Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

**РОЗРОБКА ВЕБЗАСТОСУНКУ ЩОДЕННОГО ОБЛІКУ ВЖИВАННЯ ВОДИ**
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-1

Денисенко А.В.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Вечірська І.Д.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

# ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Денисенку Артему Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебзастосунку щоденного обліку вживання води

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 3 травня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі.

4. Перелік питань, що потрібно опрацювати в роботі

1. Провести аналіз вебзастосунків щоденного обліку вживання води, виявити їх слабкі та сильні сторони.

2. Спроєктувати вебзастосунок, визначити його структуру, компоненти та взаємодію між ними.

3. Релізувати вебзастосунок щоденного обліку вживання води, визначити його перспективи.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) ER-діаграма взаємозв'язків.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на кваліфікаційну роботу | 08.04.2024 | |
| 2 | Аналіз завдання, підбір літератури | 08.04.24-15.04.24 | |
| 3 | Аналіз літератури з досліджуваної проблеми | 16.04.24-18.04.24 | |
| 4 | Аналіз технічних засобів | 19.04.24-25.04.24 | |
| 5 | Програмна реалізація | 25.04.24-01.06.24 | |
| 6 | Оформлення пояснювальної записки | 01.06.24-08.06.24 | |
| 7 | Перевірка на плагіат | 09.06.24 | |
| 8 | Рецензування | 10.06.24 | |
| 9 | Підготовка презентації та доповіді | 10.06.24-12.06.24 | |
| 10 | Занесення роботи в електронний архів | 12.06.24 | |
| 11 | Попередній захист кваліфікаційної роботи | 12.06.24 | |

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ _____ доц. Вечірська І.Д._____
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 70 с., 18 рис., 30 джерела.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВЕБЗАСТОСУНОК, JS, CSS, NODE.JS.

Об’єктом роботи є облік щоденного вживання води.

Метою роботи є розробка вебзастосунку обліку щоденного вживання води, щоб полегшити процес приведення до норми водного балансу.

Було проведено аналіз вже розроблених вебзастосунків для обліку щоденного вживання води, на основі результатів якого було виявлено їх переваги та недоліки.

В ході роботи було використано середовище VS Code, мову програмування JavaScript, БД MongoDB, платформу Node.js, фреймворк Express, бібліотеки React, Mongoose, CORS, JWT, Redux, Axios, Formik.

У результаті роботи здійснена програмна реалізація вебзастосунку обліку щоденного вживання води.

SOFTWARE, WEB APPLICATION, JS, CSS, NODE.JS.

The object of work is a web application for accounting for daily water consumption.

The purpose of the work is to develop a web application for accounting for daily water consumption to facilitate the process of bringing the water balance back to normal.

An analysis of already developed web applications for accounting for daily water consumption was conducted, based on the results of which their advantages and disadvantages were identified.

The VS Code environment, JavaScript programming language, MongoDB database, Node.js platform, Express framework, React, Mongoose, CORS, JWT, Redux, Axios, Formik libraries were used in the course of the work.

As a result of the work, a software implementation of a web application for accounting for daily water consumption was carried out.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Гейміфікація – використання ігрових практик та механізмів у неігровому конктексті

SEO (Search Engine Optimization) — це процес оптимізації веб-сайту або веб-додатка для покращення його видимості в результатах пошукових систем

API – Application Programming Interface (інтерфейс програмування застосунків)

JSON – JavaScript Object Notation (формат обміну даними)

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення)

Middleware - функції, що мають доступ до об’єкту запиту, об’єкту відповіді і до наступної функції проміжної обробки в циклі "запит-відповідь" застосунку

I/O – (введення/виведення)

# ВСТУП

У сучасному світі, де темп життя прискорюється з кожним днем, здоров’я і благополуччя є одними з найважливіших пріоритетів для багатьох людей. Одним з найважливіших елементів підтримки здоров’я є вживання потрібної кількості води. Вебзастосунки, спеціально розроблені для відстеження споживання води, стають невід’ємною частиною нашого повсякденного життя і допомагають нам гармонізувати наші фізичні потреби з повсякденними обов’язками.

Добре відомо, що недостатнє споживання рідини може мати негативні наслідки, такі як зневоднення, погіршення роботи мозку та загального стану здоров’я. За допомогою вебзастосунків ми можемо відстежувати споживання рідини і переконатися, що забезпечуємо свій організм потрібною кількістю води для оптимального функціонування.

Вебзастосунки обліку щоденного споживання води є актуальними завдяки зростаючому інтересу населення країни до власного здоров’я, збільшується кількість відвідувачів спортивних залів та фітнес центрів. Збільшуючи фізичну активність, збільшується потреба в спожитій воді. За допомогою вебзастосунків можна легко та зручно оптимізувати цей процес. Вказавши необхідні дані, вебзастосунок може розрахувати необхідну користувачеві кількість води й нагадувати про необхідність прийняття наступної порції рідини.

# 1 ТЕОРЕТИЧНИЙ АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Аналіз тематики та теорії

Цей розділ присвячений аналізу актуальності теми вебзастосунку для обліку щоденного вживання води, огляду існуючих рішень та теоретичних основ, на яких базується ця розробка. Ми розглянемо значення підтримки водного балансу для здоров’я, проаналізуємо поточний стан ринку подібних застосунків, їхні переваги та недоліки, а також теоретичні аспекти, пов’язані з розробкою та використанням таких систем.

### 1.1.1 Значення підтримки водного балансу для здоров’я людини

Вода є основою життя, і її значення для здоров’я людини неможливо переоцінити. Вона становить близько 60% маси тіла дорослої людини і бере участь у багатьох життєво важливих процесах. Підтримка водного балансу – це баланс між споживанням та втратою води, що є критично важливим для нормального функціонування організму [1].

Однією з головних функцій води є участь у метаболічних процесах. Вода необхідна для перетравлення їжі, абсорбції поживних речовин та виведення відходів з організму. Нестача води може уповільнити метаболізм, призвести до запорів та інших проблем із травленням.

Вода також відіграє ключову роль у регуляції температури тіла. Через потовиділення організм охолоджується, запобігаючи перегріванню. В умовах високої температури або інтенсивного фізичного навантаження потреба у воді зростає, і недостатнє її споживання може призвести до теплового удару.

Адекватне споживання води є важливим для здоров’я шкіри. Вода допомагає підтримувати її зволоженість, пружність та здоровий вигляд.

Зневоднення може зробити шкіру сухою, лускатою та менш еластичною, сприяючи передчасному старінню.

Вода відіграє вирішальну роль у функціонуванні нирок, які відповідають за виведення токсинів з крові та підтримання балансу електролітів. Недостатнє споживання води може призвести до утворення каменів у нирках та інших порушень функції нирок.

Мозок і нервова система також залежать від достатньої кількості води. Навіть легке зневоднення може негативно вплинути на когнітивні функції, спричинити головний біль, втому, зниження концентрації та погіршення настрою. Для дітей та підлітків, які активно ростуть та розвиваються, підтримка водного балансу є особливо важливою для оптимального фізичного та розумового розвитку.

Вода необхідна для здоров'я серцево-судинної системи. Вона забезпечує об'єм крові, що підтримує артеріальний тиск. Зневоднення може призвести до зниження об'єму крові, що змушує серце працювати інтенсивніше, підвищуючи ризик серцевих захворювань.

Таким чином, підтримка водного балансу є фундаментальною для здоров'я і добробуту людини. Вживання достатньої кількості води щодня є простим, але ефективним способом підтримувати функціонування всіх систем організму, запобігати багатьом захворюванням і забезпечувати високу якість життя. Регулярне споживання води, особливо у поєднанні зі здоровим харчуванням та активним способом життя, є запорукою довголіття та відмінного самопочуття.

### 1.1.2 Вимоги до щоденного споживання води

Щоденне споживання води є ключовим аспектом підтримки здоров'я та добробуту людини. Вода відіграє важливу роль у багатьох життєво важливих процесах, включаючи травлення, абсорбцію поживних речовин, циркуляцію

крові та регуляцію температури тіла. Визначення оптимальної кількості води, яку слід споживати щодня, є критично важливим завданням, що враховує індивідуальні потреби кожної людини.

Загальні рекомендації щодо споживання води часто вказують на правило «8 склянок води на день», що приблизно дорівнює 2 літрам. Однак ця рекомендація є досить спрощеною і не враховує різноманітні фактори, що впливають на потребу в воді. Потреба в воді може значно варіюватися залежно від віку, статі, маси тіла, рівня фізичної активності, кліматичних умов та загального стану здоров'я.

Одним із найбільш точних методів визначення потреби у воді є розрахунок на основі маси тіла. Зазвичай рекомендується споживати приблизно 30 мл води на кожен кілограм маси тіла. Таким чином, людина з масою 70 кг повинна споживати близько 2,1 літра води на день. Ця формула дозволяє індивідуалізувати рекомендації та враховувати специфічні потреби кожної особи.

Рівень фізичної активності також суттєво впливає на потребу у воді. Під час фізичних навантажень організм втрачає воду через піт, тому спортсменам та людям, які активно займаються спортом, необхідно споживати більше води для компенсації втрат. Важливо пити воду перед, під час та після фізичних вправ, щоб підтримувати оптимальний рівень гідратації.

Кліматичні умови є ще одним важливим фактором. У спекотну погоду або в умовах високої вологості потреба у воді зростає, оскільки організм інтенсивніше втрачає воду через потовиділення. В таких умовах необхідно частіше пити воду навіть тоді, коли немає відчуття спраги, оскільки спрага є пізнім сигналом зневоднення.

Вік та стан здоров'я також впливають на потребу в воді. Діти та підлітки, які активно ростуть, потребують більше води відносно своєї маси тіла, ніж дорослі. Літні люди часто відчувають меншу спрагу, що може призвести до недостатнього споживання води, тому для них важливо свідомо контролювати свій водний баланс. Люди з певними захворюваннями, такими як діабет або

ниркові проблеми, потребують спеціальних рекомендацій щодо споживання води.

Необхідно враховувати також споживання рідини з їжею та іншими напоями. Овочі, фрукти, супи та інші рідкі страви вносять вагомий внесок у загальне споживання води. Наприклад, кавуни та огірки містять понад 90% води, що допомагає підтримувати гідратацію.

Підтримка оптимального водного балансу вимагає свідомого підходу та врахування індивідуальних потреб організму. Регулярне споживання достатньої кількості води сприяє підтримці високого рівня енергії, поліпшенню фізичної та розумової продуктивності, а також забезпечує здоров’я і добробут на довгострокову перспективу. Важливо розуміти власні потреби в воді та слідкувати за ними, щоб забезпечити організму всі необхідні умови для нормального функціонування.

## 1.2 Актуальність теми

Вода є основним компонентом людського організму, і її достатнє споживання є критично важливим для підтримання здоров’я. Нестача води може призвести до зневоднення, що негативно впливає на всі системи організму, включаючи мозок, серце, нирки та м’язи. З іншого боку, надмірне споживання води також може мати шкідливі наслідки, такі як водне інтоксикація.

У сучасному світі, де люди ведуть активний спосіб життя і часто забувають про регулярне споживання води, особливо важливо мати засоби, які допомагають контролювати водний баланс. Вебзастосунки для обліку вживання води стають дедалі популярнішими, оскільки вони дозволяють користувачам легко стежити за своєю гідратацією, отримувати нагадування про необхідність пити воду та аналізувати свої звички [2, 3].

### 1.2.1 Огляд існуючих рішень

На ринку існує кілька популярних застосунків для обліку вживання води, таких як «Водокотик» (рисунок 1.1) [4], «Water Time» [5], «My Water» [6] та інші. Розглянемо їх основні характеристики, переваги та недоліки.

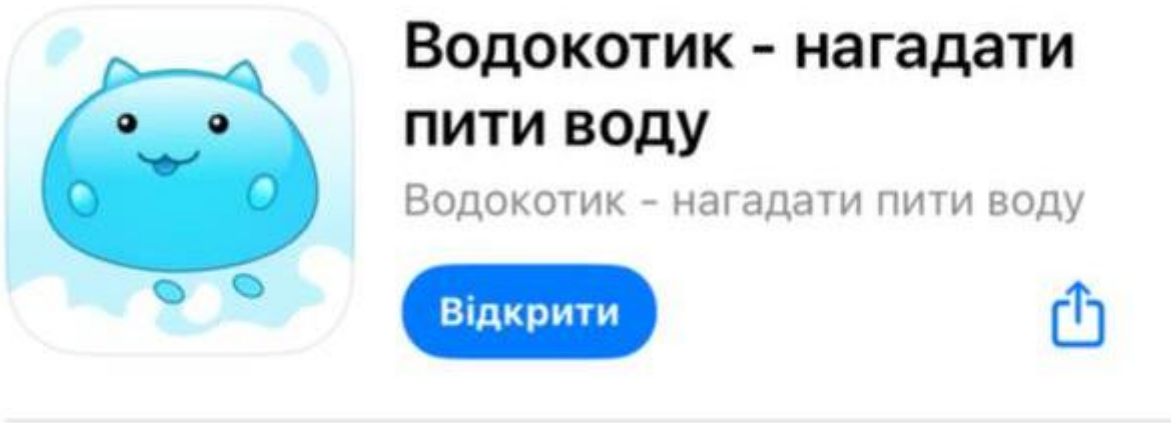


Рисунок 1.1 – Логотип застосунку «Водокотик»

До переваг застосунку «Водокотик» можна віднести темний мод, обширний функціонал на головному екрані застосунку (рисунок 1.2), відсутня платна версія, можна додавати наявність фізичної активності та можливість вказати погоду, що корегує денну норму вживання води (рисунок 1.3), зручне меню додавання води, процес гейміфіковано, наявна система досягнень, залежно від кількості вжитої води змінюється маскот на головному екрані (рисунок 1.4), підтримка різних видів напоїв.

До недоліків застосунку «Водокотик» можна віднести відсутність можливості вказати час вжитої води раніше, тільки актуальний час, машинний український переклад і через це не зовсім точний.

Розглянемо головний екран вебзастосунку «Водокотик». Вгорі розташовано шкалу прогресу денної норми. В центрі екрану розташований персонаж, котрий змінюється, в залежності від вжитої норми води, знизу екрану розташовано меню навігації.

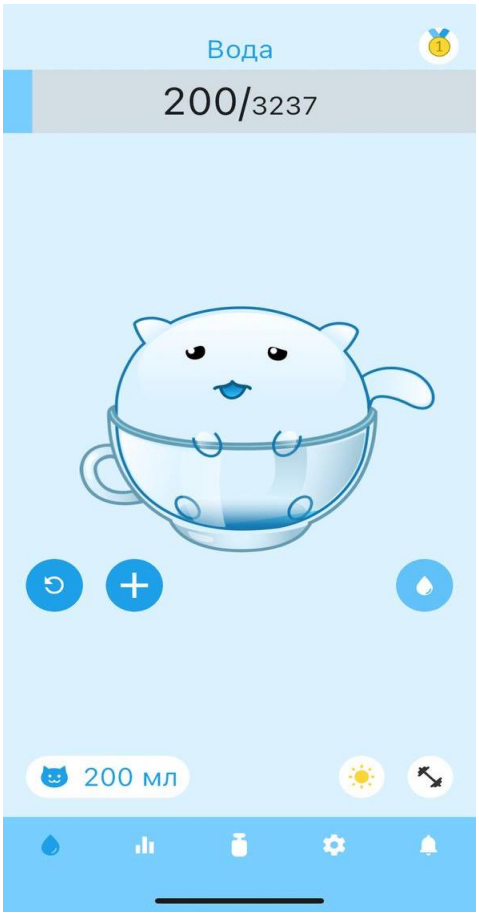


Рисунок 1.2 – Головний екран застосунку «Водокотик»

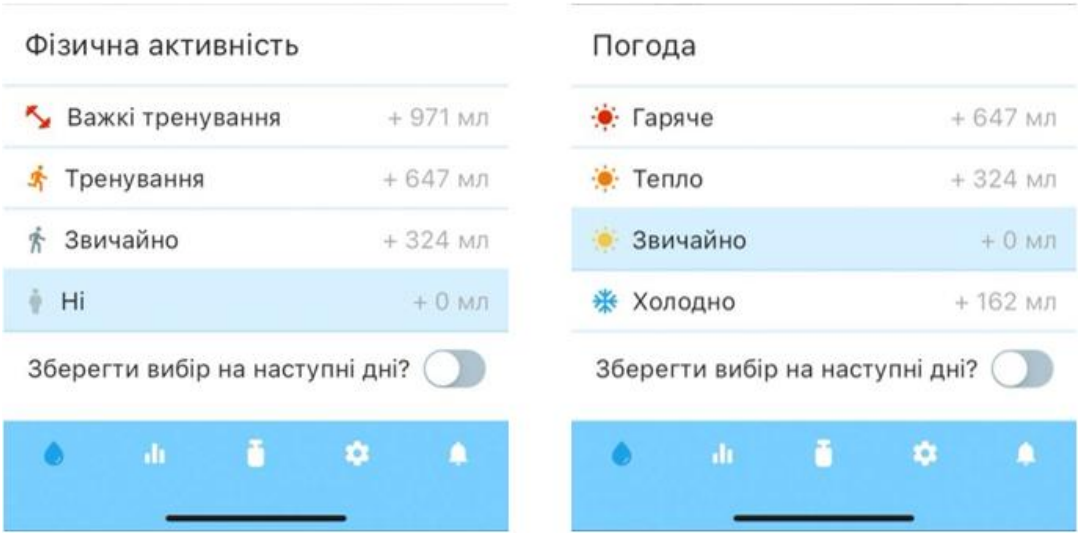


Рисунок 1.3 – налаштування для застосунку «Водокотик», котрі змінюють кількість денної норми

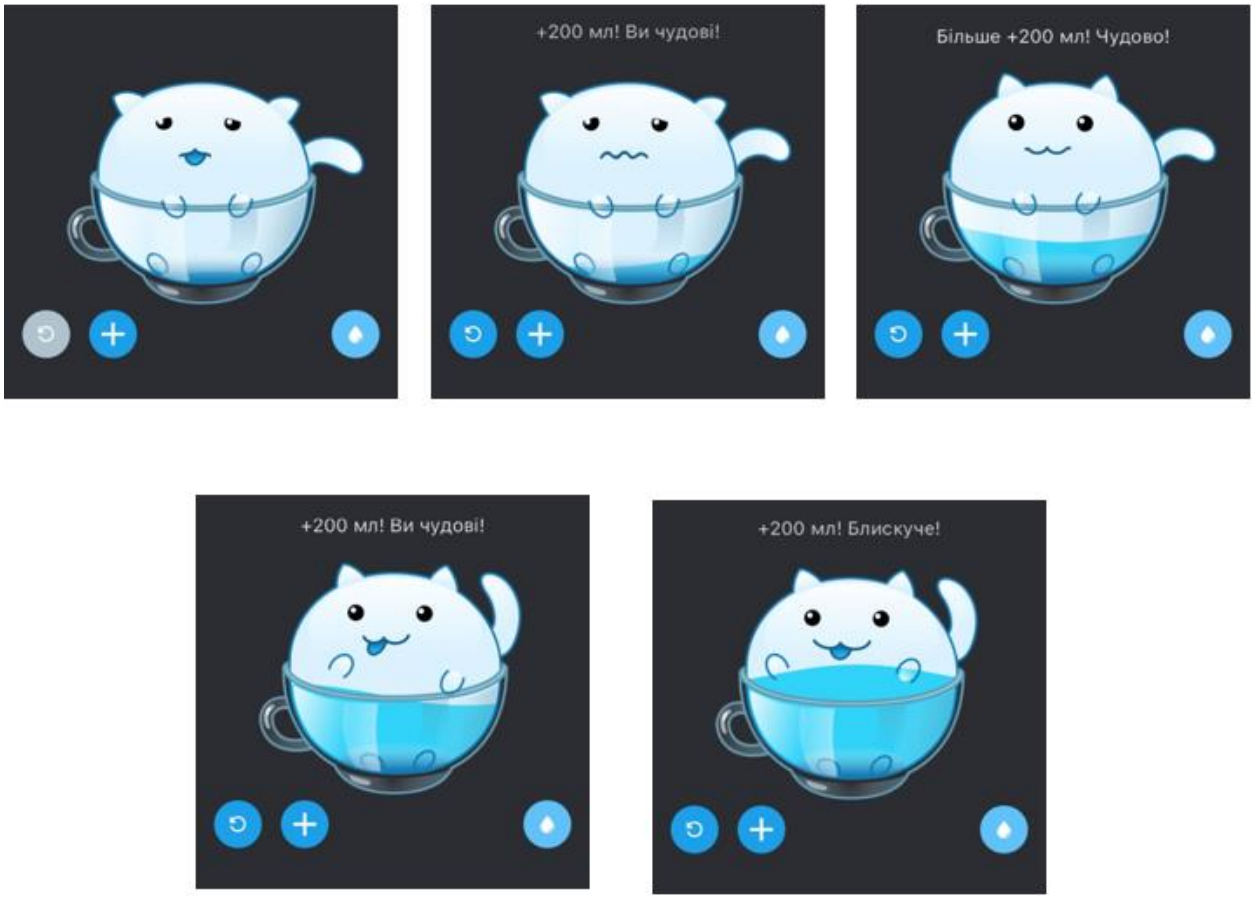


Рисунок 1.4 – Демонстрація зміни персонажу на головному екрані застосунку «Водокотик», в залежності від вжитої кількості води

Наступним розглянемо вебзастосунок «Water Time» (рисунок 1.5).


Рисунок 1.5 – Логотип застосунку «Water Time»

До переваг вебзастосунку «Water Time» можна віднести наявну синхронізацію із застосунками здоров’я, також залежно від кількості вжитої води змінюється персонаж на головному екрані, застосунок простий у використанні, наявна підтримка різних видів напоїв, після розрахунку денної норми води, можна обрати поступове досягнення норми (рисунок 1.6).

Серед недоліки вебзастосунку є наявність реклами на головному екрані застосунку (рисунок 1.7), відсутній темний мод, відсутня можливість вказати час вжитої води раніше, тільки актуальний час; обмежена функціональність у безкоштовній версії.


Рисунок 1.6 – Меню вибору поступового досягнення норми води

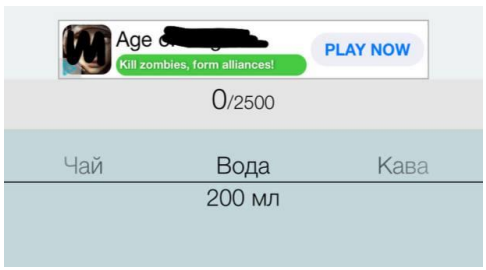


Рисунок 1.7 – Фрагмент головного екрану застосунку «Water Time»

Наступним розглянемо вебзастосунок «My water» (рисунок 1.8).


Рисунок 1.8 – Логотип застосунку «My Water»

До переваг вебзастосунку «My Water» можна віднести зручну й обширну статистика, наявний темний режим, можна обрати стиль застосунку та іконку застосунку, наявна система досягнень.

До недоліків вебзастосунку «My Water» можна віднести наявність реклами на головному екрані, обмеженість більшості функціоналу у безкоштовній версії, відсутність української мови інтерфейсу, відсутність можливості додати випиту рідину в минулі години поточного дня.

Аналізуючи ці рішення, можна виділити кілька ключових аспектів, які варто врахувати при розробці власного вебзастосунку:

– інтуїтивний інтерфейс, простота використання є критично важливою для залучення користувачів;

– персоналізація, можливість налаштування цілей та нагадувань на основі індивідуальних параметрів користувача;

– інтеграція, підтримка інтеграції з іншими фітнес-застосунками та гаджетами для більш повного контролю здоров’я;

– доступність функцій, баланс між безкоштовними та преміум-функціями, щоб залучити більше користувачів.

### 1.2.2 Теоретичні основи

Для створення ефективного вебзастосунку необхідно враховувати сучасні технології та підходи до веброзробки. Для реалізації користувацького інтерфейсу: HTML, CSS, JavaScript, фреймворки React.js для створення динамічних інтерфейсів. Для реалізації серверної частини вебзастосунку: Node.js, Express.js для реалізації серверної логіки та обробки запитів. Для реалізації бази даних: MongoDB для зберігання даних користувачів та споживання води. Для безпеки: використання HTTPS, JWT для аутентифікації, шифрування даних.

Алгоритми, що використовуються в застосунках для обліку вживання води, включають обробку введених даних, генерацію статистики, а також алгоритми для нагадувань та оповіщень:

– обробка введених даних, зберігання даних про кількість спожитої води, дата і час у базі даних;

– генерація статистики, обчислення середньодобового споживання, виявлення тенденцій і аномалій;

– нагадування, алгоритми для надсилання нагадувань на основі заданих інтервалів часу або умов (наприклад, через певний проміжок часу після останнього введення даних).

## 1.3 Постановка задачі

Таким чином, вебзастосунок для щоденного вживання води є актуальним. Тому ставиться завдання реалізації такого вебзастосунку.

Об’єктом роботи є облік щоденного вживання води.

Метою роботи є розробка вебзастосунка обліку щоденного вживання води з метою полегшення підтримання водного балансу користувача.

Розробити вебзастосунок для обліку щоденного вживання води, який буде мати наступні функціональні можливості:

– інтуїтивний інтерфейс для введення даних про споживання води;

– персоналізація цілей та нагадувань на основі індивідуальних параметрів користувача;

– відображення статистики та аналітики у вигляді графіків і діаграм;

– підтримка інтеграції з іншими фітнес-застосунками та гаджетами;

– забезпечення високого рівня безпеки даних користувачів.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз існуючих рішень, виявити їх сильні та слабкі сторони;

– спроєктувати вебзастосунок, визначити структуру вебзастосунку, його компоненти та взаємодію між ними;

– реалізувати вебзастосунок щоденного обліку вживання води;

– визначити перспективи вебзастосунку.

# 2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

## 2.1 Архітектура вебзастосунку

Архітектура вебзастосунку є важливим етапом розробки, який визначає структуру, компоненти та взаємодію між ними. В нашому випадку, для розробки вебзастосунку обліку щоденного вживання води ми використовуємо архітектуру, яка забезпечує масштабованість, надійність та зручність у використанні [7].

У сучасній веброзробці змінилися не тільки техніки, що дозволяють вебсайтам виглядати краще, завантажуватися швидше і бути приємнішими у використанні. Насамперед змінилися фундаментальні речі – те, як ми проектуємо та створюємо вебзастосунки.

Візьмемо довільний вебсайт, наприклад, для роботи з колекцією рецептів, розкладом тренувань тощо. Завжди є набір сторінок (рисунок 2.1): домашня, профіль, сторінка колекції та сторінка одного елемента колекції.

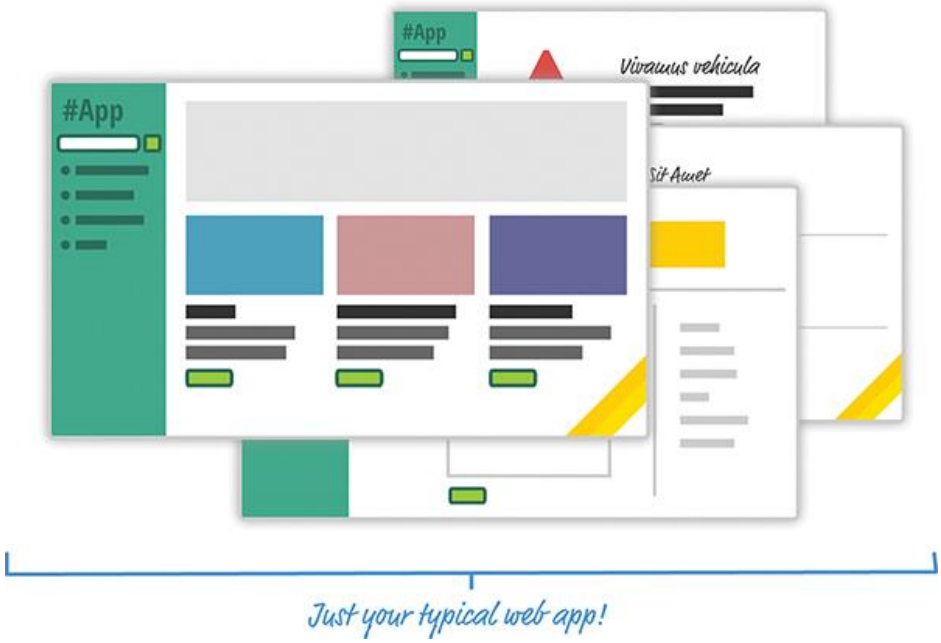


Рисунок 2.1 – Демонстрація набору сторінок вебсайту

Кілька років тому активно використовувався Multiple-page Application підхід (рисунок 2.2), який включає декілька окремих HTML-сторінок. Цей метод характеризується:

– архітектура клієнт-сервер, де сервер відповідає за зберігання та обробку даних, а клієнт здійснює запити до сервера для отримання необхідної інформації;

– вся бізнес-логіка та обробка даних знаходиться на серверній стороні;

– на кожен запит сервер надсилає готовий HTML-документ;

– при кожному запиті до сервера відбувається повне перезавантаження вебсторінки;

– через постійні перезавантаження сторінок інтерактивність користувацького інтерфейсу може бути знижена;

– відмінне SEO, додаток добре індексується пошуковими системами.


Рисунок 2.2 – Демонстрація підходу кількох окремих HTML-сторінок

Сучасний підхід Single-page Application (рисунок 2.3) – сайт, на якому користувач ніколи не переходить на інші HTML-сторінки. Інтерфейс, замість запиту HTML-документів з сервера, оновлюється на клієнті, на одній і тій самій сторінці, без перезавантаження. Розглянемо підхід детальніше:

– архітектура клієнт-сервер, де сервер відповідає за зберігання та обробку даних, а клієнт здійснює запити до сервера для отримання необхідної інформації;

– при завантаженні сайту сервер завжди повертає стартову HTML-сторінку – index.html, яка служить базовим шаблоном для подальшої роботи застосунку;

– кожен наступний запит на сервер отримує лише дані у JSON-форматі, що дозволяє уникнути повного перезавантаження сторінки та підвищує ефективність взаємодії між клієнтом і сервером;

– оновлення інтерфейсу відбувається динамічно на стороні клієнті, що забезпечує швидкий і плавний користувацький досвід без необхідності перезавантаження сторінки;

– завантаження першої сторінки може бути досить повільним через необхідність завантаження всі необхідних ресурсів, проблема може бути вирішена за допомогою різних методів оптимізації;

– вся бізнес-логіка, що не пов’язана з безпекою, виконується на стороні клієнта, що дозволяє зменшити навантаження на сервер і покращити швидкість застосунку;

– має обмеження у сфері SEO, оскільки пошукові системи мають складнощі з індексацією динамічного контенту, ці обмеження можна частково вирішити за допомогою технологій серверного рендерингу або гібридних підходів;

– зі збільшенням кількості функціональних можливостей застосунку складність коду та його підтримки зростає, що вимагає використання добре структурованої архітектури та передових практик розробки [8].

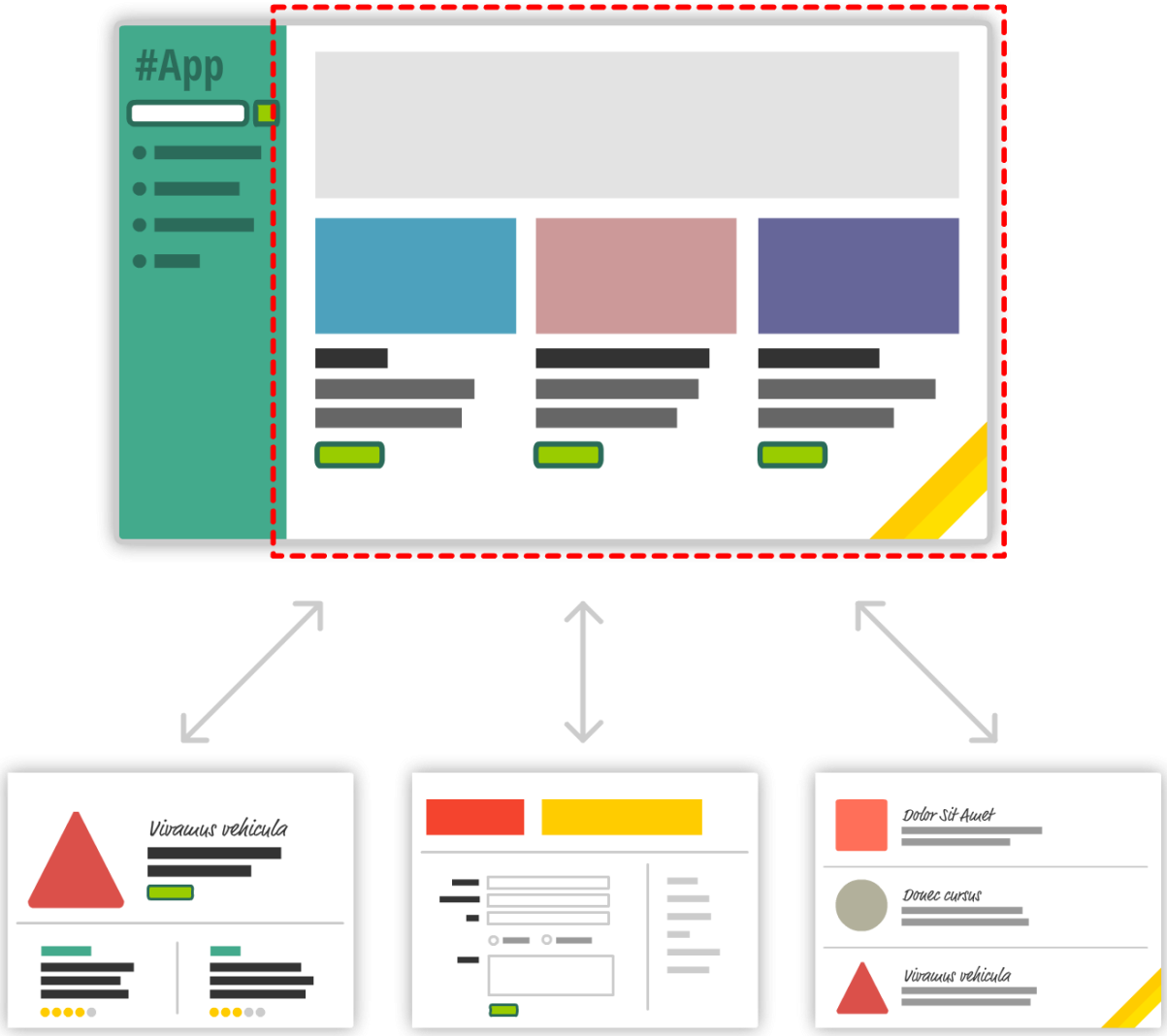


Рисунок 2.3 – Демонстрація підходу односторінкового застосунку

### 2.1.1 Загальна структура застосунку

Загальна структура вебзастосунку складається з трьох основних рівнів:

– клієнтський рівень;

– серверний рівень;

– рівень бази даних.

Кожен з цих рівнів виконує свою функцію і взаємодіє з іншими через чітко визначені інтерфейси.

Frontend відповідає за взаємодію з користувачем. Він надає інтерфейс, через який користувач може вводити дані про споживання води, переглядати статистику, отримувати нагадування та інше. Основні технології для розробки клієнтської частини:

– HTML для структури вебсторінок;

– CSS для стилізації;

– JavaScript для інтерактивності;

– React.js для побудови компонентної архітектури і забезпечення динамічності сторінок.

Backend забезпечує обробку запитів від клієнта, управління бізнес-логікою та взаємодію з базою даних. Основні функції серверного рівня включають аутентифікацію користувачів, обробку даних, управління сесіями і забезпечення безпеки. Технології для серверної частини: Node.js, Express.js, REST API, JWT (JSON Web Tokens).

Node.js для створення серверної логіки. Як асинхронне середовище виконання JavaScript, кероване подіями, Node.js призначений для створення масштабованих мережевих застосунків.

Express.js для маршрутизації запитів. Це мінімалістичний та гнучкий фреймворк для вебзастосунків, побудованих на Node.js, що надає широкий набір функціональності.

REST API для взаємодії між клієнтом і сервером [9].

JWT (JSON Web Tokens) для аутентифікації і авторизації.

База даних зберігає всю необхідну інформацію про користувачів і їх споживання води. Вибір бази даних залежить від вимог до продуктивності, масштабованості та типу даних. В даному випадку підходящими варіантами можуть бути PostgreSQL або MySQL для реляційної бази даних, MongoDB для нереляційної бази даних, якщо дані мають динамічну структуру.

### 2.1.2 Аналіз технологій для розробки

Для розробки клієнтської частини вебзастосунку обираємо наступні технології: React, Redux, Axios.

React.js забезпечує створення компонентного інтерфейсу, що полегшує підтримку та масштабування коду, React – це JavaScript бібліотека для рендерингу користувацьких інтерфейсів (UI), UI будується з невеликих блоків, таких як кнопки, текст та зображення, React дозволяє об’єднувати їх у багаторазові вкладені компоненти, від вебсайтів до телефонних застосунків, все що ви бачите на екрані, можна розбити на компоненти [10, 11].

Redux це шаблон і бібліотека для управління та оновлення стану програми за допомогою подій, які називаються «діями», вона слугує централізованим сховищем стану, який має використовуватися у всьому вашому застосунку, з правилами, що гарантують, що стан може бути оновлений лише у передбачуваний спосіб [12].

Axios для здійснення HTTP-запитів до серверної частини, Axios – це клієнт HTTP на основі Promise для Node.js та браузера, він ізоморфний (він може працювати у браузері та nodejs з тією ж базою кодів). На стороні сервера він використовує рідний http-модуль node.js, тоді як на клієнті (браузер) він використовує XMLHttpRequests [13, 14].

Для розробки серверної частини вебзастосунку обираємо наступні технології: Node.js, Express.js, JWT, MongoDB.

Node.js використовується для написання серверної логіки на JavaScript, що дозволяє використовувати одну мову програмування як на клієнтській, так і на серверній частині.

Express.js фреймворк для Node.js, що забезпечує мінімалістичний підхід до створення серверних застосунків.

JWT для безпечної аутентифікації та авторизації користувачів.

MongoDB нереляційна база даних для гнучкого зберігання даних користувачів.

Взаємодія між рівнями між рівнями застосунку відбувається наступним чином: серверний рівень надає REST API для взаємодії з клієнтським рівнем, кожна функціональність (реєстрація користувача, введення даних про споживання води, отримання статистики) реалізується як окремий API-метод. Клієнтська частина використовує AJAX-запити [15] для комунікації з сервером, що забезпечує асинхронність і покращує користувацький досвід.

Таким чином, архітектура вебзастосунку обліку щоденного вживання води включає три основні рівні (Frontend, Backend, Database), кожен з яких виконує свої специфічні функції і взаємодіє з іншими рівнями через чітко визначені інтерфейси. Це забезпечує масштабованість, надійність і зручність у використанні вебзастосунку.

## 2.2 Логіка та функціональні можливості

У цьому розділі детально описуються основні функціональні можливості вебзастосунку обліку щоденного вживання води, а також логіка його роботи. Це включає описи функціональних модулів, алгоритмів обчислення та відображення даних.

### 2.2.1 Опис функціональних модулів

Модуль реєстрації та аутентифікації користувачів відповідає за створення нових облікових записів користувачів та забезпечення їх входу до системи. Основні функції:

– реєстрація нового користувача, користувач вводить свою електронну пошту, пароль та, за бажанням, інші дані (наприклад, ім’я, вік, вагу для більш точної рекомендації щодо споживання води);

– вхід до системи, користувач вводить свої облікові дані (електронну пошту та пароль) для отримання доступу до свого облікового запису;

– відновлення паролю, користувач може запросити відновлення паролю через електронну пошту.

Модуль ведення обліку споживання води дозволяє користувачам вносити дані про кількість води, яку вони спожили протягом дня. Основні функції:

– додавання запису про спожиту воду, користувач вводить кількість води (в мл), дату та час споживання;

– редагування записів, користувач може змінювати раніше введені дані;

– видалення записів, користувач може видаляти неправильні або непотрібні записи.

Модуль статистики та аналітики надає користувачам можливість переглядати статистичну інформацію про їх споживання води. Основні функції:

– перегляд щоденної, щотижневої та щомісячної статистики;

– відображення графіків та діаграм для візуалізації даних;

– порівняння споживання води з рекомендованими нормами на основі індивідуальних параметрів користувача (вік, вага, фізична активність).

Модуль нагадувань допомагає користувачам не забувати про необхідність пити воду протягом дня. Серед основних функцій: налаштування нагадувань, користувач може встановити нагадування на певний час або з певними інтервалами та сповіщення – система надсилає повідомлення або push-сповіщення на пристрій користувача.

### 2.2.2 Алгоритми обчислення та відображення даних

Алгоритм обчислення рекомендованої кількості води:

Крок 1. Базова формула, рекомендована кількість води = 30 мл * вага користувача (кг).

Крок 2. врахування додаткових факторів: фізична активність, кліматичні умови, здоров’я тощо.

Крок 3. адаптація рекомендацій на основі індивідуальних параметрів користувача.

Алгоритм обробки введених даних:

Крок 1. Введення даних: користувач вводить кількість спожитої води.

Крок 2. Збереження даних: система зберігає дані у базі даних з прив’язкою до конкретного користувача.

Крок 3. Оновлення статистики: після кожного введення або редагування даних система перераховує загальну кількість спожитої води за день.

Алгоритм генерації статистики:

Крок 1. Збирання даних: система витягує з бази даних усі записи про спожиту воду за обраний період.

Крок 2. Обробка даних: система обчислює загальну кількість спожитої води, середнє споживання, визначає максимальні та мінімальні значення.

Крок 3. Візуалізація даних: система генерує графіки та діаграми для відображення статистики в зручному для користувача форматі.

### 2.2.3 Схема взаємодії користувача та відображення даних

Розглянемо взаємодію з інтерфейсом [16]. Реєстрація та вхід: користувач переходить на сторінку реєстрації, вводить необхідні дані та створює обліковий запис. Після цього він може увійти до системи, використовуючи свої облікові дані.

Введення даних: після входу користувач переходить на головну сторінку, де може ввести кількість спожитої води, обрати дату та час.

Перегляд статистики: користувач переходить на сторінку статистики, де може переглянути щоденну, щотижневу та щомісячну статистику у вигляді графіків та діаграм.

Налаштування нагадувань: користувач може встановити нагадування на певний час або з певними інтервалами на відповідній сторінці налаштувань.

Розглянемо взаємодію з сервером. Запити до серверу: при кожній дії користувача (реєстрація, вхід, введення даних, перегляд статистики) клієнтська частина надсилає запит до серверу через REST API.

Обробка запитів: сервер приймає запити, виконує необхідні операції (створення запису, оновлення даних, генерація статистики) і повертає відповідь клієнтській частині.

Відповіді серверу: сервер надсилає відповіді клієнтській частині, яка оновлює інтерфейс відповідно до отриманих даних.

Таким чином, логіка роботи вебзастосунку обліку щоденного вживання води базується на чітко визначених функціональних модулях та алгоритмах, що забезпечують ефективне введення, зберігання, обробку та відображення даних. Користувач отримує зручний інтерфейс для ведення обліку води, перегляду статистики та отримання нагадувань, що допомагає підтримувати здоровий водний баланс.

## 2.3 Схеми та алгоритми

У цьому розділі детально розглядаються схеми та алгоритми, які використовуються в вебзастосунку обліку щоденного вживання води. Це включає ER-діаграму бази даних, алгоритми роботи з даними, а також схему взаємодії користувача з вебзастосунком.

### 2.3.1 ER-діаграма бази даних

ER-діаграма (діаграма «сутність-зв’язок») є ключовим компонентом при проєктуванні бази даних [17, 18, 19], оскільки вона візуалізує структуру даних

та взаємозв’язки між ними [20, 21]. Основними сутностями є: user, waterintake, reminder.

Сутніть user:

– user_id (унікальний ідентифікатор);
– email (електронна пошта);
– password_hash (хеш паролю);
– name (ім’я користувача);
– age (вік);
– weight (вага);
– activity_level (рівень фізичної активності).

Сутніть waterintake:

– intake_id (унікальний ідентифікатор);
– user_id (ідентифікатор користувача, зовнішній ключ);
– amount (кількість води в мл);
– intake_date (дата споживання);
– intake_time (час споживання).

Сутність reminder:

– reminder_id (унікальний ідентифікатор);
– user_id (ідентифікатор користувача, зовнішній ключ);
– reminder_time (час нагадування);
– frequency (частота нагадувань).

Взаємозв’язки:

– кожен User може мати багато WaterIntake записів (зв’язок один-до-багатьох);
– кожен User може мати багато Reminder записів (зв’язок один-до-багатьох).

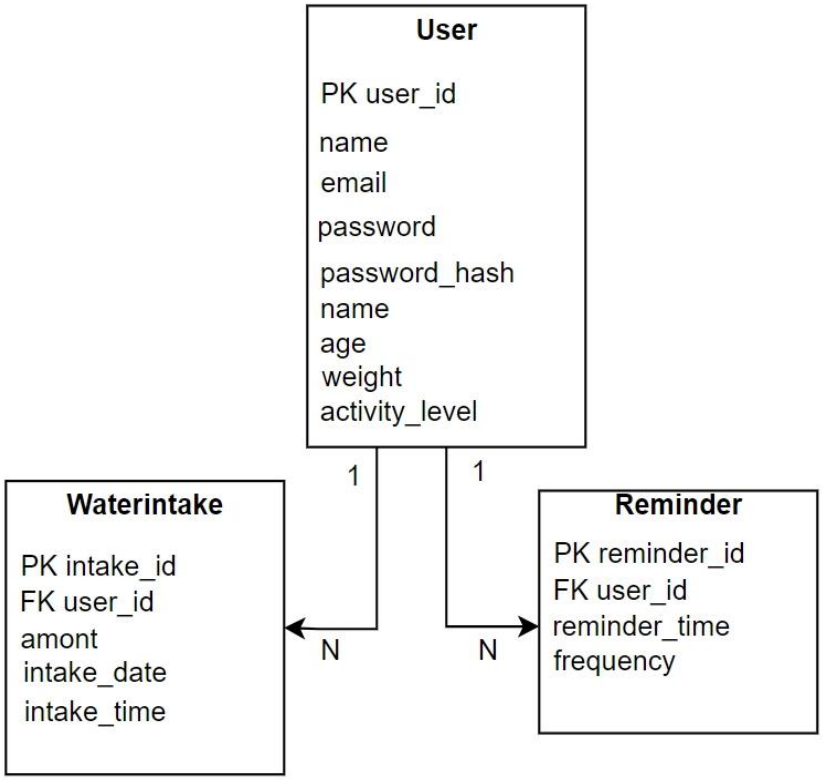


Рисунок 2.4 – ER-діаграма взаємозв’язків

### 2.3.2 Алгоритми роботи з даними

Алгоритм введення даних про споживання води:

Крок 1. Користувач вводить кількість спожитої води, дату і час у формі на клієнтському інтерфейсі.

Крок 2. Форма надсилає AJAX-запит на сервер з введеними даними.

Крок 3. Сервер приймає запит, перевіряє автентифікацію користувача та коректність введених даних.

Крок 4. Сервер зберігає дані у таблиці WaterIntake в базі даних.

Крок 5. Сервер повертає відповідь клієнту про успішне збереження даних.

Крок 6. Клієнт оновлює інтерфейс користувача, відображаючи новий запис.

Алгоритм генерації статистики:

Крок 1. Користувач запитує перегляд статистики за певний період.

Крок 2. Клієнт надсилає запит на сервер із зазначеним періодом (наприклад, щоденна, щотижнева, щомісячна статистика).

Крок 3. Сервер витягує всі записи з таблиці WaterIntake, що відповідають заданому періоду.

Крок 4. Сервер обробляє дані, обчислюючи загальну кількість спожитої води, середнє споживання за період, максимальні та мінімальні значення.

Крок 5. Сервер повертає клієнту оброблені дані.

Крок 6. Клієнт відображає дані у вигляді графіків і діаграм на інтерфейсі користувача.

Алгоритм налаштування та обробки нагадувань:

Крок 1. Користувач налаштовує нагадування, вказуючи час та частоту на клієнтському інтерфейсі.

Крок 2. Форма надсилає дані про налаштування нагадувань на сервер.

Крок 3. Сервер зберігає налаштування у таблиці Reminder в базі даних.

Крок 4. Сервер налаштовує механізм відправки нагадувань користувачу.

Крок 5. В заданий час сервер надсилає нагадування користувачу через обраний канал (пошту, push-сповіщення).

### 2.3.3 Схема взаємодії користувача з вебзастосунком

Реєстрація: користувач заповнює реєстраційну форму, дані надсилаються на сервер, де перевіряються на унікальність пошти і зберігаються у базі даних [22, 23].

Вхід: користувач вводить пошту і пароль, сервер перевіряє дані і при успішній автентифікації надсилає токен користувачу для подальшої взаємодії.

Розглянемо ведення даних про споживання води. Форма введення: користувач заповнює форму з кількістю води, датою і часом, дані надсилаються на сервер і зберігаються у базі даних. Відображення введених даних: введені дані оновлюються у реальному часі на інтерфейсі користувача.

Розглянемо перегляд статистики. Запит статистики: користувач вибирає період для перегляду статистики, запит надсилається на сервер. Обробка даних: сервер обчислює статистичні дані і надсилає їх на клієнт. Відображення: дані відображаються у вигляді графіків і діаграм на сторінці статистики.

Таким чином, схеми та алгоритми вебзастосунку обліку щоденного вживання води забезпечують ефективне введення, обробку і відображення даних, а також зручну взаємодію користувача з системою, що допомагає підтримувати здоровий водний баланс.

Реалізувати різний інтерфейс хедеру для авторизованих та неавторизованих користувачів.

Для неавторизованого користувача буде відображено компонент «Header», котрий містить компоненти «Logo» та «UserAuth». «Logo» – компонент, який відображає логотип застосунку і є клікабельним, переадресовує користувача на головну сторінку (для незареєстрованого користувача – «WelcomePage»), «UserAuth» – кнопку авторизації, котра буде перенаправляти за маршрутом авторизації.

Для авторизованого користувача буде відображено компонент «Logo» та «UserLogo» (кнопка відкриття модального вікна, що редагує дані про користувача. Дана кнопка містить в собі:

– якщо аватар присутній, ім’я користувача та його аватар, як контентне зображення;

– якщо аватар відсутній, ім’я користувача та в якості аватару першу літеру імені користувача у верхньому індексі;

– якщо відсутні ім’я та аватар, першу літеру пошти користувача.

Клік по кнопці повинен відкривати/закривати модальне вікно «UserLogoModal» (компонент, котрий рендерить модальне вікно з кнопками):

– «Settings» по кліку відкривається модальне вікно «SettingModal»;

– «LogOutBtn» відкриває модальне вікно «UserLogOutModal»;

«SettingModal» компонент рендерить модальне вікно з формою фону, коли відкривається модальне вікно, поля форми повинні відображати інформацію про користувача, яка раніше вже була ним збережена, у формі знаходяться поля:

– «Your photo», фото користувача, і кнопка «Upload a photo» для додавання/зміни фото користувача, після вибору відповідного зображення, відбувається запит на бекенд для його збереження;

– «Your gender identity» – радіо-кнопки для зміни гендера користувача;

– «Your name» – поле вводу для зміни імені користувача;

– «E-mail» – поле вводу для зміни пошти користувача;

– «Password» – блок для зміни пароля користувача складається з трьох полів вводу (Outdated password, New Password, Repeat new password).

Кожне поле для вводу пароля повинно мати можливість відображати та приховувати значення паролю. За замовчуванням значення паролю приховане.

– «Save» це кнопка типу submit для відправки даних на бекенд для оновлення даних по користувачу. Якщо від сервера була отримано повідомлення про помилку, необхідно його відобразити користувачеві за допомогою нотифікації, модальне вікно повинно залишатись відкритим.

Модальне вікно повинне закриватись:

– по кліку на фон;

– натисканню на клавішу Escape;

– по підтвердженню форми, після отримання успішної відповіді від сервера.

Розглянемо сторінки застосунку неавторизованого користувача. Сторінка-компонент «WelcomePage» рендериться на маршрут «/welcome».

Зверстати сторінку, на якій повинні знаходитись контейнери для блоків «WaterConsumptionTracker» та «WhyDrinkWater».

Компонент «WaterConsumtionTracker» містить в собі:

– заголовок «Water consumption tracker»;

– підзаголовок «Record daily water intake and track»;

– блок з підпунктів «Tracker Benefits» (пари іконка-текст: «Habit drive», «View statistics», «Personal rate setting»);

– кнопка «try tracker», по кліку на яку відбувається переадресація за маршрутом «/signup» на сторінку-компонент «SignupPage» з формою реєстрації.

Компонент «Why drink water» містить в собі підпункти:

– «Supply of nutrients to all organs»;

– «Providing oxygen to the lungs»;

– «Maintaining the work of the heart»;

– «Release of processed substances»;

– «Ensuring the stability of the internal environment»;

– «Maintaining within the normal temperature»;

– «Maintaining an immune system capable of resisting disease».

Компонент-сторінка «SignupPage» рендериться на маршрут «/signup». Сторінка реєстрації містить в собі компонент «AuthForm». Компонент «AuthForm» містить в собі форму і елемент навігації «Sign in». Кнопка «Sign up» реєструє користувача, а елемент навігації «Sign in» переадресовує користувача на сторінку «SigninPage». Всі поля валідуються, якщо введене значення невалідне, відповідне поле отримує необхідний стан. Після реєстрації користувач повинен бути переадресований на сторінку-компонент «SigninPage». Якщо з бекенда було отримано помилку реєстрації – користувачу необхідно вивести відповідну інформацію у вигляді нотифікації.

Компонент-сторінка «SigninPage» рендериться на маршрут «/signin». Сторінка логінізації містить в собі компонент «AuthForm». Компонент «AuthForm» містить в собі форму та елементи навігації «SignUp».

Кнопка «Sign in» активізує перевірку на валідність введеної користувачем інформації. У разі введення користувачем невалідних значень – візуалізувати це йому, змінивши стан. У разі їх валідності – відправляє запит на бекенд для авторизації користувача і переадресовує його на сторінку-компонент «HomePage». Якщо з бекенда було отримано помилку логінізації, користувачу необхідно вивести відповідну інформацію і вигляді нотифікації.

Елемент навігації «Sign Up» – переадресовує користувача на сторінку-компонент «SignUpPage».

Розглянемо сторінки-компоненти застосунку авторизованого користувача. Компонент «HomePage» рендериться на маршрут «/home». Зверстати головну сторінку, на якій повинні знаходитись контейнери для блоків:

– «DailyNorma» відображає заплановану денну норму води користувача;

– «WaterRatioPanel» показує відношення фактично випитої води до запланованої кількості, а також здійснює відкриття модального вікна в яке відбувається запис;

– «TodayWaterList» показує список порцій води, випитої користувачем протягом дня;

– «MonthStatsTable» дає можливість користувачеві переглянути узагальнену статистику стосовно випитої води протягом будь-якого дня.

У компоненті «DailyNorma» відображається розрахована денна норма споживання води для користувача. Наприклад, «1.5 л» (Якщо значення ще не було вирахувано для поточного користувача, за замовчуванням відображається 2.0 літри). Цей компонент має кнопку «Edit», по кліку на яку, повинне відкриватись модальне вікно для розрахунку денної норми споживання води «DailyNormaModal».

Компонент «DailyNormaModal» для розрахунку денної норми споживання води складається з блоків:

– заголовок, містить текст «My daily norma»;

– опис формули, показує формулу для обчислення норми води в залежності від статі;

– форма, отримує дані, обчислює їх та відправляє норму обрану користувачем на бекенд для збереження. Якщо денна норма вже була розрахована, слід відображати попередньо зазначені параметри для розрахунку.

Форма внесення значень для розрахунку: заголовок відображає текст «Calculate your rate». Форма містить в собі поля вибору гендеру, введення ваги, введення часу фізичної активності, поле розрахунку норми, поле для введення власної норми води та кнопку збереження інформації.

Вибір гендеру: для цього компонента потрібно використати радіо-кнопки, які дозволяють обрати один варіант. Передбачені два варіанти: «For girl» і «For man».

Введення ваги: цей компонент має бути текстовим полем, де користувач може ввести свою вагу у кілограмах.

Введення часу активних занять або фізичної активності: цей компонент також повинен бути текстовим полем, де користувач може ввести кількість часу, проведеного на активних заняттях або фізичній активності. Якщо користувач не займається активними заняттями, він може встановити значення 0.

Поле для виводу розрахункової кількості води. Як тільки користувач заповнив всі поля, буде проведений динамічний розрахунок денної норми споживання води згідно вибраного гендеру, введеної ваги та часу активності. На основі введених даних результат розрахунку (обсяг необхідної кількості води в літрах на день) буде відображений на екрані, в полі, біля напису «The required amount of water in liters per day».

Поле для запису кількості випитої води: це текстове поле, де користувач може записати кількість води в літрах, яку він планує випивати впродовж дня;

Кнопка «Save» – при натисканні на цю кнопку дані повинні бути відправлені на бекенд.

Якщо збереження даних на сервері відбулась успішно, модальне вікно має закритись, якщо з помилкою, то воно залишається відкритим і виводиться повідомлення у вигляді нотифікації. Також модальне вікно повинне закриватись по кліку на кнопку для закриття або фон, а також по натисканню на клавішу Esc.

Компонент «WaterRatioPanel» рендерить шкалу з відношенням випитої води за день відносно визначеної норми та кнопку «AddWater», що відкриває модальне вікно «TodayListModal» для вводу значення кількості випитої порції води.

Компонент TodayWaterList, містить історію про споживання води протягом дня:

– в цьому компоненті відображається список з порціями води випитими користувачем протягом дня;

– кожен запис (елемент списку) містить інформацію про обсяг випитої води (наприклад, «250 мл») та час споживання (наприклад, «7:00 AM»);

– користувач може додавати нові записи, використовуючи кнопку «Add water» з компоненту «WaterRatioPanel».

Компонент TodayListModal – модальне вікно для додавання або редагування даних по випитій води. Компонент складається з:

– заголовку редагування введеної кількісті води;

– дані щодо кількості води з попереднього прийому, показується значення кількості випитої води користувачем попереднього разу наприклад, «250 мл» та час, коли це було зроблено (наприклад «7:00AM»). Якщо відбувається перший запис за поточний день, компонент показує повідомлення «No notes yet».

Корегування введених даних складається з:

– блоку для покрокового вводу значення випитої води за допомогою кнопок «+» | «-» та панелі що відображає це значення;

– поля для вводу значення часу в год:хв, коли була випита вода, по дефолту поточний час, потрібно реалізувати як випадаючий список для вибору часу з кроком в 5хв;

– поля для вводу значення випитої води з клавіатури.

Блок для покрокового вводу кількості води та поле для вводу кількості води з клавіатури повинні показувати однакове значення, що відповідає к-сті води, яку користувач хоче зберегти в даному записі. За замовчуванням це – 0, якщо відбувається перший запис за день або значення к-сті води з попереднього прийому поточного дня. Коли відбувається зміна значення к-сті води за допомогою блоку або поля, в полі або блоці відповідно значення повинно змінитись лише після втрати фокусу на елементах того блоку, на якому користувач виконує зміну. Тобто, якщо користувач змінює значення в полі для вводу з клавіатури, то на панелі блоку з кнопками значення зміниться лише коли фокус на полі для вводу буде втрачено.

Це модальне вікно дозволяє користувачеві редагувати вже введену кількість води. Воно відображає поточні дані та надає можливість коригування і збереження змін.

Розглянемо склад компоненту «MonthStatsTable». Форма для зміни місяців. Блок з кнопками для зміни місяця на наступний або попередній та поле, що показує обраний місяць. За замовчуванням в полі показується поточний місяць. Користувач може вибрати будь-який місяць окрім наступних після поточного (в такому випадку стрілка переведена має приховуватись, не впливаючи на позиціювання інших елементів).

Перелік днів місяця з відсотками виконання плану відносно норми за кожен з днів. Компонент показує список блоків з інформацією по всім дням, які має місяць вказаний на формі зміни місяців. Кожен блок з інформацією по дню показує число даного дня в місяці та статистику по випитій за цей день води відносно норми. День за який користувач не виконав план, по випитій воді, повинен бути виділеним. По кліку на будь-який з днів, над ним повинен

бути показаний компонент «DaysGeneralStats», який містить загальну статистику по даному дню.

Компонент «DaysGeneralStats» – спливаюче вікно, що містить загальну інформацію та статистику відносно спожитої води за день місяця, який обрав користувач. Складається з розділів:

– дата, показує дату обраного дня в форматі числа та назви місяця, наприклад – «5 квітня»;

– денна норма, показує заплановану денну норму споживання води в літрах, наприклад: «1.8 л»;

– виконання денної норми, показує відношення у відсотках запланованої к-сті води до фактично спожитої к-сті за обраний день, наприклад, «60%» або «0%» якщо за день не було зроблено жодного запису;

– кількість прийнятих порцій води, показує кількість порцій води спожитої за день за день, наприклад, «6» або «0» якщо за день не було зроблено жодного запису.

Розглянемо серверну частину вебзастосунку. Розгорнути сервер для розробки: підключити необхідні модулі, налаштувати CORS для дозволу хостам на взаємодію з сервером, написати функції для відлову та обробки помилок.

База даних (БД): визначити таблиці та їх зв’язки, ініціалізувати та підключити БД до проєкту.

Розглянемо аутентифікацію. Реєстрація користувача: створити endpoint для реєстрації користувача, з перевірками унікальності інформації та створенням запису. Значення пошти повинно бути валідним. Значення паролю повинно містити мінімум 8 символів та максимум 64 символи. Логін користувача: створити endpoint для логіну користувача з перевіркою облікових даних та генерацією токену аутентифікації. Авторизація: написати прошарок авторизації, що перевірятиме наявність токену та відповідність прав доступу. Вихід користувача з системи: створити endpoint для виходу користувача з системи.

# 3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

## 3.1 Обґрунтування вибору середовища програмної реалізації

У рамках кваліфікаційної роботи був розроблений вебзастосунок обліку щоденного вживання води. Для реалізації було обране середовище Visual Studio Code. Visual Studio Code (VS Code) – це безкоштовний редактор коду з відкритим вихідним кодом, розроблений компанією Microsoft. Він призначений для розробки та відлагодження сучасних веб- і хмарних застосунків.

Підтримка багатьох мов програмування: VS Code підтримує такі мови, як JavaScript, TypeScript, Python, Java, C++, C#, PHP, Go, та багато інших, також є розширення, які дозволяють додавати підтримку нових мов.

Інтегроване управління версіями, VS Code має вбудовану підтримку для Git, що дозволяє легко керувати версіями вашого коду прямо з редактора.

Розширення та налаштування, є велика кількість розширень, які можна встановити з Visual Studio Code Marketplace, щоб додати нові функції або підтримку мов. Користувачі можуть також налаштовувати редактор відповідно до своїх потреб.

Інтелектуальна підказка коду, VS Code надає розумні підказки для коду (IntelliSense), які базуються на типах змінних, визначеннях функцій та інших елементах коду.

Відлагоджувач: вбудований відлагоджувач підтримує налагодження різних мов та платформ, дозволяючи ставити точки зупину, крокувати кодом та перевіряти змінні в реальному часі.

Підтримка командного рядка: VS Code має інтегрований термінал, що дозволяє виконувати командний рядок безпосередньо з редактора.

Кросплатформеність: VS Code доступний для Windows, macOS та Linux, що робить його універсальним інструментом для розробників на різних операційних системах.

Цей редактор став дуже популярним завдяки своїй гнучкості, потужним можливостям та активній спільноті користувачів та розробників, які постійно покращують його за допомогою нових розширень і функцій.

При розробці вебзастосунку було використано: мову програмування JavaScript, бібліотеку React для створення елементів інтерфейсу користувача, Платформу Node.js, вебфреймворк Express, ODM-бібліотеку Mongoose для роботи з MongoDB, JSON Web Token (JWT), документоорієнтовну БД MongoDB типу NoSQL, бібліотеку Redux для управління станами, бібліотеку Axios для здійснення запитів, бібліотеку Formik для форм.

React – це популярна бібліотека JavaScript, розроблена Facebook для створення користувацьких інтерфейсів. Вона використовує компонентний підхід, дозволяючи розробникам створювати повторно використовувані компоненти з власним станом та логікою. React використовує віртуальний DOM для підвищення продуктивності, оновлюючи тільки ті частини інтерфейсу, які змінилися. React має багато переваг розглянемо їх детальніше.

Компонентна архітектура дозволяє розділяти інтерфейс на невеликі, незалежні компоненти, що полегшує повторне використання коду та його обслуговування.

Віртуальний DOM підвищує продуктивність застосунків, оновлюючи лише змінені частини реального DOM, що зменшує кількість маніпуляцій з ним.

JSX дозволяє писати HTML-подібний код всередині JavaScript, що спрощує створення компонентів та читання коду.

Односпрямований потік даних робить поведінку застосунку більш передбачуваною та спрощує налагодження.

Широка екосистема та підтримка, велика кількість бібліотек, інструментів та активна спільнота, що постійно розвивається.

Також розглянемо недоліки React: крута крива навчання, потрібно вивчати не лише React, але й супутні інструменти та бібліотеки, такі як Redux, Router, та інші.

Відсутність офіційної структури: відкритість для різних підходів може спричиняти розбіжності в організації проектів, що ускладнює роботу в команді.

Часті оновлення: постійні оновлення бібліотеки та супутніх інструментів можуть вимагати регулярного оновлення та адаптації коду.

Проблеми з SEO: як і всі клієнтські бібліотеки, React може мати проблеми з пошуковою оптимізацією, хоча це можна вирішити за допомогою серверного рендерингу (наприклад, Next.js).

React підходить для створення складних, інтерактивних вебзастосунків, забезпечуючи високу продуктивність та зручність у розробці завдяки компонентній архітектурі та віртуальному DOM [24, 25].

Node.js – це кросплатформене середовище виконання для JavaScript, яке дозволяє запускати JavaScript-код поза браузером. Воно побудоване на рушії V8 від Google і використовує неблокуючий I/O та подійно-орієнтовану архітектуру, що забезпечує високу продуктивність та масштабованість. Розглянемо переваги Node.js.

Висока продуктивність: завдяки неблокуючому I/O та рушію V8 Node.js забезпечує швидке виконання коду та обробку великої кількості запитів.

Асинхронність: подійно-орієнтована архітектура дозволяє ефективно керувати асинхронними операціями, що важливо для реального часу застосунків.

Одна мова для клієнта та сервера: використання JavaScript як на клієнтській, так і на серверній стороні спрощує розробку та обслуговування застосунків.

Велика екосистема NPM: Node.js має доступ до найбільшого сховища модулів та бібліотек, що спрощує додавання нових функцій до застосунків.

Активна спільнота: велика та активна спільнота розробників, яка постійно створює нові інструменти, бібліотеки та ресурси.

Також розглянемо недоліки Node.js. Однопоточність: Node.js працює в однопоточному середовищі, що може бути обмеженням для CPU-інтенсивних завдань.

Необхідність управління асинхронністю: робота з асинхронними операціями може бути складною, особливо для новачків.

Менша продуктивність для великих обчислень: для важких обчислювальних завдань продуктивність Node.js може бути меншою в порівнянні з деякими іншими середовищами виконання;

Відсутність стабільних API: часті оновлення можуть призводити до змін у API, що може вимагати регулярного оновлення та адаптації коду.

Node.js є потужним інструментом для розробки сучасних вебзастосунків, особливо тих, які потребують обробки великої кількості одночасних запитів або реального часу.

Express – це мінімалістичний та гнучкий вебфреймворк для Node.js, який спрощує розробку серверних застосунків і API. Він надає потужний набір інструментів для обробки HTTP-запитів, маршрутизації, та управління middleware, дозволяючи швидко створювати масштабовані вебзастосунки. Розглянемо переваги Express простота та гнучкість: Express має простий та інтуїтивний API, що робить його легким у вивченні та використанні. Він не нав’язує жорстких структур, дозволяючи розробникам організовувати свій код на власний розсуд.

Масштабованість: завдяки middleware архітектурі, Express дозволяє додавати функціональність шляхом підключення додаткових модулів, що робить застосунки масштабованими та легкими в обслуговуванні.

Широка екосистема: велика кількість доступних плагінів та модулів через NPM дозволяє легко розширювати можливості Express.

Активна спільнота: велика спільнота користувачів та розробників, що забезпечує наявність безлічі ресурсів, навчальних матеріалів та підтримки.

Також розглянемо недоліки Express відсутність структурованості: гнучкість Express може стати недоліком для великих проектів, оскільки відсутність жорстких правил організації коду може призвести до неструктурованості та ускладнити обслуговування.

Ручне управління помилками: Express не надає вбудованих засобів для управління помилками, тому розробники повинні самостійно впроваджувати обробку помилок.

Обмежена функціональність: Express є мінімалістичним фреймворком, і багато функцій, доступних у більш повнофункціональних фреймворках, потрібно додавати вручну через плагіни та модулі.

Асинхронне програмування: робота з асинхронним кодом може бути складною, особливо для новачків, що може ускладнити розробку та відлагодження.

Express є потужним інструментом для швидкої та гнучкої розробки серверних застосунків та API, особливо підходящим для невеликих та середніх проектів. Його простота та можливість легко розширювати функціональність роблять його популярним вибором серед розробників Node.js.

MongoDB – це документоорієнтована база даних NoSQL, яка зберігає дані у форматі BSON (розширений JSON). Вона призначена для забезпечення високої продуктивності, масштабованості та гнучкості у зберіганні та обробці даних, що робить її популярним вибором для сучасних вебзастосунків та аналітичних платформ. Розглянемо переваги MongoDB: гнучкість схеми, MongoDB дозволяє зберігати документи з різними структурами в одній колекції, що забезпечує велику гнучкість у моделюванні даних та легкість у їх зміні.

Масштабованість: підтримка горизонтального масштабування через шардінг дозволяє ефективно обробляти великі обсяги даних та високі навантаження.

Висока продуктивність: MongoDB забезпечує швидке читання та запис даних завдяки використанню індексів та оптимізованому механізму зберігання.

Реплікація та відмовостійкість: підтримка реплікації даних на кілька серверів забезпечує високу доступність та захист від втрати даних;

Потужний API для запитів: MongoDB надає потужний API для виконання складних запитів, агрегацій та обробки даних безпосередньо в базі даних.

Також розглянемо недоліки MongoDB: високе споживання пам’яті, зберігання даних у форматі BSON може вимагати більше пам’яті порівняно з іншими форматами, що може призвести до збільшення витрат на інфраструктуру.

Відсутність транзакційності: до версії 4.0 підтримка транзакцій була обмежена, що ускладнювало обробку складних операцій, хоча нові версії додали цю можливість.

Управління даними: гнучкість схеми може призвести до неструктурованості даних, якщо не дотримуватися чітких правил та стандартів моделювання.

Навчальна крива: для новачків робота з MongoDB може бути складною через відмінності від традиційних реляційних баз даних.

MongoDB є потужним інструментом для зберігання та обробки великих обсягів даних з високою гнучкістю та продуктивністю, що робить її популярною для створення сучасних застосунків, особливо тих, які вимагають масштабованості та швидкої обробки даних.

Mongoose – це об’єктно-документний мапер (ODM) для Node.js, який дозволяє взаємодіяти з базою даних MongoDB за допомогою об’єктів JavaScript. Він забезпечує модельно-орієнтований підхід до роботи з MongoDB, дозволяючи створювати схеми для документів і здійснювати валідацію, перетворення та управління даними. Розглянемо переваги

Mongoose: схеми та моделі, Mongoose дозволяє створювати чіткі схеми для документів, що полегшує структурування та організацію даних.

Валідація даних: вбудована підтримка валідації даних на рівні схеми, що забезпечує коректність та консистентність даних у базі.

Мідлвари: Mongoose підтримує мідлвари (проміжні функції) для обробки даних до та після певних операцій, що дозволяє додаткову обробку або логування.

Зручні методи для роботи з даними: надає потужні методи для створення, читання, оновлення та видалення (CRUD) даних, що спрощує розробку застосунків.

Асоціації та популяції: підтримка референцій між документами та можливість «популяції» (заповнення референційних полів реальними даними) для більш ефективної роботи з пов’язаними даними.

Також розглянемо недоліки Mongoose виконання складних запитів: деколи Mongoose може бути менш гнучким для виконання складних агрегаційних запитів у порівнянні з використанням нативного драйвера MongoDB.

Відставання від функціональності MongoDB: оновлення Mongoose можуть іноді запізнюватися з підтримкою нових функцій MongoDB, що може обмежувати використання останніх можливостей бази даних.

Додатковий рівень абстракції: використання Mongoose додає додатковий рівень абстракції, що може ускладнити налагодження та діагностику проблем у порівнянні з використанням нативного драйвера MongoDB.

Продуктивність: у деяких випадках, через додатковий рівень абстракції, Mongoose може бути менш продуктивним, ніж нативні запити до MongoDB, особливо при обробці великих обсягів даних.

Mongoose є потужним інструментом для розробників Node.js, які працюють з MongoDB, забезпечуючи зручний і структурований спосіб

взаємодії з базою даних, а також спрощуючи роботу з даними завдяки валідації, схемам та зручним методам CRUD.

JWT (JSON Web Token) – це компактний, самодостатній спосіб передачі інформації між сторонами як JSON-об’єкт. Токени підписуються з використанням секрету або пари ключів (публічний/приватний), що забезпечує цілісність і безпеку даних. JWT широко використовується для автентифікації та авторизації у вебзастосунках. Переваги JWT самодостатність, JWT містить всю необхідну інформацію для аутентифікації, що дозволяє уникнути додаткових запитів до бази даних.

Безпека: підписані токени гарантують, що дані не були змінені. Використання шифрування забезпечує конфіденційність інформації.

Простота передачі: JWT можна легко передавати через URL, заголовки HTTP або параметри запиту завдяки їх компактному розміру.

Масштабованість: токени працюють однаково добре як на клієнтській, так і на серверній стороні, що робить їх ідеальними для масштабованих архітектур мікросервісів.

Стандартизований формат: JWT є відкритим стандартом (RFC 7519), що забезпечує його сумісність і підтримку в багатьох платформах і бібліотеках.

Також розглянемо недоліки JWT розмір токенів, велика кількість інформації в токені може збільшити його розмір, що може вплинути на швидкість передачі даних.

Відкликання токенів: JWT не має вбудованого механізму для відкликання токенів, що може створити проблеми з безпекою, якщо токен скомпрометований.

Термін дії: токени мають фіксований термін дії, що може бути недоліком у випадках, коли потрібен більш гнучкий контроль доступу.

Складність управління ключами: управління парами ключів для підпису і валідації токенів може бути складним завданням, особливо в масштабованих системах.

Вразливість до атаки повторного використання: якщо токен потрапляє до зловмисника, він може використовувати його до закінчення терміну дії, якщо не впроваджено додаткових механізмів безпеки.

JWT є потужним і гнучким інструментом для автентифікації та авторизації, який забезпечує безпеку і ефективність передачі даних, але потребує обережного підходу до управління ключами та строком дії токенів [29].

Redux – це бібліотека для управління станом застосунків JavaScript, особливо популярна в екосистемі React. Вона дозволяє централізовано зберігати стан застосунку в одному місці, що спрощує управління і передбачуваність стану. Redux використовує концепції єдиного сховища (store), дій (actions) та ред'юсерів (reducers), які визначають, як змінюється стан у відповідь на дії. Розглянемо переваги Redux.

Передбачуваність стану: оскільки весь стан зберігається в одному об'єкті, легко відстежувати та передбачати зміни стану застосунку;

Простота налагодження: завдяки єдиному сховищу та концепції незмінності стану, відлагодження застосунків стає простішим. Інструменти, такі як Redux DevTools, дозволяють відстежувати дії та стан у реальному часі.

Масштабованість: Redux добре працює в великих застосунках з багатьма компонентами, оскільки спрощує управління складним станом.

Сумісність з будь-яким UI фреймворком: хоча Redux часто використовують з React, він сумісний з будь-яким UI фреймворком або бібліотекою.

Уніфікація логіки: логіка управління станом централізована, що дозволяє легше розділяти і підтримувати код, особливо в командах.

Також розглянемо недоліки Redux. Додаткова складність: впровадження Redux додає додатковий рівень складності, що може бути надмірним для невеликих або простих застосунків.

Бойлерплейт код: Redux вимагає написання значної кількості коду для визначення дій, ред’юсерів та підключення компонентів до сховища, що може збільшити обсяг коду.

Крива навчання: для новачків концепції Redux можуть бути складними для розуміння та застосування.

Проблеми з продуктивністю: якщо не правильно організувати структуру стану та не оптимізувати компоненти, це може призвести до проблем з продуктивністю, таких як зайві перерендери.

Redux є потужним інструментом для управління станом у складних застосунках, забезпечуючи передбачуваність та простоту налагодження, проте його використання слід ретельно обдумати через додаткову складність та обсяг коду.

Axios – це популярна бібліотека для виконання HTTP-запитів у браузерах та Node.js. Вона забезпечує простий і зручний інтерфейс для роботи з промісами, що полегшує обробку асинхронних запитів, таких як отримання або відправка даних на сервер. Розглянемо переваги Axios. Простота використання: Axios має інтуїтивний і зручний API, що полегшує виконання HTTP-запитів.

Підтримка промісів: Axios використовує проміси, що дозволяє легко працювати з асинхронними запитами та забезпечує простий спосіб обробки успіху та помилок.

Інтерцептори: можливість встановлювати інтерцептори для запитів та відповідей, що дозволяє обробляти або змінювати запити перед їх надсиланням і відповідей перед їх обробкою.

Підтримка JSON: Axios автоматично перетворює JSON-дані, що полегшує роботу з API, які використовують JSON.

Вбудована обробка помилок: Axios надає зручні механізми для обробки помилок HTTP-запитів.

Браузер та Node.js: працює як у браузерах, так і в середовищі Node.js, що робить його універсальним інструментом для виконання HTTP-запитів у різних середовищах.

Також розглянемо недоліки Axios. Розмір бібліотеки: Axios є відносно великою бібліотекою, що може бути недоліком для проектів, де важлива мінімізація розміру коду.

Зовнішня залежність: використання Axios додає зовнішню залежність до проекту, що може бути проблемою з точки зору безпеки та управління залежностями.

Обмежена функціональність: Axios спеціалізується на виконанні HTTP-запитів, і його функціональність може бути обмеженою у порівнянні з більш повнофункціональними бібліотеками.

Конкуренція з Fetch API: сучасні браузери надають вбудований Fetch API, який має схожу функціональність і не вимагає додаткової бібліотеки, що може зробити Axios менш привабливим вибором.

Axios є потужним і зручним інструментом для виконання HTTP-запитів, який забезпечує простоту використання та підтримку промісів, але може мати обмеження у випадках, коли важливий розмір коду або необхідна ширша функціональність.

## 3.2 Реалізація функціональності

### 3.2.1 Реалізація інтерфейсу вебзастосунку

Першочергово при написанні frontend частини застосунку було вирішено розробити базові компоненти, котрі потім будуть використані знову: кнопку (лістинг 3.1) та модальне вікно, також було створено компоненти іконок та написано загальні стилі, налаштовано шрифти.

Лістинг 3.1 Реалізація базового компоненту кнопка:

```
import { StyledButton } from ‘./Button.styled’;
const Button = ({
  type,  label,
  icon,
  onClick,
  width,
  backgroundColor,
  textColor,
  fontSize,
  fontWieght,
  fontHeight,
}) => {
  return (
    <StyledButton
      type={type}
      onClick={onClick}
      width={width}
      $backgroundColor={backgroundColor}
      $textColor={textColor}
      $fontSize={fontSize}
      $fontWieght={fontWieght}
      $fontHeight={fontHeight}
    >
      {icon && <img src={icon} alt="Icon" />} {label}
    </StyledButton>
  );
};
export default Button;
```

Компоненти – основні будівельні блоки React-застосунків, за допомогою яких інтерфейс розділяється на незалежні частини.

Розробник створює невеликі компоненти, які можна поєднувати, щоб сформувати більші, або використовувати їх як самостійні елементи інтерфейсу. Найголовніше в цій концепції те, що і великі, і маленькі компоненти можна використовувати повторно і в поточному, і в новому проекті.

React-застосунок можна уявити як дерево компонентів. На верхньому рівні стоїть кореневий компонент, у якому вкладена довільна кількість інших компонентів. Кожен компонент повинен повернути JSX-розмітку, тим самим вказуючи, який HTML ми хочемо відрендерити в DOM.

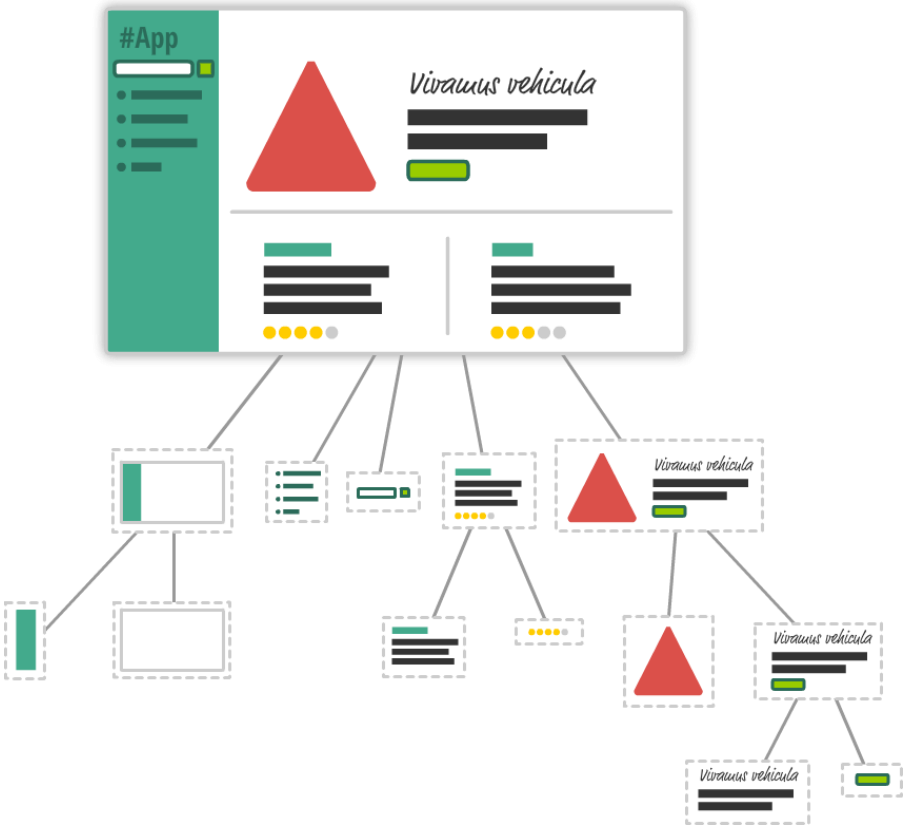


Рисунок 3.1 – Демонстрація складання сторінки з компонентів

Компоненти-функції складають більшу частину React-застосунку. Це дає такі переваги:

– менше boilerplate-коду;

– легше сприймати;

– легше тестувати;

– немає контексту (this) [30].

Розглянемо ред’юсер, що відповідає за управління станом аутентифікації та авторизації користувача у вебзастосунку. Ред’юсер визначений у файлі slice.js з використанням бібліотеки Redux Toolkit. Його основне завдання полягає в обробці асинхронних операцій, таких як реєстрація, вхід, вихід з системи, оновлення інформації про користувача та отримання актуальних даних.

Ред’юсер використовує початковий стан (initialState), який визначає структуру та початкові значення стану (лістинг 3.2).

Лістинг 3.2 Початковий стан:

```
const initialState = {
  user: {
    name: null,
    email: null,
    password: null,
    avatarURL: null,
    gender: null,
    dailyNorma: null,
  },
  token: null,
  isLoggedIn: false,
  isRefreshing: false,
  error: null,
  isLoading: false,
};
```

Цей початковий стан включає:

– інформацію про користувача: ім’я, пошту, пароль, URL аватара, стать, щоденна норма води;

– токен аутентифікації (token);

– прапор (isLoggedIn), що вказує, чи користувач увійшов у систему;

– прапор (isRefreshing), що вказує на процес оновлення даних користувача;

– поле для зберігання помилок (error);

– прапор (isLoading), що вказує на процес завантаження.

Обробка асинхронних операцій. Ред’юсер використовує createSlice для визначення стану та обробки різних асинхронних дій через extraReducers.

Вхідні та відхилені стани. Для обробки станів «очікування» (pending) та «відхилення» (rejected) використовуються окремі функції (лістинг 3.3).

Лістинг 3.3 Демонстрація реалізації функції для обробки станів:

```
const handleRejected = (state, action) => {
  state.isLoading = false;
  state.error = action.payload;
  state.isLoading = false;
};
const handlePending = (state, action) => {
  state.isLoading = true;
};
```

Обробка дій. Дії обробляються за допомогою extraReducers, що використовує builder для додавання обробників різних станів дій (лістинг 3.4).

Лістинг 3.4 Обробка дій:

```
const authSlice = createSlice({
  name: ‘auth’,
  initialState,
```

```
extraReducers: builder => {
  builder
    .addCase(signUp.pending, state => {
      state.isLoading = true;
    })
    .addCase(signUp.fulfilled, (state, { payload }) => {
      state.user = payload.user;
      state.token = payload.token;
      state.isLoggedIn = true;
      state.error = '';
      state.isLoading = false;
    })
    .addCase(signIn.pending, handlePending)
    .addCase(signIn.fulfilled, (state, { payload }) => {
      state.user = payload.user;
      state.token = payload.token;
      state.isLoggedIn = true;
      state.error = '';
      state.isLoading = false;
    })
    .addCase(logOut.pending, handlePending)
    .addCase(logOut.fulfilled, state => {
      state.user = { name: null, email: null };
      state.token = null;
      state.isLoggedIn = false;
      state.isLoading = false;
    })
    .addCase(refreshUser.pending, state => {
      state.isRefreshing = true;
    })
```

```
      .addCase(refreshUser.fulfilled, (state, action) => {
        state.user = action.payload;
        state.isLoggedIn = true;
        state.isRefreshing = false;
      })
      .addCase(refreshUser.rejected, (state, action) => {
        state.isRefreshing = false;
      })
      .addCase(updateWaterRate.fulfilled, (state, { payload }) => {
        state.user.waterRate = payload.waterRate;
      })
      .addCase(signIn.rejected, handleRejected)
      .addCase(signUp.rejected, handleRejected);
  },
});
```

Докладний опис обробки дій реєстрації:

– pending, встановлює isLoading у true, сигналізуючи про початок процесу реєстрації;

– fulfilled, зберігає дані користувача і токен у стані, встановлює isLoggedIn у true, очищає error і isLoading;

– rejected, обробляється функцією handleRejected, яка встановлює isLoading у false і зберігає помилку у стані.

Докладний опис обробки дій signIn (вхід):

– pending, обробляється функцією handlePending, яка встановлює isLoading у true;

– fulfilled, зберігає дані користувача і токен у стані, встановлює isLoggedIn у true, очищає error і isLoading;

– rejected, обробляється функцією handleRejected.

Докладний опис обробки дій logOut (вихід):

– pending, обробляється функцією handlePending;

– fulfilled, очищає дані користувача, токен і встановлює isLoggedIn у false, встановлює isLoading у false.

Докладний опис обробки дій оновлення даних користувача:

– pending, встановлює isRefreshing у true;

– fulfilled, зберігає актуальні дані користувача у стані, встановлює isLoggedIn у true і isRefreshing у false;

– rejected,, встановлює isRefreshing у false.

UpdateWaterRate (оновлення норми води): fulfilled – оновлює норму води (waterRate) користувача у стані.

Ред'юсер у файлі slice.js є ключовим компонентом управління станом аутентифікації у вебзастосунку. Використання Redux Toolkit дозволяє спростити конфігурацію і управління станом, забезпечуючи передбачуваність та централізованість. Це спрощує підтримку, налагодження та тестування застосунків. На рисунку 3.2 представлено, як виглядає інтерфейс модального вікна додавання вжитої води. На рисунку 3.3 представлено, як виглядає інтерфейс модального вікна форми розрахунку денної норми води. На рисунку 3.4 представлено, як виглядає інтерфейс модального вікна редагування вжитої води.

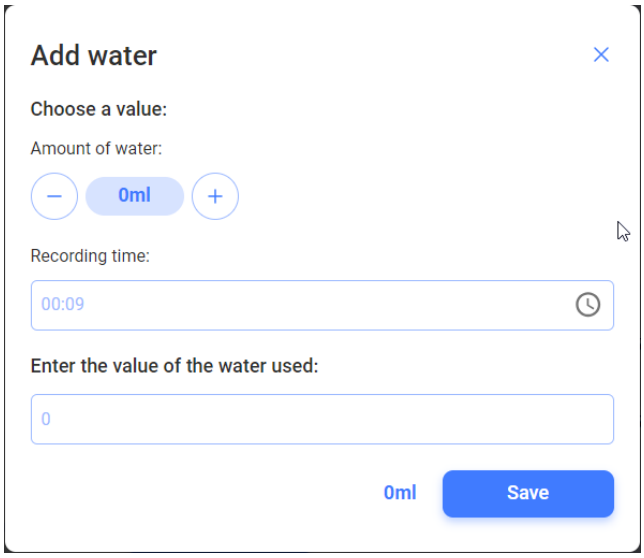


Рисунок 3.2 – Модальне вікно додавання вжитої води

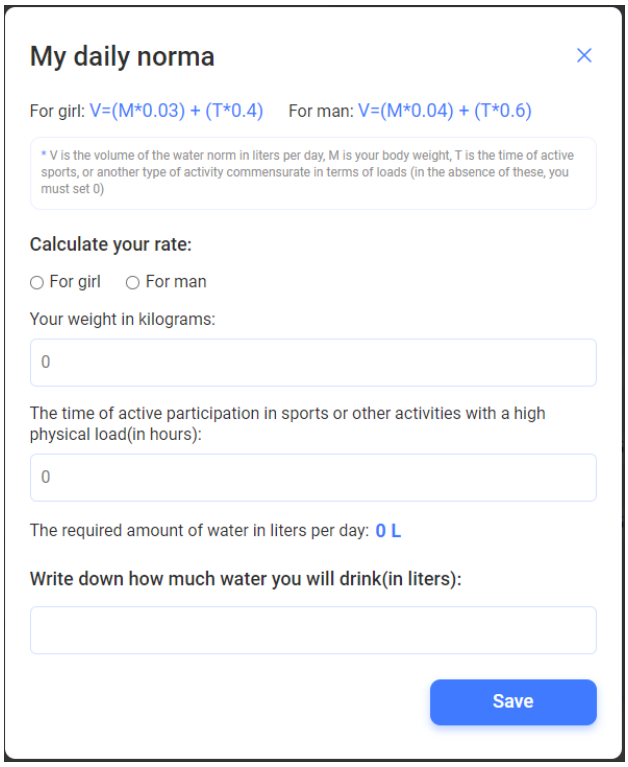


Рисунок 3.3 – Модальне вікно розрахунку денної норми вживання води

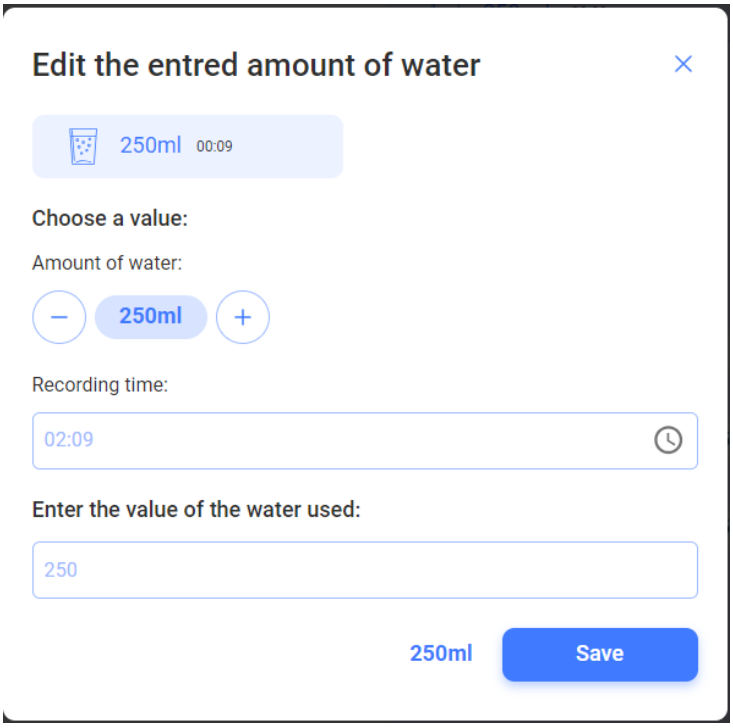


Рисунок 3.4 – Модальне вікно редагування введеної в журнал кількості вжитої води

### 3.2.2 Реалізація серверної частини

Архітектура застосунку базується на модульному підході, що забезпечує розподіл логіки на окремі компоненти. Основні компоненти включають: сервер, роутери, контролери, моделі, мідлвари.

Сервер: головний файл, що відповідає за налаштування сервера та підключення основних middleware (лістинг 3.5).

Лістинг 3.5 Реалізація серверу:

```
const app = require("./app");
const { connect } = require("mongoose");
require("dotenv").config();
const { MONGO_URI, PORT = 5000 } = process.env;
const server = async () => {
  try {
    const db = await connect(MONGO_URI);
    console.log(`Database "${db.connection.name}" connection successful`);
    app.listen(PORT, () => {
      console.log(`Server running. Use our API on port: ${PORT}`);
    });
  } catch (error) {
    console.log(error.message);
    process.exit(1);
  }
};
server();
```

Роутери: визначають маршрути та обробляють HTTP запити (лістинг 3.6).

Лістинг 3.6 WaterRouter:

```
waterRouter
  .route("/")
  .all(authVerification)
  .get(getTodayWater)
  .post(validateData(addWaterSchema), addWater);
```

Контролери: містять логіку обробки запитів та формування відповідей (лістинг 3.6).

Лістинг 3.6 WaterControllers:

```
const getTodayWater = decorateConrtoller(async (req, res) => {
  const result = await getTodayWaterService(req.user._id);
  res.json(result);
});
```

Моделі: визначають структуру даних та взаємодію з базою даних.

Middleware: проміжне ПЗ для обробки запитів, таких як аутентифікація, логування тощо.

Лістинг 3.7 middleware isEmptyBody:

```
const { HttpError } = require("../utils");
const isEmptyBody = async (req, res, next) => {
  const keys = Object.keys(req.body);
  if (!keys.length) {
    next(new HttpError(400, "missing fields"));
  }
  next();
};
module.exports = isEmptyBody;
```

### 3.2.3 Реалізація бази даних вебзастосунку

Для зручної роботи з базою даних для MongoDB було використано графічний редактор MongoDB Compass. В ньому було налаштовано підключення хмарної БД до Compass. Через Compass створили базу даних db-watertracker (рис. 3.1) і в ній колекції waternotes та users (рис. 3.2).

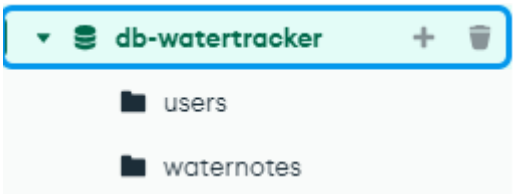


Рисунок 3.5 – БД watertracker

users

| | _id ObjectId | email String | password String | token String | avatarURL String |
|---|---|---|---|---|---|
| 1 | ObjectId('658c5398e3e9eefd204… | "testuser@testmail.com" | "$2a$10$R6GXqzs81mLJnS3383uTE… | "eyJhbGciOiJIUzI1NiIsInR5cCI6… | "https://res.cloudinary.com/d… |
| 2 | ObjectId('658c8145d50997c3e0d… | "serhii@gmail.com" | "$2a$10$sMI7WS15hoMzAuJc8e4/G… | "" | "" |
| 3 | ObjectId('658f248e9b1b47adc11… | "marina.smile1m@gmail.com" | "$2a$10$0gPPm/GZEwAJI8lfMKZrE… | "" | "" |
| 4 | ObjectId('65918efc413e1b3b865… | "xipipat992@regapts.com" | "$2a$10$rI7lfd7XY0U4J1oNQSye3… | "" | "avatars/65918efc413e1b3b8652… |
| 5 | ObjectId('6591d441bdcf58caf0d… | "test123@gmail.com" | "$2a$10$8Emyypiq2h0MeNy5rFHAj… | "" | "" |
| 6 | ObjectId('6592ad341f6666215c3… | "12e@gmail.com" | "$2a$10$ScIw0KGT.Eg5jo/yyhsI1… | "" | "" |
| 7 | ObjectId('659310397e83e01b1ab… | "marinatest@gmail.com" | "$2a$10$Kj9Xp5xT9Rv2yP2Z.B8Bz… | "eyJhbGciOiJIUzI1NiIsInR5cCI6… | "" |
| 8 | ObjectId('6594585eec6e22b793a… | "usertest@mail.com" | "$2a$10$ELGlQqYR6Tz3XOBiyTMSQ… | "" | "" |
| 9 | ObjectId('65965ead72524bc5a39… | "serhii_test@gmail.com" | "$2a$10$jueaZk2kqVzaZBR90B56U… | "" | "avatars/65965ead72524bc5a393… |
| 10 | ObjectId('65967280f007cdad08a… | "2@mail.com" | "$2a$10$dzAO2GDkFj4FGZBMYcXFJ… | "eyJhbGciOiJIUzI1NiIsInR5cCI6… | "" |
| 11 | ObjectId('65971865ddeb2b05977… | "testup@ukr.net" | "$2a$10$42bGkdTSixJK0cI923jN1… | "eyJhbGciOiJIUzI1NiIsInR5cCI6… | "" |
| 12 | ObjectId('6598395ec9862bee1b3… | "artem.test@gmail.com" | "$2a$10$J/Gjfm31vxKVdRdkOQ0/W… | "eyJhbGciOiJIUzI1NiIsInR5cCI6… | "" |

Рисунок 3.6 – Демонстрація колекції users в базі даних

## 3.3 Перспективи застосунку

Вебзастосунок для обліку щоденно спожитої води вже має широкий набір функцій, які роблять його надзвичайно зручним та ефективним

інструментом для підтримки водного балансу. Нижче наведено описано реалізований функціоналу вебзастосунку щоденного обліку вживання води.

Функціонал вирахування норми води за формулою. Однією з ключових функцій є можливість автоматичного розрахунку щоденної норми споживання води. Застосунок використовує формулу, яка враховує стать, вагу та зріст користувача, що дозволяє надавати персоналізовані рекомендації. Це забезпечує точність і індивідуальний підхід, що дуже важливо для ефективного підтримання водного балансу.

Функціонал можливості завантаження аватару. Для підвищення рівня персоналізації користувачі можуть завантажувати свої аватари. Це додає індивідуальності профілю кожного користувача та робить інтерфейс більш привабливим і дружнім.

Функціонал реєстрація та авторизація. Вебзастосунок пропонує зручний процес реєстрації та авторизації. Користувачі можуть швидко створити свій обліковий запис і почати користуватися всіма можливостями застосунку. Це забезпечує безпеку даних і персональний доступ до індивідуальної статистики.

Функціонал вказування часу випитої рідини у минулому протягом поточного дня. Для більш точного обліку спожитої рідини користувачі мають можливість зазначати час випитої води не лише в реальному часі, але й заднім числом протягом поточного дня. Це особливо корисно для тих, хто забув внести дані вчасно, але бажає зберегти точність обліку.

Функціонал зручний журнал випитої рідини. Застосунок має зручний журнал, де зазначається час вживання та літраж випитої рідини. Це дозволяє користувачам легко відстежувати свій прогрес протягом дня та аналізувати звички щодо споживання води.

Функціонал календарю з детальною статистикою**.** Вбудований календар дозволяє користувачам зручно переглядати статистику споживання рідини за будь-який день. При наведенні на конкретну дату з’являється спливаюче вікно, яке містить загальну інформацію про обраний день: дату, денну норму,

результат виконання денної норми та кількість спожитих порцій води. Це робить процес аналізу даних зручним і наочним.

Завдяки таким функціям, вебзастосунок не лише допомагає користувачам підтримувати водний баланс, але й робить цей процес зручним, зрозумілим і персоналізованим. Це сприяє більшій мотивації користувачів дотримуватися рекомендованих норм споживання води та досягати своїх цілей щодо здоров'я та добробуту.

Вебзастосунок для обліку щоденно спожитої води має значний потенціал для подальшого розвитку та вдосконалення. У нинішньому вигляді він уже допомагає користувачам підтримувати водний баланс у тілі, нагадує пити воду та веде облік спожитої рідини. Проте існує ще низка функцій, які можна реалізувати, щоб підвищити корисність та зручність застосунку.

Додавання інших різновидів рідин. Поточна версія вебзастосунку фокусується переважно на споживанні води. Однак для більш точного обліку загального водного балансу тіла доцільно включити можливість врахування інших рідин, таких як чай, кава, соки тощо. Це дозволить користувачам більш точно відстежувати кількість спожитої рідини і отримувати більш релевантні рекомендації щодо підтримки водного балансу.

Система оповіщення. Додатковою корисною функцією може стати система оповіщення, яка буде нагадувати користувачам про необхідність випити воду. Наразі нагадування реалізоване на базовому рівні, але можна вдосконалити цю систему, зробивши її більш персоналізованою. Наприклад, нагадування можуть бути налаштовані відповідно до індивідуального режиму дня користувача, враховуючи час пробудження, робочий графік, тренування та інші активності.

Система досягнень. Введення системи досягнень може стати потужним мотиваційним інструментом для користувачів. Вони зможуть отримувати віртуальні нагороди за досягнення певних цілей, таких як регулярне вживання води протягом тижня або досягнення щоденного рекомендованого рівня

споживання рідини. Це може стимулювати користувачів більш відповідально підходити до підтримання водного балансу.

Темний режим інтерфейсу. Впровадження темного режиму інтерфейсу покращить користувацький досвід, особливо для тих, хто користується застосунком в умовах низького освітлення. Темний режим зменшує навантаження на очі та дозволяє економити заряд батареї на мобільних пристроях, що особливо актуально для користувачів, які активно користуються застосунком протягом дня.

Синхронізація із фітнес-застосунками. Однією з ключових функцій, яка може значно підвищити привабливість застосунку, є можливість синхронізації з популярними фітнес-застосунками, такими як Google Fit, Apple Health, Fitbit тощо. Це дозволить користувачам автоматично отримувати дані про їхню фізичну активність, що може впливати на їхні потреби у споживанні рідини. Наприклад, після інтенсивного тренування застосунок може рекомендувати збільшити споживання води для відновлення водного балансу.

Перерахування денної норми в залежності від погодних умов. Ще однією важливою функцією, яку можна додати, є перерахування денної норми споживання води залежно від погодних умов. У спекотні дні потреба організму у воді зростає через підвищене потовиділення, тому корисно, щоб застосунок автоматично збільшував рекомендовану денну норму рідини. В прохолодні дні ця норма може бути зменшена. Реалізація цієї функції потребує інтеграції з сервісами прогнозу погоди та можливості налаштування рекомендацій для користувачів.

Усі ці додаткові функції мають потенціал зробити вебзастосунок більш універсальним, зручним та корисним для широкого кола користувачів. Реалізація цих можливостей вимагатиме додаткових зусиль, проте перспективи покращення користувацького досвіду роблять ці зусилля виправданими.

# ВИСНОВКИ

У рамках кваліфікаційної роботи було проведено аналіз існуючих вебзастосунків щоденного обліку води, виявлено їх сильні та слабкі сторони. Спроєктовано вебзастосунок, визначено його структуру, компоненти та взаємодію між ними. Реалізовано вебзастосунок обліку щоденного вживання води за допомогою середовища VS Code, мови програмування JavaScript, БД MongoDB, платформи Node.js, фреймворка Express, бібліотек React, Mongoose, CORS, JWT, Redux, Axios, Formik. Також було визначено перспективи вебзастосунку щоденного обліку вживання води.

React підходить для створення складних, інтерактивних вебзастосунків, забезпечуючи високу продуктивність та зручність у розробці завдяки компонентній архітектурі та віртуальному DOM.

Node.js є потужним інструментом для розробки сучасних вебзастосунків, особливо тих, які потребують обробки великої кількості одночасних запитів або реального часу.

MongoDB є потужним інструментом для зберігання та обробки великих обсягів даних з високою гнучкістю та продуктивністю, що робить її популярною для створення сучасних застосунків, особливо тих, які вимагають масштабованості та швидкої обробки даних.

В результаті чого було отримано вебсайт з інтуїтивним інтерфейсом для введення даних про споживання води, з відображенням статистики та аналітики.

В перспективах вебзастосунку щоденного обліку вживання води можна реалізувати різні види рідин з їх відсотком водної біодоступності, систему оповіщень, систему досягнень, темний режим інтерфейсу, синхронізацію із фітнес-застосунками, а також функцію перерахунку денної норми в залежності від погодних умов.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Творошенко, І.С. (2021). Технології прийняття рішень в інформаційних системах: навч. посібник. Харків: ХНУРЕ.

2. Гороховатський, В.О., & Творошенко, І.С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

3. Iryna Vechirska, Oleg Kobylin, Stepan Prokopiev, Anna Vechirska, Maksym Kucherenko. Building a logical network for solving the problem of car rental by means algebra of finite predicates// Computer Systems and Information Technologies. – Хмельницький, 2022. - № 2. – с. 78-87.

4. Застосунок «Водокотик». ULR: https://apps.apple.com/ua/app/%D0%B2%D0%BE%D0%B4%D0%BE%D0%BA%D0%BE%D1%82%D0%B8%D0%BA-%D0%BD%D0%B0%D0%B3%D0%B0%D0%B4%D0%B0%D1%82%D0%B8-%D0%BF%D0%B8%D1%82%D0%B8-%D0%B2%D0%BE%D0%B4%D1%83/id1579056677?l=uk (дата звернення 14.05. 2024).

5. Застосунок «Water Time». URL: https://apps.apple.com/ua/app/water-time-drink-reminder/id1208916325?l=uk (дата звернення 14.05.2024).

6. Застосунок «My Water». URL: https://apps.apple.com/ua/app/my-water-daily-drink-tracker/id964748094?l=uk (дата звернення 14.05.2024).

7. React Architecture Patterns. URL: https://www.etatvasoft.com/blog/react-architecture-patterns/ (дата звернення 14.05.2024).

8. Single-page application vs. multiple-page application. URL: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58 (дата звернення 14.05.2024).

9. What is a REST API? URL: https://www.ibm.com/topics/rest-apis (дата звернення 14.05.2024).

10. Посібник React для початківців 1: Знайомство з React. URL: https://www.w3ctech.com/topic/1877 (дата звернення 14.05.2024).

11. Describing the UI. URL: https://react.dev/learn/describing-the-ui (дата звернення 18.05.2024).

12. What is Redux? URL: https://redux.js.org/tutorials/essentials/part-1-overview-concepts#what-is-redux (дата звернення 18.05.2024).

13. Що таке Axios? URL: https://axios-http.com/uk/docs/intro (дата звернення 18.05.2024).

14. XMLHttpRequest API. URL: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest (дата звернення 18.05.2024).

15. AJAX - Send a Request To a Server. URL: https://www.w3schools.com/xml/ajax_xmlhttprequest_send.asp (дата звернення 19.05.2024).

16. Introduction to React.js: Building Interactive User Interfaces. URL: https://thehumblecoder.medium.com/introduction-to-react-js-building-interactive-user-interfaces-647bd9ce1af3 (дата звернення 19.05.2024).

17. Method for building of logical data transform in the problem of establishing links between the objects in intellectual telecommunication systems/ Gorokhovatskyi V.A., Vechirska, I.D., Chetverikov, G.G. // Telecommunications and engineering. – 2016. – Issue 18. - Vol. 75.- Pp. 1645-1655.

18. Вечірська І.Д. Побудова логічної мережі для діагностики та управління надзвичайними ситуаціями/І. Д. Вечірська, І.Е. Гончаров, Т.М. Хамітов //Біоніка інтелекту. – Харків, 2015. – № 2 (85).- с. 41-51.

19. Вечірська І.Д., Вечірська А.Д. Проблематика застосування засобів алгебри скінченних предикатів на теренах системного аналізу // Proceedings of XI International scientific and practical conference «Fundamental and applied research in the modern world». – Boston, USA. 9-11 June 2021 – С. 275-277.

20. Вечірська І.Д., Вечірська А.Д. Аналіз застосування інструментарію алгебри предикатів для оцінення якості складної системи /II International Scientific and Practical Conference «Ricerche scientifiche e metodi della loro realizzazione: esperienza mondiale e realtà domestiche», *26.11.2021 . Відень, AUT/ Band2. P.48-50.*

21. Vechirska A.D., Shyrokorad K.A., Vechirska I.D. Visual modeling of child registration to kindergarten process.//Technologies and strategies for the implementation of scientific achievements: collection of scientific papers «SCIENTIA» with Proceedings of the I International Scientific and Theoretical Conference (Vol. 2), May 27, 2022. Stockholm, Kingdom of Sweden: European Scientific Platform. – P.73-76.

22. Vechirska A.D., Shyrokorad K.A., Supervisor: Vechirska I.D. Visual modeling of purchase process management in the electronics online store// Діджиталізація науки як виклик сьогодення: матеріали III Міжнародної студентської наукової конференції, м. Львів, 3 червня, 2022 рік / ГО «Молодіжна наукова ліга». — Вінниця: ГО «Європейська наукова платформа», 2022. —с.189-192.

23. Vechirska I.D., Vechirska A.D., Shyrokorad K.A. Solving the problem of choosing the bestassembly plan for software deployment in the cloud environment using the analytic hierarchy process//Розвиток суспільства та науки в умовах цифрової трансформації:матеріали IV Міжнародної студентської наукової конференції,м.Дніпро, 23 червня, 2023 рік / ГО «Молодіжна наукова ліга».— Вінниця: ГО «Європейська наукова платформа» P. 126-129.

24. Get learning React. Introducing React. URL: https://www.oreilly.com/library/view/learning-react-a/9780134843582/ch01.xhtml (дата звернення 18.05.2024).

25. Посібник React для початківців 1: Знайомство з React. URL: https://www.w3ctech.com/topic/1877 (дата звернення 19.05.2024).

26. Introduction to Node.js. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (дата звернення 19.05.2024).

27. Vechirska I.D., Vechirska A.D., Shyrokorad K.A. Аналіз проблем програмного забезпечення дронів, що застосовуються у розумних містах//Сучасні аспекти та перспективні напрямки розвитку науки:матеріали V Міжнародної студентської наукової конференції, м.Житомир, 9 червня, 2023 рік / ГО «Молодіжна наукова ліга».— Вінниця: ГО «Європейська наукова платформа», 2023 - 148-149 ст.

28. Chetverikov, G., Puzik O., Vechirska, I. /Multiple-valued structures of inteiiectual systems // XI International Scientific and Technical Conference Computer Science and Information Technologies, Conference Proceedings. -2016.- Pp. 204-207.

29. Introduction to JSON Web Tokens. URL: https://jwt.io/introduction (дата звернення 18.05.2024).

30. Single-page application vs. multiple-page application. URL: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58 (дата звернення 18.05.2024).