. Такі результати підтверджують доцільність використаних технологій і правильність обраної архітектурної структури в контексті забезпечення доступності та зручності для нейровідмінних користувачів.

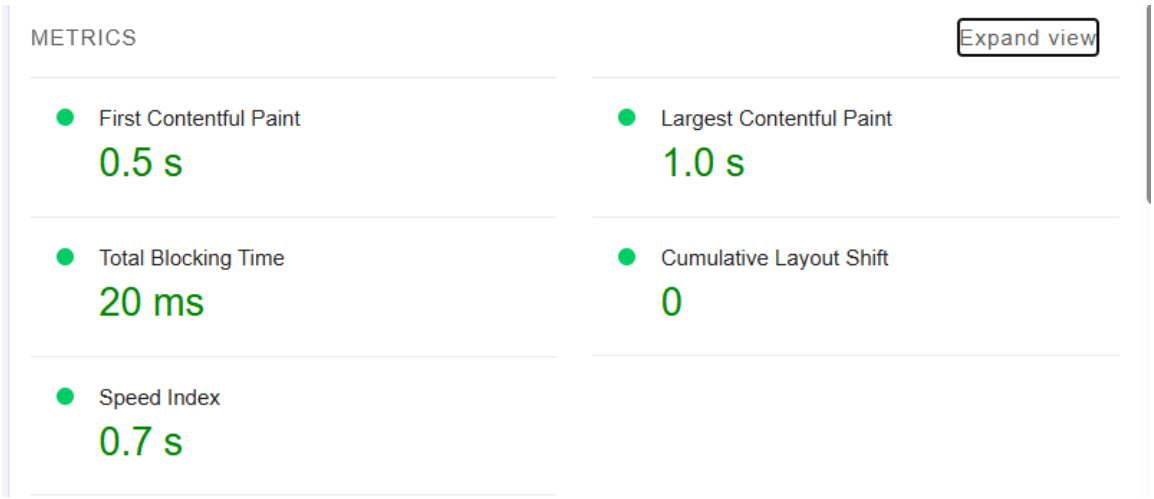

Рисунок 4.10 – Метрики інструменту Lighthouse на сторінці нотаток

React Profiler є частиною інструментарію React DevTools і використовується для аналізу продуктивності компонентів у React-застосунках. Його основна мета — виявлення компонентів, які споживають надмірні ресурси або рендеряться частіше, ніж це необхідно, що може негативно впливати на загальну швидкодію інтерфейсу та користувацький досвід.

У межах бакалаврської кваліфікаційної роботи React Profiler було використано для детального аналізу продуктивності окремих частин інтерфейсу, зокрема для виявлення надлишкових повторних рендерів та визначення «важких» компонентів, які уповільнюють завантаження сторінки. Завдяки цьому інструменту вдалося покращити деякі логічні зв’язки між компонентами та зменшити кількість непотрібних оновлень, що позитивно позначилось на часі реакції інтерфейсу, особливо при взаємодії з динамічними елементами.

Результати проведеного аналізу за допомогою React Profiler наведено на рисунку 4.11 та 4.12. Вони демонструють загальну д компонентної структури, а

також підтверджують, що більшість рендерів є обґрунтованими та не створюють надмірного навантаження на систему. Це особливо важливо для забезпечення плавної роботи вебзастосунку, що є ключовим у контексті інклюзивного дизайну для нейровідмінних користувачів.

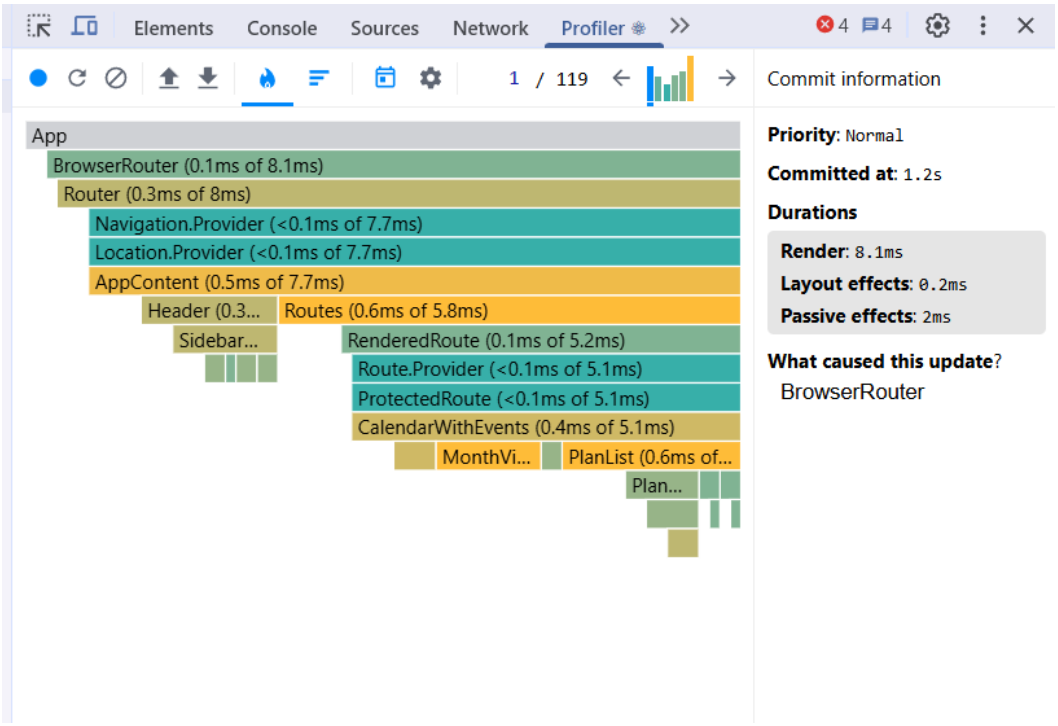

Рисунок 4.11 – Результати тестування React Profiler

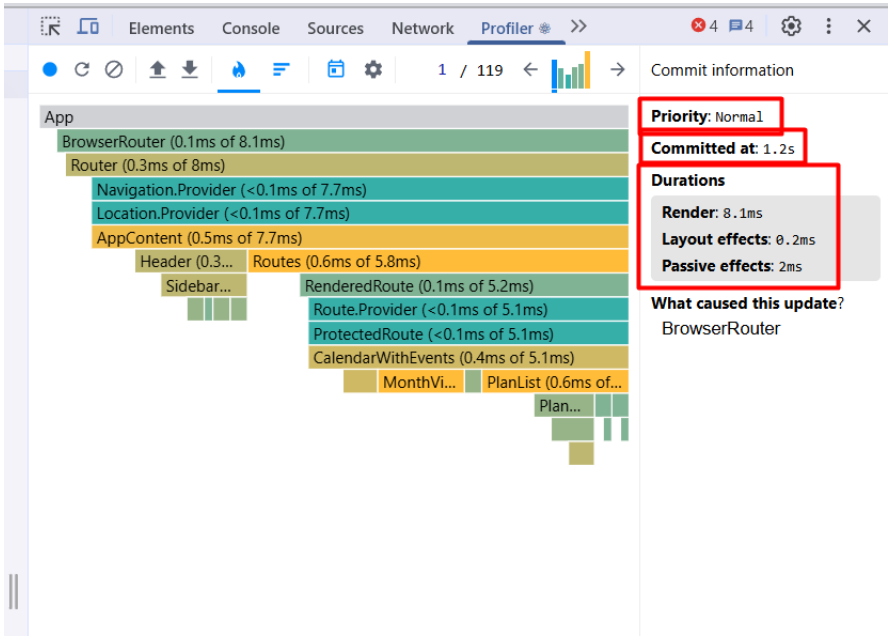

Рисунок 4.12 – Метрики тестування React Profiler

У рамках покращення продуктивності вебзастосунку було проведено роботу з удосконалення React-компонентів з метою зменшення часу початкового завантаження та підвищення швидкості рендерингу. Один із ключових кроків полягав у декомпозиції великих компонентів на менші, функціонально незалежні частини. Такий підхід дозволяє уникнути надмірних повторних рендерів і мінімізувати кількість даних, які обробляються одночасно, що є особливо важливим для інтерфейсів із великою динамікою або складною структурою.

Отже, тестування застосунку було проведено успішно.

4.2 Розробка інструкції користувача

Керівництво користувача містить інформацію щодо технічних вимог, необхідних для запуску програмного продукту. Відомості про мінімальну та рекомендовану конфігурацію системи наведено в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i3-10400/AMD Ryzen 5 3600	Intel Core i3-3210/AMD Athlon II X4 635
Оперативна пам'ять	8 ГБ RAM	4 ГБ RAM
Вільний простір на диску	SSD, 20+ ГБ	10 ГБ
Відеокарта	Nvidia GeForce MX130/AMD Radeon RX Vega 5	Вбудована в процесор, наприклад Intel HD Graphics 4400
Операційна система	Windows 11	Windows 10
Монітор	Роздільна здатність не менше 1920x1080 пікселів або вище	Роздільна здатність не менше 1366x768 пікселів
Мережеве підключення	Стабільне з’єднання 20 Мбіт/с і більше	Інтернет 5 Мбіт/с

Оскільки застосунок працює у браузері, більшість обчислень виконується на клієнтській стороні через JavaScript. Не вимагається встановлення

спеціального ПЗ, окрім сучасного браузера (Google Chrome, Firefox, Edge). Тому користувачу не потрібні високі системні параметри, що також є плюсом.

Після запуску вебзастосунку у вікні браузера відображається головна сторінка, де користувач має можливість авторизуватись або зареєструватись. Для цього необхідно ввести електронну пошту та пароль у відповідні поля. У разі першого входу користувач має натиснути кнопку «Реєстрація» та заповнити коротку форму для створення облікового запису.

Після успішної авторизації відкривається персональна панель користувача з навігаційним меню. Основний функціонал системи доступний через такі модулі: головна, календар, нотатки, тамагочі, селфхелп. Користувач може перемикатися між модулями за допомогою бічного меню навігації. Також користувач може скористатись швидкими діями на головному екрані, переглянути свої плани на сьогодні та відмітити уже виконані, та переглянути події в календарі на конкретний день. Також тут присутній індикатор «ложок» у вигляді панелі. У ньому користувач щодня встановлює максимальну кількість доступних ресурсів (ложок), що відображають поточний енергетичний резерв. У випадку перевищення вже використаного ресурсу при спробі зменшити ліміт система видає попередження. А також доступна коротка картка улюбленця, де вказаний його стан, ім’я та спрайт.

У модулі календаря користувач може створювати події, обираючи дату, час, назву події та кількість ресурсів (ложок), які вона потребує. Події можуть бути одноразовими або повторюваними (щоденно/щотижнево). Події автоматично враховуються у системі ложок для обчислення загального щоденного навантаження.

Модуль нотаток дозволяє створювати, редагувати та видаляти текстові записи. Кожна нотатка може бути позначена тегами, що полегшує її пошук. Реалізовано функцію фільтрації за тегами та пошуку за ключовими словами. Додавання або редагування нотаток здійснюється через модальні вікна, що відкриваються без перезавантаження сторінки.

Модуль тамагочі дозволяє завести тамагочі та дати йому ім’я, одного із трьох на вибір: котик, собачка та папуга. А також, якщо у користувача він вже наявний – то просто одразу відкриває поле із улюбленцем та способами догляду за ним: «Спати», «Помити» та «Погодувати». Після виконання будь-якої дії підіймається певна «потреба» на сталу кількість відсотків заповненості. Якщо всі потреби тамагочі впадуть до нуля, він не помре, а буде просто дуже нещасним. В цілях забезпечення користувачів від психологічної травми поняття «смерті» не було введено для улюбленців. Також існує магазин, де користувач за монети може купувати аксесуари для свого тамагочі. Монети видаються в рандомному значенню раз в день, якщо користувач відвідує сервіс та користується ним, для заохочення до тайм-менеджменту.

Розділ «Селфхелп» містить в собі такі підрозділи: «Поняття РДУГ та його ознаки у людини», «Поради для тайм-менеджменту», «Фокус-сесія» та «Відпочинок для нейровідмінних», де користувач може дізнатись відповідно до заголовка інформацію або скористатись реалізацією таймеру для техніки Помодоро. У «Фокус-сесія» також можна налаштувати час роботи та час відпочинку.

Вся навігація інтуїтивно зрозуміла, що забезпечує легку адаптацію до інтерфейсу навіть для недосвідчених користувачів. Отже, інструкція користувача є складовою частиною забезпечення якісної взаємодії з вебзастосунком та сприяє підвищенню зручності його використання.

4.3 Висновки

Отже, було здійснено тестування програмних модулів системи для антропометричних вимірювань із використанням тривимірного моделювання. У ході перевірки оцінювалась стійкість програмного забезпечення до помилкових ситуацій та коректність обробки вхідних даних різних форматів. Крім того, було підготовлено інструкцію користувача, що описує основні етапи початкової роботи з програмним продуктом.

ВИСНОВКИ

У межах бакалаврської кваліфікаційної роботи було створено програмні модулі, орієнтовані на підтримку тайм-менеджменту для людей з нейровідмінністю. Розробку здійснено у середовищі Visual Studio Code.

У процесі виконання роботи було здійснено аналітичний огляд актуального стану проблеми, а також проведено аналіз існуючих аналогів програмного забезпечення. Виявлені недоліки сторонніх рішень стали підґрунтям для формування власного бачення системи. Отримані результати дозволили чітко сформулювати основні цілі та завдання роботи, що визначили подальший напрям розробки.

Під час аналізу технологій розробки було обґрунтовано вибір мов програмування JavaScript, а саме бібліотеку React, для серверу – платформа Node.js, для бази даних – MySQL, а саме MySQLWorkbench.

У бакалаврській кваліфікаційній роботі було зроблено наступне:

- проаналізовано існуючі аналоги та визначено їхні обмеження щодо потреб нейровідмінних користувачів;

- спроєктовано архітектуру вебсистеми з урахуванням адаптивного інтерфейсу та моделі «ресурсних ложок»;

- реалізовано серверну частину з використанням Node.js для забезпечення гнучкої обробки подій і збереження даних;

- розроблено модуль клієнтської частини, що дозволяють створювати, редагувати та переглядати події з різними типами повторюваності;

- розроблено модуль клієнтської частини, що відповідає за створення заміток;

- розроблено модуль клієнтської частини для взаємодії із інтерактивним тамагочі та системою «монет» для покупки аксесуарів для тваринки;

- створено базу даних для зберігання інформації про користувачів, події та ресурси, з використанням MySQL;

- протестовано роботу системи та оцінено її зручність для користувачів з нейровідмінністю.

Було розроблено схеми загального алгоритму роботи програмного застосунку та описано модулі системи ложок, календаря, нотаток та тамагочі. Також було розроблено модель системи з використанням UML діаграм.

Результати тестування підтвердили повну функціональну справність розробленого програмного забезпечення та його відповідність вимогам, визначеним у технічному завданні. Окрім цього, для зручності користувачів було підготовлено інструкцію з експлуатації системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Рейті Д., Гелловелл Е. Книга РДУГ: перезавантаження. Київ: Yakaboo Publishing, 2022. 224 с.

2. Пітерс Т. Аутизм. Від теоретичного розуміння до педагогічного впливу. Львів: Сварог, 2023. 324 с.

3. Станчишин В. Стіни в моїй голові. Жити з тривожністю і депресією. Київ: Віхола, 2020. 176 с.

4. Глоба А. Р., Бабюк Н. П. Цифровий тайм-менеджмент як інструмент підтримки учнів із РДУГ у процесі навчання. Матеріали X Всеукраїнської науково-методичної конференції «Освіта, наука та виробництво: розвиток та перспективи» (2025), Суми, 24 квітня, 2025 р. – с. 270-271.

5. Глоба А. Р., Бабюк Н. П. Інформаційні технології як інструмент тайм-менеджменту: розробка інтерфейсів для людей з нейровідмінністю. Матеріали III Міжнародної міждисциплінарної науково-практичної конференції «Актуальні питання, проблеми та перспективи розвитку науки та освіти» (2025), Полтава, 24-26 квітня, 2025 р. – с. 65-69.

6. Habitica [Електронний ресурс] – Режим доступу до ресурсу: https://habitica.com/static/home (дата звернення: 21.03.2024).

7. Microsoft To Do [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/uk-ua/microsoft-365/microsoft-to-do-list-app (дата звернення: 21.03.2024).

8. Todoist [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/fsdeqe (дата звернення: 21.03.2024).

9. Trello [Електронний ресурс] – Режим доступу до ресурсу: https://trello.com/en (дата звернення: 21.03.2024).

10. Фрімен Е., Робсон Е. Head First. Програмування на JavaScript. Харків: Фабула, 2022. 672 с.

11. React [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev (дата звернення: 25.03.2024).

12. Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.dev (дата звернення: 25.03.2024).

13. Vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org (дата звернення: 25.03.2024).

14. Node.js. [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk (дата звернення: 25.03.2024).

15. MacIntyre P., Tatroe K. Programming PHP: Creating Dynamic Web Pages. California: O'Reilly Media, 2020. 544 p.

16. .NET [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-gb/dotnet/welcome (дата звернення: 25.03.2024).

17. MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.mysql.com (дата звернення: 25.03.2024).

18. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org (дата звернення: 25.03.2024).

19. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com (дата звернення: 25.03.2024).

20. Вайншенк С. 100 речей, які кожен дизайнер повинен знати про людей. Харків: ArtHuss, 2024. 244 с.

21. Spoon Theory [Електронний ресурс] – Режим доступу до ресурсу: https://neurodivergentinsights.com/the-neurodivergent-spoon-drawer-spoon-theory-for-adhders-and-autists/ (дата звернення: 21.03.2024).

22. Сейден Д., Готельф Д. Книга Lean UX: Створення класних продуктів із командами Agile. Харків: ArtHuss, 2024. 206 с.

23. Круг С. Не змушуйте мене думати. Розсудливий підхід до зручності в користуванні сайтами. Харків: ArtHuss, 2024. 198 с.

24. Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com (дата звернення: 25.03.2024).

25. WebStorm [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/webstorm/# (дата звернення: 25.03.2024).

26. Atom [Електронний ресурс] – Режим доступу до ресурсу: https://atom-editor.cc (дата звернення: 25.03.2024).

27. Тестування програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://university.sigma.software/what-is-software-testing/ (дата звернення: 10.04.2024).

28. Функціональне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/funktsionalne-testuvannya/ (дата звернення: 10.04.2024).

29. Тестування відмовостійкості [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/testuvannya-na-vidmovu-ta-ponovlennya/ (дата звернення: 10.04.2024).

30. Тестування продуктивності [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/testuvannya-produktivnosti/ (дата звернення: 10.04.2024).

31. Тестування сумісності [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/nvdhyu (дата звернення: 10.04.2024).

72

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"25" 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка інклюзивного вебзастосунку для тайм-менеджменту нейровідмінних людей» за спеціальністю
121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____к.т.н., доц. Бабюк Н. П.
"24" 03 2025 р.
Виконав:
_____студент гр.5ПІ-21б Глоба А. Р.
"24" 03 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка інклюзивного вебзастосунку для тайм-менеджменту нейровідмінних людей». Галузь застосування – інформаційні технології у сфері ментального здоров'я та нейрорізноманіття.

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від «20» березня 2025 р. ректора ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою бакалаврської кваліфікаційної роботи є вдосконалення процесу тайм-менеджменту та щоденного планування шляхом розробки адаптивної вебсистеми, що забезпечує нейровідмінним користувачам можливість краще управляти своїм часом з урахуванням їхніх унікальних когнітивних, емоційних і ресурсних потреб.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. Глоба А. Р., Бабюк Н. П. Цифровий тайм-менеджмент як інструмент підтримки учнів із РДУГ у процесі навчання. Матеріали X Всеукраїнської науково-методичної конференції «Освіта, наука та виробництво: розвиток та перспективи» (2025), Суми, 24 квітня, 2025 р. – с. 270-271.

2. Глоба А. Р., Бабюк Н. П. Інформаційні технології як інструмент тайм-менеджменту: розробка інтерфейсів для людей з нейровідмінністю. Матеріали III Міжнародної міждисциплінарної науково-практичної конференції «Актуальні питання, проблеми та перспективи розвитку науки та освіти» (2025), Полтава, 24-26 квітня, 2025 р. – с. 65-69.

5. Технічні вимоги

Вихідні дані до роботи: особисті параметри користувача; інформація про події; дані нотаток та планів; обрана кількість «ложок» на день, як показник енергії; режим перегляду календаря; алгоритм взаємодії інтерактивного тамагочі; фокус-сесії за допомогою таймера; середовище розробки – Visual Studio Code; мова розробки – JavaScript, React; база даних – MySQL; платформа для реалізації серверної логіки – Node.js.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БКР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ п/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз сучасного стану систем для тайм-менеджменту та їх можливості використання для нейровідмінних людей	25.03.25 – 01.04.25
2	Розробка інклюзивного інтерфейсу для нейровідмінних людей	01.04.25 – 08.04.25
3	Розробка алгоритмів вебзастосунку	08.04.25 – 22.04.25
4	Розробка модулів вебзастосунку	22.04.25 – 16.05.25
5	Тестування програмного застосунку	16.05.25 – 23.05.25
6	Оформлення матеріалів до захисту БКР	23.05.25 – 30.05.25

10. Порядок контролю та прийняття

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

76

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка інклюзивного вебзастосунку для тайм-менеджменту нейровідмінних людей

Тип роботи: Бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра ПЗ, ФІТКІ, 5ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 2,5 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ к.т.н., доц. каф. ПЗ Бабюк Н. П.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Глоба А. Р.
(підпис) (прізвище, ініціали)

Додаток В Лістинг програми

Модуль SpoonSystem.css:

```
.message {
    margin-top: 10px;
    font-weight: bold;
}

/* Контейнер у жовтих тонах */
.spoon-container {
    background: #f6d868;
    /* Фон, схожий на приклад */
    padding: 20px;
    border-radius: 15px;
    text-align: center;
    width: 94%;
    max-width: 600px;
    /* Обмеження ширини для кращої читабельності */
    margin: 0 auto;
    /* Вирівнювання по центру */
}

/* Блок із самим слайдером і підписами */
.range-container {
    position: relative;
    width: 80%;
    margin: 0 auto;
}

/* Слайдер (основна доріжка в Chrome) */
input[type="range"] {
    -webkit-appearance: none;
    /* Прибираємо дефолтні стилі */
    width: 100%;
    height: 8px;
    /* Висота доріжки */
    outline: none;
    border-radius: 8px;
    margin: 20px 0 10px 0;
    /* Верхній/нижній відступи */
    background: #d6bf76;
    /* Базовий колір доріжки (fallback) */
}

/* Слайдер: доріжка у Firefox (не показує заповнену частину, якщо не налаштовано ::-moz-range-progress) */
input[type="range"]::-moz-range-track {
    background: transparent;
    height: 8px;
    border-radius: 8px;
}
```

```
/* Заповнена частина у Firefox (ліворуч від повзунка) */
input[type="range"]::-moz-range-progress {
    background-color: #4d2c91;
    height: 8px;
    border-radius: 8px;
}
input[type="range"]::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
    background: url("/public/images/Spoon.png") no-repeat center center;
    /* Іконка ложки */
    background-size: contain;
    width: 32px;
    /* Розмір повзунка */
    height: 32px;
    cursor: pointer;
    margin-top: -12px;
    /* Вирівнювання повзунка по центру доріжки */
}

/* Повзунок (thumb) - Firefox */
input[type="range"]::-moz-range-thumb {
    background: url("/public/images/Spoon.png") no-repeat center center;
    background-size: contain;
    width: 32px;
    height: 32px;
    cursor: pointer;
    border: none;
}

/* Цифри під слайдером */
.tickmarks {
    display: flex;
    justify-content: space-between;
    font-weight: bold;
    color: #6a5c33;
    /* Темно-коричневий колір підписів */
    margin: 0 3px;
}

.tickmarks span {
    flex: 1;
    text-align: center;
}

.spoons-title{
    font-weight: 700;
    font-size: 24px;
    text-align: center;
    color: #776935;
}
.spoons-subtitle{
    font-weight: 500;
```

```
  font-size: 20px;
  text-align: center;
  color: #776935;
}
```

Модуль SpoonSystem.jsx:

```
import { useState, useEffect } from "react";
import axios from "axios";
import "./SpoonSystem.css";

const SpoonSystem = () => {
 const [spoons, setSpoons] = useState(5);
 const [usedToday, setUsedToday] = useState(0);
 const [user, setUser] = useState(null);
 const [message, setMessage] = useState("");
 const [submitted, setSubmitted] = useState(false);
 const [alreadySetToday, setAlreadySetToday] = useState(false);
 const [dbSpoons, setDbSpoons] = useState(null);
 const maxSpoons = 14;

 useEffect(() => {
  const storedUser = localStorage.getItem("user");
  if (storedUser) {
   setUser(JSON.parse(storedUser));
  }
 }, []);

 const formatDateForSQL = (jsDate) => {
  const d = new Date(jsDate);
  d.setMinutes(d.getMinutes() - d.getTimezoneOffset());
  return d.toISOString().split("T")[0];
 };

 useEffect(() => {
 if (!user) return;

 const fetchSpoons = async () => {
  try {
   const res = await axios.get(`http://localhost:5000/api/spoons/${user.id}`);
   const { spoons, last_spoons_update } = res.data;
   setDbSpoons(spoons);

   if (last_spoons_update) {
    const today = new Date().toISOString().split("T")[0];
    const updatedDate = new Date(last_spoons_update).toISOString().split("T")[0];
    setAlreadySetToday(today === updatedDate);
   }
  } catch (error) {
   console.error("Помилка завантаження ложок:", error);
  }
 };
```

```
  const fetchUsedToday = async () => {
    try {
      const today = new Date();
      const todayYMD = formatDateForSQL(today);

      const res = await axios.get(`http://localhost:5000/events?user_id=${user.id}`);
      const total = res.data
        .filter((ev) => {
          const evDate = new Date(ev.event_date);
          return formatDateForSQL(evDate) === todayYMD;
        })
        .reduce((sum, ev) => sum + (parseInt(ev.resource) || 0), 0);

      setUsedToday(total);
    } catch (err) {
      console.error("Помилка завантаження подій:", err);
    }
  };

  fetchSpoons();
  fetchUsedToday();
}, [user]);

  const handleSpoonChange = (event) => {
    setSpoons(Number(event.target.value));
  };

  const handleSubmit = async () => {
    if (!user) return;

    if (spoons < usedToday) {
      alert(`⛔ Ви вже використали ${usedToday} ложок. Неможливо встановити менше.`);
      return;
    }

    try {
      const res = await axios.post("http://localhost:5000/api/spoons", {
        userId: user.id,
        spoons: Number(spoons),
      });
      setSubmitted(true);
      setAlreadySetToday(true);
      setDbSpoons(Number(spoons));
      setMessage("Ложки оновлено ☑");
    } catch (error) {
      alert(error.response?.data?.message || "Помилка при збереженні");
    }
  };

  const fillPercentage = (spoons / maxSpoons) * 100;

  return (
```

```
    <div className="spoon-container">
      {alreadySetToday ? (
        <>
          <h2 className="spoons-title">Мої ложки на сьогодні</h2>
          <p className="spoons-subtitle">
            Встановлено: <strong>{dbSpoons}</strong><br />
            Використано: <strong>{usedToday}</strong> ложок
          </p>
        </>
      ) : (
        <>
          <h2 className="spoons-title">На скільки ложок ви сьогодні себе відчуваєте?</h2>
          <input
            type="range"
            min="0"
            max={maxSpoons}
            step="1"
            value={spoons}
            onChange={handleSpoonChange}
            style={{
              background: `linear-gradient(to right,
                          #776935 0%,
                          #776935 ${fillPercentage}%,
                          #d6bf76 ${fillPercentage}%,
                          #d6bf76 100%)`,
            }}
          />
          <div className="tickmarks">
            {Array.from({ length: maxSpoons + 1 }, (_, i) => (
              <span key={i}>{i}</span>
            ))}
          </div>
          <p className="spoons-subtitle">{spoons} ложок</p>
          <p className="spoons-subtitle">Використано сьогодні: <strong>{usedToday}</strong></p>
          <button onClick={handleSubmit} className="home-button">Зберегти</button>
          {message && <p className="message">{message}</p>}
        </>
      )}

      <p className="spoons-subtitle">Вдалого дня! Не забудьте відпочити ще сьогодні 🫖 </p>
    </div>
  );
};

export default SpoonSystem;
```

Модуль AddNoteModal.css:

```
// AddNoteModal.jsx
import React, { useState, useEffect, useRef } from "react";
import axios from "axios";
import "./styles.css";
```

```
const AddNoteModal = ({ isOpen, onClose, user, onNoteAdded }) => {
  const [newNote, setNewNote] = useState({ title: "", content: "" });
  const modalRef = useRef(null);

  // Закриття модального вікна при кліку поза його межами
  useEffect(() => {
    const handleClickOutside = (e) => {
      if (modalRef.current && !modalRef.current.contains(e.target)) {
        onClose();
      }
    };
    document.addEventListener("mousedown", handleClickOutside);
    return () => document.removeEventListener("mousedown", handleClickOutside);
  }, [onClose]);

  const addNote = async () => {
    if (!newNote.content.trim()) {
      alert("Текст нотатки не може бути порожнім!");
      return;
    }
    try {
      await axios.post(`http://localhost:5000/notes`, {
        title: newNote.title.trim() || "",
        content: newNote.content.trim(),
        user_id: user.id,
      });
      onNoteAdded(); // Оновлюємо список нотаток
      setNewNote({ title: "", content: "" });
      onClose();
    } catch (error) {
      console.error("Помилка при додаванні нотатки:", error);
    }
  };

  if (!isOpen) return null;
  return (
    <div className="modal">
      <div className="modal-content" ref={modalRef}>
        <input
          type="text"
          className="modal-input-header"
          placeholder="Назва..."
          value={newNote.title}
          onChange={(e) => setNewNote({ ...newNote, title: e.target.value })}
        />
        <textarea
          className="modal-input"
          placeholder="Почніть введення..."
          value={newNote.content}
          onChange={(e) => setNewNote({ ...newNote, content: e.target.value })}
        ></textarea>
        <button className="save-btn" onClick={addNote}>
          <svg width="24" height="24" viewBox="0 0 24 24" fill="none">
```

```
          <path
            d="M5 21H19C19.5304 21 20.0391 20.7893 20.4142 20.4142C20.7893 20.0391 21 19.5304 21 19V8L16 3H5C4.46957 3 3.96086 3.21071 3.58579 3.58579C3.21071 3.96086 3 4.46957 3 5V19C3 19.5304 3.21071 20.0391 3.58579 20.4142C3.96086 20.7893 4.46957 21 5 21ZM7 5H11V7H13V5H15V9H7V5ZM7 13H17V19H7V13Z"
            fill="#24418A"
          />
        </svg>
        Зберегти
      </button>
      <button className="save-btn" onClick={onClose}>
        Скасувати
      </button>
    </div>
  </div>
 );
};

export default AddNoteModal;
```

Модуль Notes.jsx:

```
// Notes.jsx
import React, { useState, useEffect, useRef } from "react";
import axios from "axios";
import "./styles.css";
import AddNoteModal from "./AddNoteModal";

const Notes = () => {
 const [allNotes, setAllNotes] = useState([]); // Всі нотатки
 const [notes, setNotes] = useState([]); // Відфільтровані нотатки
 const [tags, setTags] = useState([]);
 const [selectedTags, setSelectedTags] = useState([]);
 const [searchQuery, setSearchQuery] = useState("");
 const [modalNote, setModalNote] = useState({ title: "", content: "", tags: [] });
 const [modalOpen, setModalOpen] = useState(false); // Для редагування нотатки
 const [filterOpen, setFilterOpen] = useState(false);
 const [editingTag, setEditingTag] = useState(null);
 const [newTagName, setNewTagName] = useState("");
 const [newTagColor, setNewTagColor] = useState("#FFD700"); // Жовтий за замовчуванням
 const [user, setUser] = useState(null);
 const [isAddModalOpen, setIsAddModalOpen] = useState(false); // Для додавання нової нотатки

 const editModalRef = useRef(null);

 useEffect(() => {
  const storedUser = localStorage.getItem("user");
  if (storedUser) {
   setUser(JSON.parse(storedUser));
  }
 }, []);

 useEffect(() => {
  if (user?.id) {
```

```
    fetchNotes();
    fetchTags();
  }
}, [user]);

const fetchNotes = async () => {
  const res = await axios.get(`http://localhost:5000/notes?user_id=${user.id}`);
  const notesData = res.data;

  const notesWithTags = await Promise.all(
    notesData.map(async (note) => {
      const tagsRes = await axios.get(`http://localhost:5000/note_tags?note_id=${note.id}`);
      return { ...note, tags: tagsRes.data };
    })
  );

  setAllNotes(notesWithTags);
  setNotes(notesWithTags);
};

const fetchTags = async () => {
  const res = await axios.get(`http://localhost:5000/tags?user_id=${user.id}`);
  setTags(res.data);
};

const deleteNote = async (id) => {
  await axios.delete(`http://localhost:5000/notes/${id}`);
  fetchNotes();
};

const updateNote = async () => {
  if (!modalNote) return;
  await axios.put(`http://localhost:5000/notes/${modalNote.id}`, modalNote);
  fetchNotes();
  closeModal();
};

const toggleTagFilter = (tagId) => {
  setSelectedTags((prev) =>
    prev.includes(tagId) ? prev.filter((t) => t !== tagId) : [...prev, tagId]
  );
};

const filterNotes = () => {
  return allNotes.filter((note) => {
    const matchesSearch =
      note.title.toLowerCase().includes(searchQuery.toLowerCase()) ||
      note.content.toLowerCase().includes(searchQuery.toLowerCase());

    if (selectedTags.length === 0) return matchesSearch;

    const matchesTags = selectedTags.some((tagId) =>
      note.tags.some((tag) => tag.id === tagId)
```

```
    );

    return matchesSearch && matchesTags;
  });
};

const openModal = async (note) => {
  try {
    const tagsRes = await axios.get(`http://localhost:5000/note_tags?note_id=${note.id}`);
    const noteWithTags = { ...note, tags: tagsRes.data };
    setModalNote(noteWithTags);
    setModalOpen(true);
  } catch (error) {
    console.error("Помилка при завантаженні міток:", error);
  }
};

useEffect(() => {
  console.log("Modal state updated:", modalOpen, modalNote);
}, [modalOpen, modalNote]);

// Функції для роботи з мітками у нотатках
const addTagToNote = async (tagName) => {
  if (!modalNote) return;
  try {
    const res = await axios.post("http://localhost:5000/tags", {
      user_id: user.id,
      name: tagName,
      color: newTagColor,
    });
    const tagId = res.data.id;
    await axios.post("http://localhost:5000/note_tags", {
      note_id: modalNote.id,
      tag_id: tagId,
    });
    const newTag = { id: tagId, name: tagName, color: newTagColor };
    setModalNote((prev) => ({
      ...prev,
      tags: [...prev.tags, newTag],
    }));
    fetchNotes();
  } catch (error) {
    console.error("Помилка при додаванні мітки:", error);
  }
};

const updateTagName = async (tag) => {
  if (!newTagName.trim()) return;
  try {
    const res = await axios.put("http://localhost:5000/tags/update", {
      id: tag.id,
      newName: newTagName,
      newColor: tag.color,
```

```
    });
    console.log(res.data.message);
    setModalNote((prev) => ({
      ...prev,
      tags: prev.tags.map((t) => (t.id === tag.id ? { ...t, name: newTagName } : t)),
    }));
    setTags((prev) =>
      prev.map((t) => (t.id === tag.id ? { ...t, name: newTagName } : t))
    );
    setEditingTag(null);
  } catch (error) {
    console.error("Помилка оновлення мітки:", error);
  }
};

const updateTagColor = async (tag, color) => {
  try {
    const res = await axios.put("http://localhost:5000/tags/update", {
      id: tag.id,
      newName: tag.name,
      newColor: color,
    });
    console.log(res.data.message);
    setModalNote((prev) => ({
      ...prev,
      tags: prev.tags.map((t) => (t.id === tag.id ? { ...t, color } : t)),
    }));
    setTags((prev) =>
      prev.map((t) => (t.id === tag.id ? { ...t, color } : t))
    );
  } catch (error) {
    console.error("Помилка оновлення кольору мітки:", error);
  }
};

const removeTagFromNote = async (tag) => {
  if (!modalNote) return;
  try {
    await axios.delete("http://localhost:5000/note_tags", {
      data: { note_id: modalNote.id, tag_id: tag.id },
    });
    setModalNote((prev) => ({
      ...prev,
      tags: prev.tags.filter((t) => t.id !== tag.id),
    }));
    fetchNotes();
    fetchTags();
  } catch (error) {
    console.error("Помилка при видаленні мітки:", error);
  }
};

const addExistingTagToNote = async (tagId) => {
```

```
    if (!modalNote) return;
    try {
      await axios.post("http://localhost:5000/note_tags", {
        note_id: modalNote.id,
        tag_id: tagId,
      });
      const selectedTag = tags.find((tag) => tag.id === tagId);
      if (selectedTag) {
        setModalNote((prev) => ({
          ...prev,
          tags: [...prev.tags, selectedTag],
        }));
      }
      fetchNotes();
    } catch (error) {
      console.error("Помилка при додаванні мітки:", error);
    }
  };

  const closeModal = () => {
    setModalOpen(false);
    setModalNote({ title: "", content: "", tags: [] });
  };

  const filteredNotes = filterNotes();

  const formatUpdateDate = (updated_at) => {
  if (!updated_at) return "";

  const updated = new Date(updated_at);
  const now = new Date();

  const updatedYMD = updated.toDateString();
  const yesterdayYMD = new Date(new Date().setDate(now.getDate() - 1)).toDateString();

  if (updatedYMD === yesterdayYMD) {
    return `Вчора, ${updated.toLocaleTimeString("uk-UA", { hour: "2-digit", minute: "2-digit" })}`;
  }

  return `${updated.toLocaleDateString("uk-UA")}, ${updated.toLocaleTimeString("uk-UA", { hour: "2-digit", minute: "2-digit" })}`;
};

  return (
    <div className="notes-container">
      <header className="notes-header">
        <div className="filter-container">
          <button onClick={() => setFilterOpen(!filterOpen)}>
            <svg width="22" height="26" viewBox="0 0 22 26" fill="none" xmlns="http://www.w3.org/2000/svg" style={{ marginLeft: "20px" }}>
              <path
                d="M13.6246 12.9982V24.3789C13.6771 24.8122 13.5459 25.2743 13.2441 25.5776C13.1227 25.7115 12.9784 25.8177 12.8197 25.8902C12.6609 25.9627 12.4908 26 12.3189 26C12.147 26 11.9768 25.9627 11.8181 25.8902C11.6593 25.8177 11.5151 25.7115 11.3937 25.5776L8.75594 22.6747C8.61289 22.5207 8.50412 22.3323 8.4381 22.1244C8.37209 21.9164 8.35062 21.6945 8.37536 21.476V12.9982H8.33599L0.77704 2.33968C0.563931 2.0386 0.467771 1.65693 0.509574 1.27806C0.551377 0.899192 0.727743 0.553936 1.00013
```

```
0.317735C1.24948 0.11554 1.52506 0 1.81377 0H20.1862C20.4749 0 20.7505 0.11554 20.9999 0.317735C21.2723 0.553936 21.4486 0.899192
21.4904 1.27806C21.5322 1.65693 21.4361 2.0386 21.223 2.33968L13.664 12.9982H13.6246Z"
            fill="#24418A"
           />
          </svg>
          Фільтрувати за мітками
         </button>
         {filterOpen && (
          <div className="filter-dropdown">
           {tags.map((tag) => (
            <div key={tag.id} className="filter-tag" onClick={() => toggleTagFilter(tag.id)}>
             <input type="checkbox" checked={selectedTags.includes(tag.id)} readOnly />
             <span>{tag.name}</span>
            </div>
           ))}
          </div>
         )}
        </div>

        <input
         type="text"
         placeholder="Пошук"
         value={searchQuery}
         className="search-bar"
         onChange={(e) => setSearchQuery(e.target.value)}
        />

        <button className="add-note-modal" onClick={() => setIsAddModalOpen(true)}>
         <svg width="45" height="40" viewBox="0 0 45 40" fill="none" xmlns="http://www.w3.org/2000/svg">
          <rect width="45" height="40" rx="20" fill="#F6D868" />
          <path
           d="M33 21.5H24V30.5H21V21.5H12V18.5H21V9.5H24V18.5H33V21.5Z"
           fill="#776935"
          />
         </svg>
        </button>
       </header>

       <div className="notes-grid">
        {filteredNotes.length === 0 ? (
         <div className="empty-notes-message">
          Тут ще немає ваших нотаток.
          <br />
          Натисніть кнопку + на верхній панелі, щоб додати першу нотатку.
         </div>
        ) : ( filteredNotes.map((note) => {
        const updatedAt = new Date(note.updated_at);
        const now = new Date();
        const isYesterday = updatedAt.toDateString() === new Date(now.setDate(now.getDate() - 1)).toDateString();

        const formattedUpdate = isYesterday
         ? `Вчора, ${updatedAt.toLocaleTimeString('uk-UA', { hour: '2-digit', minute: '2-digit' })}`
         : `${updatedAt.toLocaleDateString('uk-UA')}, ${updatedAt.toLocaleTimeString('uk-UA', { hour: '2-digit', minute: '2-digit' })}`;
```

```
    return (
      <div key={note.id} className="note" >
        <div className="tags-container" onClick={() => openModal(note)}>
          {(note.tags || []).map((tag) => (
            <span key={tag.id} className="tag" style={{ backgroundColor: tag.color }}>
              {tag.name}
            </span>
          ))}
        </div>

        <div className="note-controls">
          <h3 className="note-title" onClick={() => openModal(note)}>
            {note.title}
          </h3>
          <div className="note-actions">
            <button className="note-icon" onClick={() => openModal(note)}>🖍 </button>
            <button className="note-icon" onClick={() => deleteNote(note.id)}>🗑 </button>
          </div>
        </div>

        <p className="note-content" onClick={() => openModal(note)}>
          {note.content.length > 100 ? note.content.substring(0, 100) + "..." : note.content}
        </p>

        <p className="note-updated" onClick={() => openModal(note)}>{formattedUpdate}</p>
      </div>
    );
  })
)}
</div>

  {/* Модальне вікно для редагування нотатки */}
  {modalOpen && modalNote && (
    <div className="modal">
      <div className="modal-content" ref={editModalRef}>
        <input
          type="text"
          className="modal-input modal-title"
          value={modalNote.title}
          onChange={(e) => setModalNote({ ...modalNote, title: e.target.value })}
        />
        <p className="note-updated-modal">
          Оновлено: {formatUpdateDate(modalNote.updated_at)}
        </p>
        <textarea
          className="modal-input"
          value={modalNote.content}
          onChange={(e) => setModalNote({ ...modalNote, content: e.target.value })}
        ></textarea>
        <div className="tags-container">
          {modalNote.tags.length > 0
```

```
? modalNote.tags.map((tag, index) => (
  <span
   key={tag.id || index}
   className="tag"
   style={{ backgroundColor: tag.color, color: "#fff" }}
  >
   {editingTag?.id === tag.id ? (
    <>
     <input
      className="tag-input"
      type="text"
      style={{ backgroundColor: tag.color, color: "#fff" }}
      value={newTagName}
      onChange={(e) => setNewTagName(e.target.value)}
      onBlur={() => updateTagName(tag)}
      onKeyDown={(e) => e.key === "Enter" && updateTagName(tag)}
     />
     <div className="tag-colors">
      {["#FFD700", "#87CEEB", "#FF6347", "#90EE90"].map((color) => (
       <span
        key={color}
        className="color-dot"
        style={{
         backgroundColor: color,
         border: color === tag.color ? "3px solid black" : "none",
        }}
        onClick={() => updateTagColor(tag, color)}
       ></span>
      ))}
     </div>
    </>
   ) : (
    <span
     onClick={() => {
      setEditingTag(tag);
      setNewTagName(tag.name);
     }}
    >
     {tag.name}
    </span>
   )}
                <svg onClick={() => removeTagFromNote(tag)} width="17" height="17" viewBox="0 0 17 17" fill="none"
xmlns="http://www.w3.org/2000/svg">
    <path
                d="M5.16663 14.5C4.79996 14.5 4.48618 14.3696 4.22529 14.1087C3.9644 13.8478 3.83374 13.5338 3.83329
13.1667V4.5H3.16663V3.16667H6.49996V2.5H10.5V3.16667H13.8333V4.5H13.1666V13.1667C13.1666 13.5333 13.0362 13.8473 12.7753
14.1087C12.5144 14.37 12.2004 14.5004 11.8333 14.5H5.16663ZM6.49996 11.8333H7.83329V5.83333H6.49996V11.8333ZM9.16663
11.8333H10.5V5.83333H9.16663V11.8333Z"
     fill="#554560"
    />
   </svg>
  </span>
 ))
```

```
    : null}
  </div>
  <select
   className="tag-select"
   onChange={(e) => addExistingTagToNote(Number(e.target.value))}
  >
   <option value="">Оберіть мітку із існуючих</option>
   {tags
    .filter((tag) => !modalNote.tags.some((t) => t.id === tag.id))
    .map((tag) => (
     <option key={tag.id} value={tag.id}>
      {tag.name}
     </option>
    ))}
  </select>
  <div className="tag-colors">
   <input
    type="text"
    className="new-tag-input"
    placeholder="Напишіть назву мітки..."
    onKeyDown={(e) => {
     if (e.key === "Enter" && e.target.value.trim()) {
      addTagToNote(e.target.value.trim());
      e.target.value = "";
     }
    }}
   />
   <div className="tag-colors-container">
    <p>Оберіть колір мітки</p>
    {["#FFD700", "#87CEEB", "#FF6347", "#90EE90"].map((color) => (
     <span
      key={color}
      className="color-dot"
      style={{
       backgroundColor: color,
       border: color === newTagColor ? "3px solid black" : "none",
      }}
      onClick={() => setNewTagColor(color)}
     ></span>
    ))}
   </div>
  </div>
  <button onClick={updateNote} className="save-btn">
   <svg width="24" height="24" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">
    <path
     d="M5 21H19C19.5304 21 20.0391 20.7893 20.4142 20.4142C20.7893 20.0391 21 19.5304 21 19V8L16 3H5C4.46957 3 3.96086 3.21071 3.58579 3.58579C3.21071 3.96086 3 4.46957 3 5V19C3 19.5304 3.21071 20.0391 3.58579 20.4142C3.96086 20.7893 4.46957 21 5 21ZM7 5H11V7H13V5H15V9H7V5ZM7 13H17V19H7V13Z"
     fill="#24418A"
    />
   </svg>
   Зберегти
  </button>
```

```
          <button onClick={closeModal} className="save-btn">
            Закрити
          </button>
        </div>
      </div>
    )}

    {/* Виклик компоненту модального вікна для додавання нотатки */}
    <AddNoteModal
      isOpen={isAddModalOpen}
      onClose={() => setIsAddModalOpen(false)}
      user={user}
      onNoteAdded={fetchNotes}
    />
  </div>
 );
};

export default Notes;
```

Модуль Tamagochi.css:

```
.tama-intro-container {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

.tama-intro-header {
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;
  padding: 16px;
  width: 85%;
}
.tama-intro-coin{
  width: 25%;
}
.tama-intro-name{
  width: 70%;
  text-align: center;
}

.tama-intro-box {
  background: url("../../../public/images/tamagotchi/newPage.png") no-repeat center center / cover;
  /* background-color: rgba(255, 255, 255, 0.3); */
  /* Накладення кольору */
  /* box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1); */
  padding: 30px;
  width: 90%;
  text-align: center;
```

```
    font-size: 18px;
    font-weight: bold;
}
.tama-box{
    background: url("../../../public/images/tamagotchi/tama-block.png") no-repeat center center / cover;
    margin-top: 26px;
    margin-bottom: 26px;
    padding: 30px;
    width: 90%;
    text-align: center;
    height: 440px;
    display: flex;
    justify-content: center;
}
.tama-intro-box ul {
    list-style: none;
    padding: 0;
    margin: 10px 0;
}

.tama-intro-box li {
    font-weight: bold;
    font-size: 20px;
    margin-bottom: 6px;
}

.tama-start-button {
    background-color: #f6d868;
    color: #4a402e;
    font-size: 18px;
    padding: 14px 30px;
    border: none;
    border-radius: 16px;
    margin-top: 25px;
    cursor: pointer;
    font-weight: bold;
    box-shadow: 0 3px 8px rgba(0, 0, 0, 0.15);
    transition: transform 0.2s ease;
    width: 85%;
}

.tama-start-button:hover {
    transform: scale(1.05);
}

.tama-selection-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    padding: 30px 20px;
}

.tama-name-input {
```

```
    font-size: 18px;
    padding: 10px 16px;
    border-radius: 12px;
    border: 2px solid #ccc;
    width: 100%;
    text-align: center;
    background-color: #eef0f4;
}

.carousel-btn {
    background: none;
    border: none;
    font-size: 30px;
    cursor: pointer;
    color: #444;
}

.tama-image {
    max-height: 220px;
    transition: transform 0.2s ease;
}

.tama-image:hover {
    transform: scale(1.05);
}
.pet-main-screen {
    display: flex;
    flex-direction: column;
    align-items: center;
}

.pet-topbar {
    display: flex;
    width: 100%;
    align-items: center;
    justify-content: center;
    gap: 32px;
}

.coin-display {
    background-color: #fff9e0;
    border-radius: 12px;
    padding: 10px 14px;
    font-weight: bold;
    box-shadow: 0 1px 4px rgba(0, 0, 0, 0.1);
}

.pet-name-bar {
    flex-grow: 1;
    text-align: center;
}

.pet-name-input {
```

```
    font-size: 18px;
    padding: 8px 12px;
    border-radius: 12px;
    border: 2px solid #ccc;
    background-color: #eaefff;
    max-width: 250px;
    text-align: center;
}

.shop-button {
    display: flex;
    align-items: center;
    background-color: #fefcf5;
    padding: 8px 14px;
    border-radius: 30px;
    font-weight: 600;
    font-size: 16px;
    color: #6c5624;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.06);
    gap: 8px;
    width: 140px;
    justify-content: center;
    border: none;
    cursor: pointer;
}

.pet-image-area {
    margin: 120px 0px;
    padding-left: 70px;
}

.main-pet-image {
    max-height: 250px;
}

.pet-actions {
    display: flex;
    gap: 30px;
    width: 100%;
    align-content: center;
    justify-content: center;
}

.pet-action-btn {
    font-size: 16px;
    border: none;
    border-radius: 12px;
    padding: 12px 20px;
    cursor: pointer;
    font-weight: bold;
    color: #444;
    display: flex;
    align-items: center;
```

```
    gap:20px;
    width: 25%;
}

.green {
    background-color: #c8efcb;
}

.blue {
    background-color: #d1e4ff;
}

.yellow {
    background-color: #ffec9f;
}

.carousel-top-bar {
    display: flex;
    justify-content: flex-start;
    width: 100%;
    padding: 10px;
}

.back-button {
    background: #FFFAED;
    border: none;
    width: 128px;
    height: 40px;
    border-radius: 30px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
    font-size: 24px;
    font-weight: bold;
    color: #555;
    cursor: pointer;
}

.back-button:hover {
    background: #f5f5f5;
}

.tama-carousel {
    display: flex;
    align-items: center;
    justify-content: center;
    gap: 20px;
    margin: 20px 0;
}
```

Модуль TamagochiPage.jsx:

```
import React, { useEffect, useState } from "react";
import PetIntroScreen from "./IntroScreen";
import PetSelectionScreen from "./PetSelectionScreen";
```

```
import MainPetScreen from "./MainPetScreen";
import axios from "axios";
import AccessoryShopScreen from "./AccessoryShopScreen";

const TamagotchiPage = () => {
  const [user, setUser] = useState(null);
  const [pet, setPet] = useState(null);
  const [step, setStep] = useState("loading"); // loading | intro | select | main | shop

  useEffect(() => {
    const storedUser = localStorage.getItem("user");
    if (storedUser) {
      const parsedUser = JSON.parse(storedUser);
      setUser(parsedUser);
      loadPet(parsedUser.id);
    }
  }, []);

  const loadPet = async (userId) => {
  try {
    const res = await axios.get(`http://localhost:5000/api/pets/${userId}`);
    if (res.data && res.data.name) {
      setPet(res.data);
      setStep("main"); // показує головний екран
    } else {
      setStep("intro"); // показує вступ
    }
  } catch (error) {
    console.log("🐾 Помилка завантаження улюбленця:", error);
    setStep("intro"); // якщо помилка — все одно показує вступ
  }
};

  const handleStartSelect = () => {
    setStep("select");
  };

const handlePetCreated = async (type, name) => {
  try {
    console.log("🐾 Надсилаємо:", { user_id: user.id, name, type });
    const res = await axios.post("http://localhost:5000/api/pets", {
      user_id: user.id,
      name,
      type,
    });
    setPet({ ...res.data, type, name });
    setStep("main");
  } catch (error) {
    console.error("❌ Помилка створення улюбленця:", error);
  }
};
```

```
  const handleEditName = (newName) => {
    setPet((prev) => ({ ...prev, name: newName }));
    axios.put(`http://localhost:5000/api/pets/${user.id}`, {
      name: newName,
    });
  };

const handleAction = async (actionType) => {
  const randomValue = Math.floor(Math.random() * (25 - 10 + 1)) + 10;

  // Поточний рівень з pet
  const currentLevel =
    actionType === "feed"
      ? pet.hunger_level
      : actionType === "clean"
      ? pet.clean_level
      : pet.sleep_level;

  if (currentLevel >= 100) {
    alert("🔔 Цей параметр вже на максимумі!");
    return;
  }

  try {
    await axios.put(`http://localhost:5000/api/pets/${user.id}/action`, {
      action: actionType,
      value: randomValue,
    });

    alert(
      `🎉 ${actionType === "feed" ? "Голод" : actionType === "clean" ? "Чистота" : "Сон"} відновлено на ${randomValue}`
    );

    loadPet(user.id); // оновлюємо
  } catch (err) {
    console.error("❌ Помилка виконання дії:", err);
  }
};
  if (step === "loading") return <p>Завантаження...</p>;

  return (
    <>
      {step === "intro" && <PetIntroScreen onContinue={handleStartSelect} />}
      {step === "select" && <PetSelectionScreen onConfirm={handlePetCreated} onClose={() => {
          setStep("intro");
        }} />}
      {step === "main" && pet && (
        <MainPetScreen
          pet={pet}
          onFeed={() => handleAction("feed")}
          onClean={() => handleAction("clean")}
          onSleep={() => handleAction("sleep")}
```

```
          onEditName={handleEditName}
          onShop={() => setStep("shop")}
          userId={user.id}
        />
      )}
      {step === "shop" && (
        <AccessoryShopScreen
          userId={user.id}
          onClose={() => {
            loadPet(user.id);
            setStep("main");
          }}
        />
      )}
    </>
  );
};

export default TamagotchiPage;
```

Модуль Calendar.css:

```
/* Головний контейнер календаря */
.calendar-root {
    max-width: 900px;
    margin: 0 auto;
    background-color: #fefdf8;
    border-radius: 8px;
    padding: 16px;
}

/* Кнопки перемикання (день, тиждень, місяць) */
.calendar-view-switch {
    display: flex;
    gap: 8px;
    margin-bottom: 10px;
}

.calendar-view-switch button {
    padding: 6px 12px;
    background-color: #ddd;
    border: none;
    border-radius: 6px;
    cursor: pointer;
}

.calendar-view-switch button.active {
    background-color: #1976d2;
    color: #fff;
}
```

```
/* Шапка з кнопками навігації */
.calendar-header {
    display: flex;
    align-items: center;
    justify-content: space-between;
    margin-bottom: 10px;
}

.calendar-header h2 {
    margin: 0;
    font-size: 1.2rem;
}

/* Кнопки "Додати подію" */
.calendar-add-event {
    display: flex;
    margin-bottom: 10px;
}

.calendar-add-event button {
    background-color: #121212;
    color: #FAF4E4;
    font-weight: 500;
    font-size: 18px;
    border: none;
    border-radius: 12px;
    padding: 10px;
    width: 100%;
    font-size: 14px;
    cursor: pointer;
    display: flex;
    justify-content: center;
    align-items: center;
    gap:12px;
}

.calendar-add-event button:hover {
    background-color: rgb(39, 49, 78);
}

/* ========== МІСЯЧНИЙ ВИГЛЯД ========== */
.calendar-month-view {
    width: 100%;
}
.week-day-header {
    text-align: center;
    font-weight: bold;
    padding: 6px;
    background-color: #f0f0f0;
    border-radius: 4px;
}
```

```
/* Номер дня */
.day-number {
    font-weight: bold;
    margin-bottom: 4px;
}

/* Події (коротка позначка в клітинці) */
.calendar-event-marker {
    background-color: #1976d2;
    color: #fff;
    border-radius: 4px;
    font-size: 0.75rem;
    padding: 2px 4px;
    margin-bottom: 2px;
    display: inline-block;
    cursor: pointer;
}

/* ========== ТИЖНЕВИЙ ВИГЛЯД ========== */
.calendar-week-view {
    width: 100%;
}

.week-days-container {
    display: grid;
    grid-template-columns: repeat(7, 1fr);
    gap: 4px;
}

.week-day-cell {
    min-height: 100px;
    background-color: #fff;
    border: 1px solid #ececec;
    border-radius: 4px;
    padding: 4px;
    position: relative;
}

.week-day-cell.today {
    background-color: #e0ecff;
}

.week-day-label {
    font-weight: bold;
    margin-bottom: 4px;
}

/* ========== ДЕННИЙ ВИГЛЯД ========== */
.calendar-day-view {
    background-color: #fff;
    padding: 10px;
    border-radius: 4px;
    min-height: 150px;
```

```
}

.calendar-day-view .calendar-event-marker {
    display: block;
    margin: 4px 0;
    font-size: 0.9rem;
}

/* ============ ФОРМА ДОДАВАННЯ ПОДІЇ ============ */
.add-event-container {
    max-width: 600px;
    margin: 0 auto;
    background-color: #fff;
    border-radius: 8px;
    padding: 20px;
}

.add-event-container h2 {
    margin-bottom: 16px;
}

.add-event-form label {
    display: block;
    margin-bottom: 10px;
    font-size: 0.9rem;
}

.add-event-form input,
.add-event-form select {
    display: block;
    width: 100%;
    max-width: 300px;
    margin-top: 4px;
    padding: 6px;
    border: 1px solid #ccc;
    border-radius: 4px;
}

.form-buttons {
    margin-top: 15px;
    display: flex;
    gap: 10px;
}

.form-buttons button {
    border: none;
    padding: 8px 14px;
    border-radius: 6px;
    cursor: pointer;
    background-color: #1976d2;
    color: #fff;
}
```

```
.calendar-page {
   display: flex;
   flex-direction: column;
   gap: 20px;
   padding: 16px;
}

.calendar-main {
   display: flex;
   gap: 32px;
   align-items: flex-start;
}

.calendar-root {
   flex: 2;
   box-shadow: 0 4px 10px 0 rgba(0, 0, 0, 0.25);
   background: #fffaed;
   border-radius: 30px;
   padding: 16px 23px;
   width: 95%;
}

.calendar-sidebar {
   flex: 1;
   display: flex;
   flex-direction: column;
   gap: 24px;
}

.calendar-layout {
   display: flex;
   gap: 48px;
   align-items: flex-start;
}

.calendar-left {
   flex: 2;
   display: flex;
   flex-direction: column;
}

.calendar-below {
   margin-top: 16px;
}
.compact-event-calendar{
   box-shadow: 0 4px 10px 0 rgba(0, 0, 0, 0.25);
   background: #fffaed;
   border-radius: 30px;
   padding: 16px 24px;
}
```

Модуль Calendar.jsx:

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import DayView from "../components/calendar/DayView";
import WeekView from "../components/calendar/WeekView";
import MonthView from "../components/calendar/MonthView";
import EventForm from "../components/Events/EventForm";
import "./Calendar.css";
import PlanList from "../components/Plan/PlanList";
import CompactEventList from "../components/Events/CompactEventList";
import CalendarHeader from "./CalendarHeader";

const CalendarWithEvents = () => {
  const [user, setUser] = useState(null);
  const [view, setView] = useState("month");
  const [selectedDate, setSelectedDate] = useState(new Date());
  const [allEvents, setAllEvents] = useState([]);
  const [events, setEvents] = useState([]);
  const [showForm, setShowForm] = useState(false);
  const [editingEvent, setEditingEvent] = useState(null);

  useEffect(() => {
    const storedUser = localStorage.getItem("user");
    if (storedUser) {
      setUser(JSON.parse(storedUser));
    }
  }, []);

  useEffect(() => {
    if (user?.id) {
      axios
        .get(`http://localhost:5000/events?user_id=${user.id}`)
        .then((res) => setAllEvents(res.data || []))
        .catch((err) => console.error("Помилка отримання подій:", err));
    }
  }, [user]);

  useEffect(() => {
    setEvents(allEvents);
  }, [allEvents, selectedDate, view]);

  const changeView = (newView) => setView(newView);

  const handleNavigate = (direction) => {
    const newDate = new Date(selectedDate);
    if (view === "month") newDate.setMonth(newDate.getMonth() + direction);
    else if (view === "week") newDate.setDate(newDate.getDate() + direction * 7);
    else newDate.setDate(newDate.getDate() + direction);
    setSelectedDate(newDate);
  };

  const handleAddEvent = () => {
    setEditingEvent(null);
    setShowForm(true);
```

```
};

const handleEditEvent = (event) => {
  setEditingEvent(event);
  setShowForm(true);
};

const handleSave = () => {
  setShowForm(false);
  axios
    .get(`http://localhost:5000/events?user_id=${user.id}`)
    .then((res) => setAllEvents(res.data || []));
};

const handleCancel = () => {
  setShowForm(false);
  setEditingEvent(null);
};

const formatDateTitle = (date) =>
  date.toLocaleDateString("uk-UA", {
    day: "numeric",
    month: "long",
    year: "numeric",
  });

return (
  <div className="calendar-page">
<CalendarHeader
  selectedDate={selectedDate}
  onNavigate={handleNavigate}
  view={view}
  onChangeView={changeView}
/>

<div className="calendar-layout">
  <div className="calendar-left">
<div className="calendar-root">
  {/* {view === "day" && (
    <div className="calendar-date-title">
      <h2>{formatDateTitle(selectedDate)}</h2>
    </div>
  )} */}

  {showForm ? (
    <EventForm
      date={selectedDate}
      event={editingEvent}
      onSave={handleSave}
      onCancel={handleCancel}
    />
  ) : (
    <>
```

```
      {view === "day" && <DayView date={selectedDate} events={events} onEdit={handleEditEvent} />}
      {view === "week" && <WeekView date={selectedDate} events={events} onEdit={handleEditEvent} />}
      {view === "month" && <MonthView date={selectedDate} events={events} onEdit={handleEditEvent} />}
    </>
  )}
 </div>

 <div className="calendar-below">
  <div className="calendar-add-event">
   <button onClick={handleAddEvent}>Додати подію
    <svg width="21" height="20" viewBox="0 0 21 20" fill="none" xmlns="http://www.w3.org/2000/svg">
     <path d="M8.92105 18.4211C8.92105 18.8398 9.08741 19.2414 9.38352 19.5375C9.67963 19.8336 10.0812 20 10.5 20C10.9188 20 11.3204 19.8336 11.6165 19.5375C11.9126 19.2414 12.0789 18.8398 12.0789 18.4211V11.5789H18.9211C19.3398 11.5789 19.7414 11.4126 20.0375 11.1165C20.3336 10.8204 20.5 10.4188 20.5 10C20.5 9.58124 20.3336 9.17963 20.0375 8.88352C19.7414 8.58741 19.3398 8.42105 18.9211 8.42105H12.0789V1.57895C12.0789 1.16018 11.9126 0.758573 11.6165 0.462463C11.3204 0.166353 10.9188 0 10.5 0C10.0812 0 9.67963 0.166353 9.38352 0.462463C9.08741 0.758573 8.92105 1.16018 8.92105 1.57895V8.42105H2.07895C1.66018 8.42105 1.25857 8.58741 0.962463 8.88352C0.666353 9.17963 0.5 9.58124 0.5 10C0.5 10.4188 0.666353 10.8204 0.962463 11.1165C1.25857 11.4126 1.66018 11.5789 2.07895 11.5789H8.92105V18.4211Z" fill="#FAF4E4" />
    </svg>
   </button>
  </div>

  <div className="compact-event-calendar">
   <CompactEventList title="Події на сьогодні" />
  </div>
 </div>
</div>

  <div className="calendar-sidebar">
   <PlanList />
  </div>
 </div>
</div>

 );
};

export default CalendarWithEvents;
```

Додаток Г – Ілюстративна частина

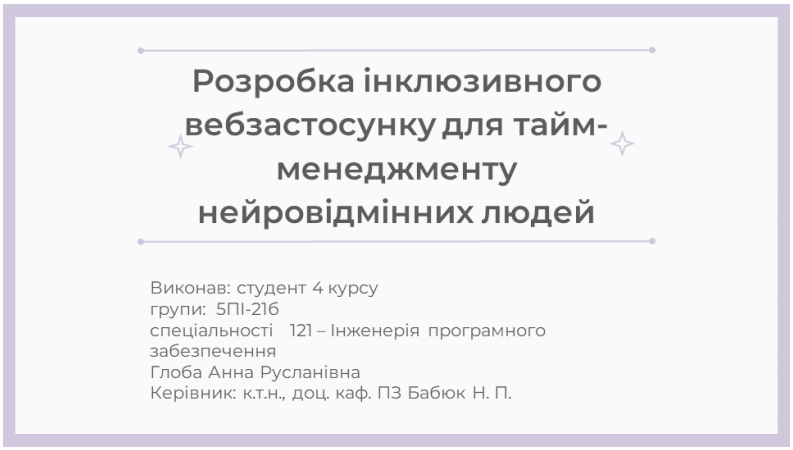

Рисунок Г.1 – Титульний аркуш

Рисунок Г.2 – Актуальність теми

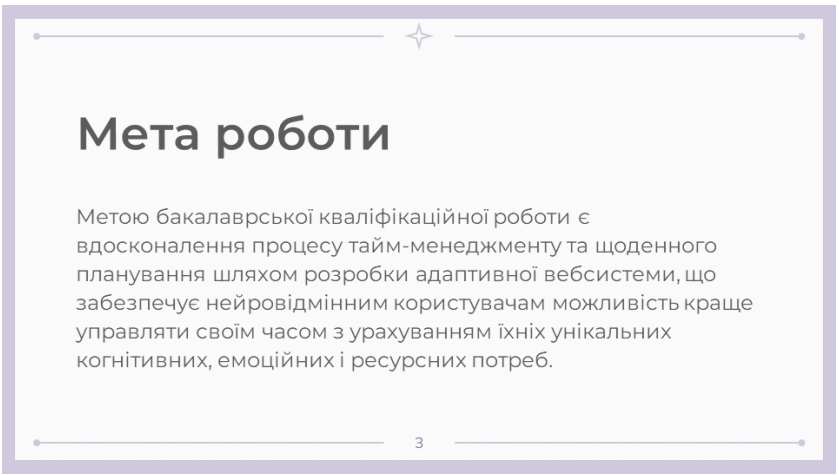

Рисунок Г.3 – Мета роботи

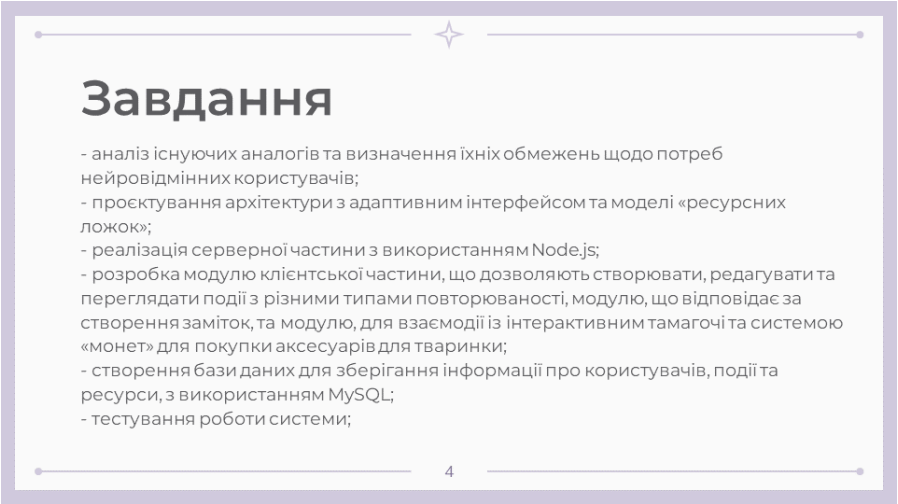

Рисунок Г.4 – Завдання

Рисунок Г.5 – Об’єкт і предмет дослідження

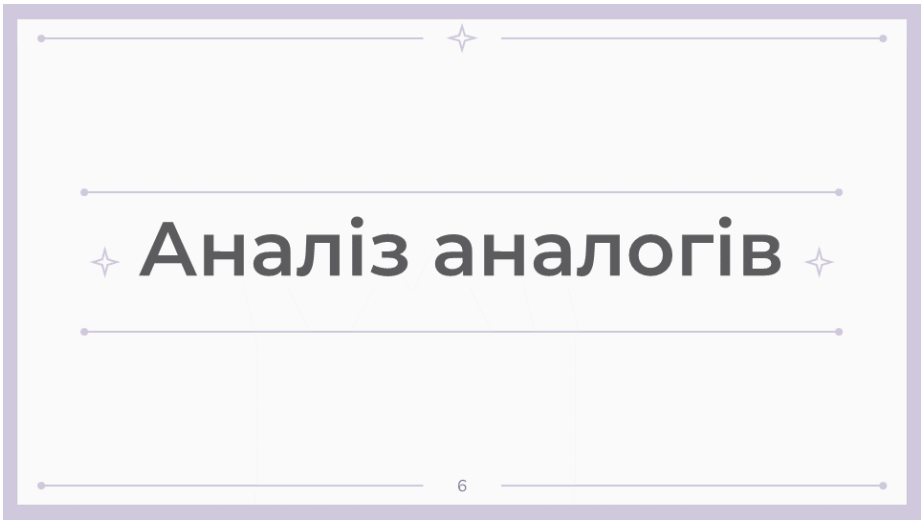

Рисунок Г.6 – Аналіз аналогів

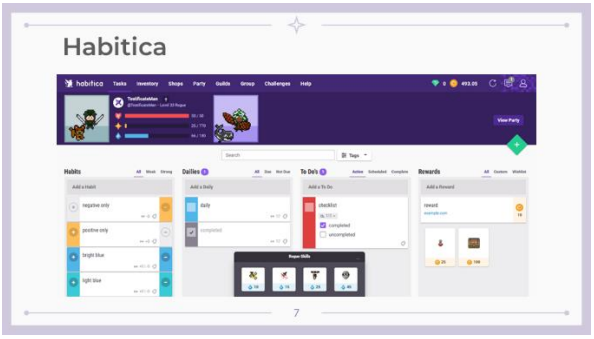

Рисунок Г.7 – Аналог Habitica

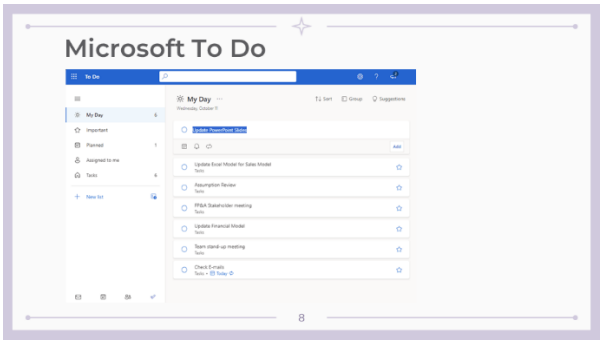

Рисунок Г.8 – Аналог Microsoft To Do

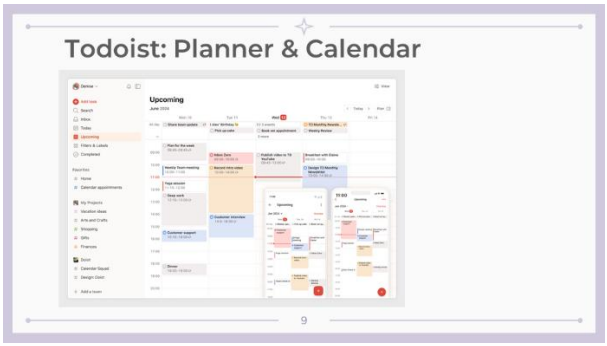

Рисунок Г.9 – Аналог Todoist: Planner & Calendar

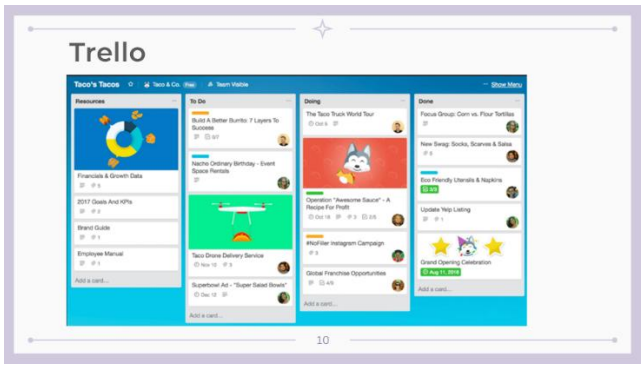

Рисунок Г.10 – Аналог Trello

Рисунок Г.11 – Технічна реалізація

Рисунок Г.12 – Алгоритм роботи чатсина 1

Рисунок Г.13 – Алгоритм роботи частина 2

Рисунок Г.14 – Інтерфейс застосунку

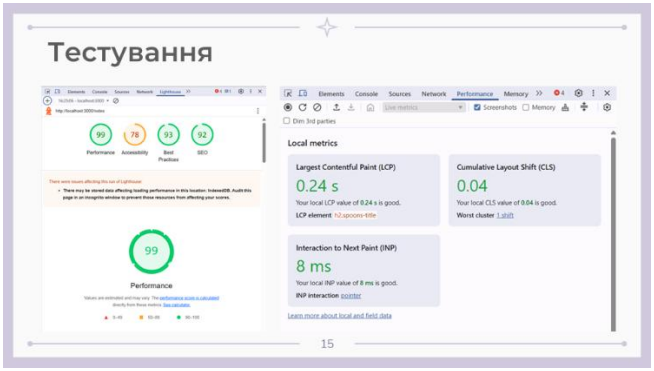

Рисунок Г.15 – Тестування

Рисунок Г.16 – Обгрунтування пропозиції

Рисунок Г.17 – Апробація та публікація

Рисунок Г.18 – Висновки