Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота
за освітнім ступенем “бакалавр”

на тему:

ВЕБ-ДОДАТОК ДЛЯ ПЛАНУВАННЯ ДНЯ ТА ВАЖЛИВИХ ПОДІЙ МІСЯЦЯ

Виконав:
здобувач IV курсу
групи КН-41
спеціальності 122 “Комп'ютерні науки”
Косарєв Дмитро Сергійович

Науковий керівник:
кандидат технічних наук,
доцент Бабич Степанія Михайлівна

Рівне - 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

СУБД – Система управління базами даних.

JSON – Текстовий формат обміну даними.

Node.js – програмне середовище виконання мови *JavaScript.*

Bootstrap – це безкоштовний набір інструментів з відкритим кодом, призначений для створення вебсайтів.

ВСТУП

Сучасний світ відрізняється швидким темпом життя, постійними змінами і великою кількістю завдань та важливих подій, які ми повинні враховувати і планувати у нашому житті. Перед людьми стоїть виклик ефективно організувати свій час, планувати завдання і не пропускати важливі події.

У зв'язку з цим, веб-додатки планування стали невід'ємною частиною повсякденного життя, допомагаючи раціонально використовувати час, керувати завданнями і нагадувати про важливі події. Однак, багато з існуючих веб-додатків планування недостатньо функціональні або не забезпечують високу зручність використання.

Тому обґрунтованою є **актуальність** розробки нового веб-додатка планування, який відповідає потребам сучасних користувачів і забезпечує їм зручність, ефективність та надійність у плануванні їхнього дня та важливих подій місяця.

Основні переваги такого веб-додатка наступні:

- організація та структурованість завдань: розроблений веб-додаток надаватиме користувачам можливість ефективно організовувати свої завдання, створювати списки, надавати їм перевагу та відстежувати виконання;
- гнучкість та персоналізація: додаток забезпечуватиме можливість налаштування індивідуальних пріоритетів, нагадувань та повідомлень, що дозволить користувачам адаптувати його до своїх потреб;
- інтеграція з календарем та подіями: веб-додаток буде забезпечувати синхронізацію з календарем та можливість додавання важливих подій, які будуть відображатися у гнучкому та зручному календарі;
- доступність та зручність: додаток має бути розроблено як веб-додаток, що дозволить його використання на різних пристроях (комп'ютерах, планшетах, смартфонах) та надасть користувачам можливість з легкістю отримувати доступ до своїх завдань та подій з будь-якого місця та в будь-який час.

Таким чином, розробка веб-додатка є актуальною задачею, що вирішує проблему неефективного управління часом. Розроблений додаток дозволить користувачам зосередитися на важливих справах та досягати своїх цілей з більшою ефективністю і зручністю.

Зважаючи на вищевикладене, **метою** даної кваліфікаційної роботи є розробка і реалізація веб-додатку планування дня та важливих подій місяця, який відповідає потребам сучасних користувачів і забезпечує їм зручність, ефективність та надійність у плануванні їхнього часу та завдань.

Щоб досягти мети розробки веб-додатка необхідно виконати наступні **завдання**:

- дослідити поняття веб-додатку;
- аналіз існуючих веб-додатків планування та їх функціональних можливостей;
- визначення основних вимог та потреб користувачів у веб-додатку планування дня та важливих подій місяця;
- розробка архітектури та інтерфейсу веб-додатку з урахуванням встановлених вимог та потреб користувачів;
- реалізація веб-додатку планування дня та важливих подій місяця;
- оцінка ефективності, зручності використання та задоволення користувачів веб-додатком.

Об'єкт дослідження - розробка веб-додатку для планування дня та важливих подій місяця.

Предмет дослідження - функціональні можливості та технології, використовувані при створенні веб-додатку для планування дня та важливих подій місяця.

Методи дослідження:

- вивчення навчальної та наукової літератури з означеної теми;
- структурно-функціональні та об’єктно-орієнтовані аналіз;
- моделювання процесів планування в складних системах.

Практичне значення дослідження. Веб додаток календар дозволяє зручно та ефективно планувати свій час.

Апробація і впровадження результатів дослідження. Основні результати роботи доповідалися на звітній науковій конференції викладачів, співробітників і здобувачів вищої освіти Рівненського державного гуманітарного університету за 2024 рік (16 травня 2024 р.).

Структура роботи. Дипломна робота складається зі вступу, трьох розділів, висновків, переліку використаних джерел та додатків. Загальний обсяг роботи становить 46 сторінки. Список використаних джерел включає 15 найменувань.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ВЕБ-ДОДАТКІВ

1.1 Поняття веб-додатку

Веб-додаток – це програмне забезпечення або програма, яку можна відкрити за допомогою будь-якого браузера. Він знаходиться на веб-сервері і забезпечує взаємодію користувача з сервером за допомогою мережевого протоколу *HTTP*. Користувачі отримують доступ до веб-додатку, використовуючи *URL*-адресу (веб-адресу), яку вони вводять у свій веб-браузер.

Веб-додаток базується на клієнт-серверній архітектурі (рис. 1.1), де веб-браузер виступає як клієнт, а веб-сервер - як сервер. Клієнтська сторона включає *HTML*-сторінки, *CSS*-стилі та *JavaScript*-скрипти, які виконуються безпосередньо на боку користувача. Це дозволяє створювати інтерактивні та динамічні веб-інтерфейси. Серверна сторона веб-додатка включає веб-сервер, бази даних та серверні скрипти. Веб-сервер обробляє запити від клієнтської сторони і надсилає відповіді. Бази даних використовуються для збереження та управління даними, що використовуються в додатку. Серверні скрипти забезпечують обробку запитів, взаємодію з базою даних та логіку додатку.

Клієнт і сервер взаємодіють за допомогою мережевих запитів та відповідей, що дозволяє обмінюватися даними та виконувати різні операції. Користувач створює запит, який повинен буде відправитися на веб-сервер за допомогою запиту в інтернет-браузері, або в самому веб-додатку. Після того як сервер отримав запит, він надходить до бази даних для формування відповіді на запит користувача. Фінальним етапом буде відправка необхідних даних, а потім і відображення їх користувачеві. І так повторюється знову і знову, поки користувач створює запити у веб-додатку.

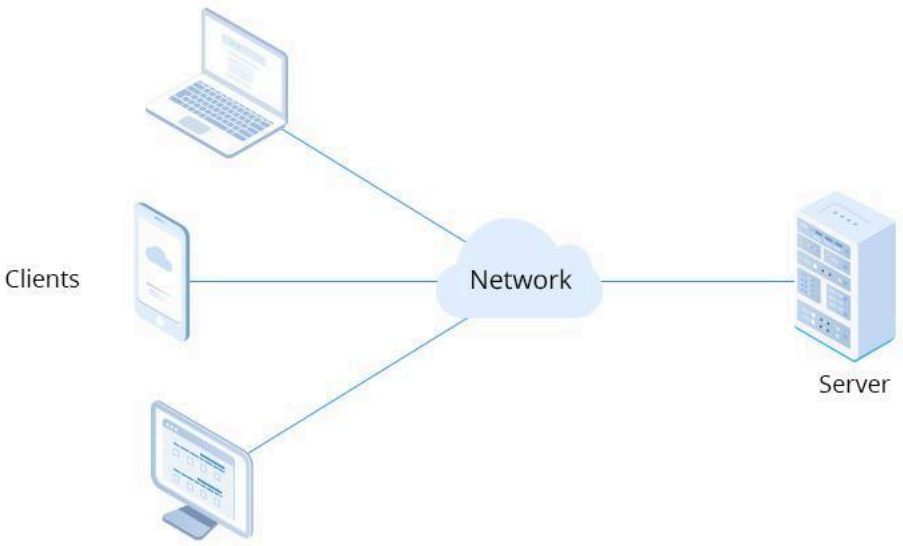


Рисунок 1.1. Архітектура клієнт-сервер

Веб-додатки використовують протокол *HTTP* для взаємодії між клієнтом та сервером. Клієнт надсилає *HTTP*-запити на сервер, який обробляє ці запити і надсилає назад *HTTP*-відповіді з необхідною інформацією. Це може включати отримання статичного контенту, виконання запитів до бази даних, виконання операцій над даними тощо.

Ключові переваги веб-додатків:

- веб-додаток можна використовувати в будь-якій операційній системі (*Linux*, *Mac*, *Windows*), оскільки всі вони підтримують сучасні браузери;
- оскільки веб-програми використовують той самий код, що й настільні програми, їх легше підтримувати;
- додаток легше програмувати, тому що не вимагає багато роботи з елементами ПК (ядро, процесор, відеокарта);
- на відміну від мобільних додатків, веб-додатки не вимагають певної платформи для публікації свого додатка;

- веб-додатки є більш економічним вибором для будь-якого бізнесу. Тому що веб-програми не потребують підписки чи купівлі ліцензій, а доступні як послуга *SaaS*, що набагато дешевше.

Web-додатки створюють з різною метою, під різні задачі. Розглянемо основні їх типи:

- *E-commerce* системи. Веб-додаток для електронної комерції – варіант для інтернет-магазинів, щоб просувати свої товари чи послуги;
- *CRM*-системи. Веб-додатки *CRM* містять інструменти для покращення роботи з клієнтами. Вони дозволяють вам автоматизувати продажі, починаючи від пошуку потенційних клієнтів і закінчуючи прискоренням обробки повторних замовлень;
- *ERP*-системи. Веб-програми *ERP* дозволяють реалізувати комплексне керування підприємством у найпростішому форматі – зі смартфона чи планшета, навіть без мобільного додатка. Такі рішення дозволяють впроваджувати нові бізнес-моделі, приймати оперативні бізнес-рішення, контролювати фінанси тощо;
- корпоративні портали. Корпоративні портали – це комплексні платформи, які надають користувачам доступ до корпоративних даних та найрізноманітнішого сервісу. Вони дозволяють працювати одразу з декількома додатками компанії (*CRM*, *ERP*, поштою, чатом тощо), використовуючи один інтерфейс. Кожен працівник має окремий доступ, тому має пройти автентифікацію.

1.2. Огляд найбільш популярних веб-додатків планування

Існує широкий вибір веб-додатків для планування дня та організації важливих подій. Кожна з цих програм має власні особливості та функції, які допомагають

користувачам керувати своїм часом і ефективно працювати. Розглянемо кілька популярних веб-програм, щоб спланувати свій день.

Google Calendar (рис. 1.2) є одним з найпопулярніших і найвідоміших веб-додатків планування, який пропонує широкий набір функцій та зручний інтерфейс.

Користувачі можуть перемикатися між кількома переглядами і перемикати відображення окремих календарів. Таким чином ви зможете зосередитися на майбутніх завданнях, не забиваючи, наприклад, дні народження чи свята.

З точки зору інтерфейсу *Google Calendar* має простий і зрозумілий дизайн, з легкою навігацією та доступом до основних функцій. Події відображаються у вигляді кольорових блоків у календарі, який користувачі можуть переглядати за днями, тижнями, місяцями та розкладом.

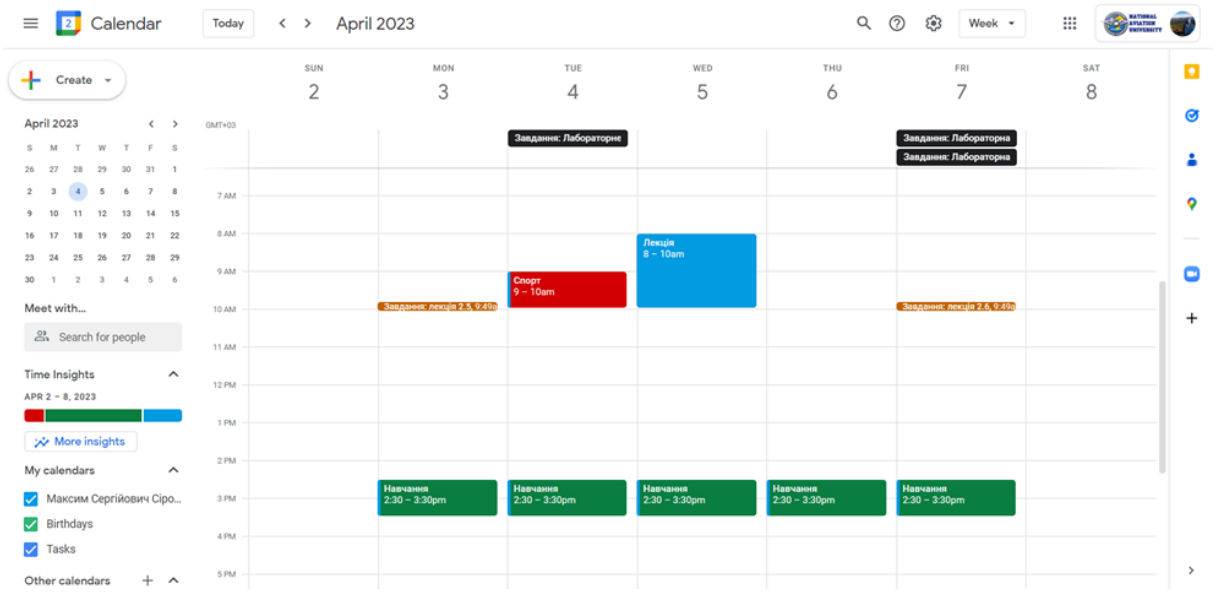


Рисунок 1.2. Інтерфейс *Google Calendar*

Серед переваг *Google Calendar* варто виділити:

- інтеграцію з іншими сервісами *Google*, такими як *Gmail*, *Google Drive*, *Google Meet* і *Google Tasks*;
- можливість ділитися календарем з колегами, друзями або сім'єю;

– автоматичну синхронізацію подій між пристроями і доступ до календаря з будь-якого пристрою з підключенням до Інтернету.

До недоліків можна віднести обмежені можливості настройки вигляду та кастомізації інтерфейсу.

Microsoft Outlook Calendar — це веб-програма для планування подій, зустрічей і керування часом (рис. 1.3). Інтерфейс календаря *Microsoft Outlook* має професійний та зручний дизайн із панелями навігації та доступом до основних функцій. Події відображаються в календарі різними кольорами для легкої ідентифікації. Користувачі можуть переглядати календар за днями, тижнями, місяцями та розкладом.

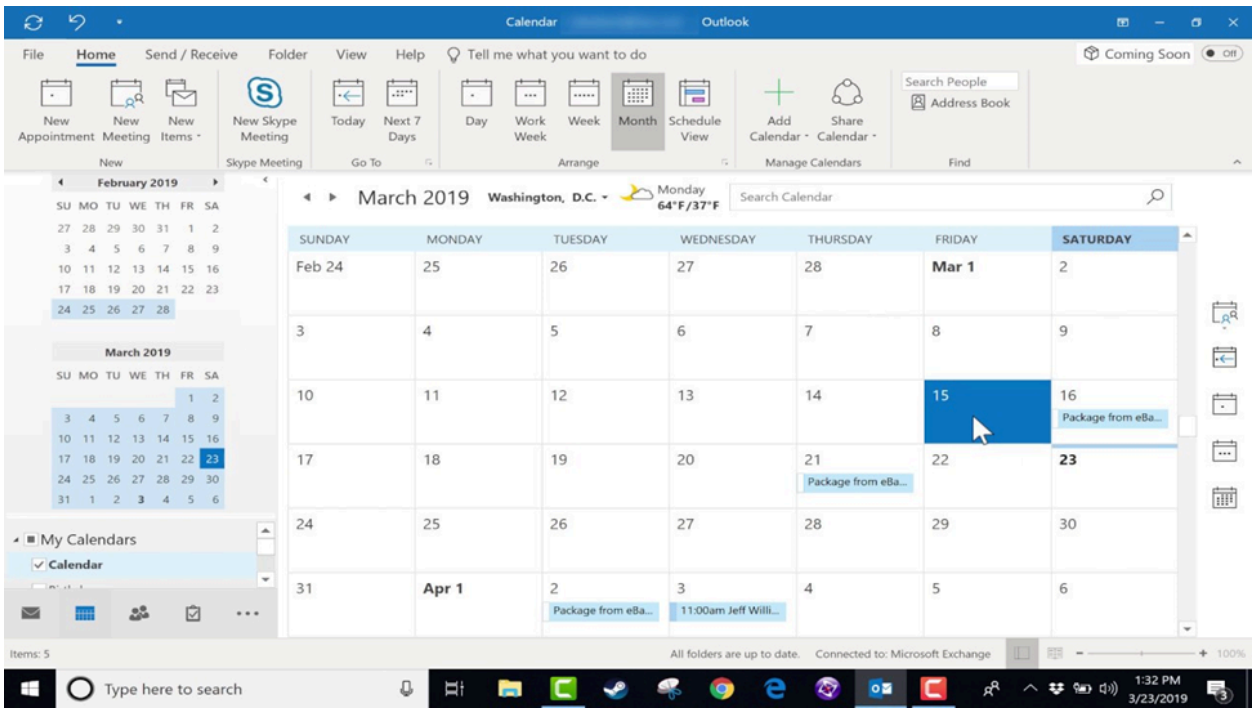


Рисунок 1.3. Інтерфейс *Microsoft Outlook Calendar*

Переваги *Microsoft Outlook*:

– інтеграція з *Microsoft Outlook*, що дозволяє керувати електронною поштою, контактами та завданнями в одному інтерфейсі;

– можливість спільно працювати над календарями та ділитися доступом до подій з колегами;

- автоматична синхронізація з іншими пристроями та програмами *Microsoft* забезпечує постійний доступ до вашого календаря.

Недоліки календаря *Microsoft Outlook*:

- деякі функції можуть бути обмеженими або доступними лише за підпискою на платформу *Microsoft 365*;
- інтерфейс може бути менш інтуїтивно зрозумілим порівняно з іншими календарними додатками.

Any.do - це веб-додаток, який надає користувачам можливість планувати свій день, організовувати завдання та керувати своїм часом (рис.1.4). *Any.do* має сучасний та інтуїтивно зрозумілий інтерфейс. Він простий у використанні, має чистий дизайн та зручні елементи керування, що допомагають легко створювати, редагувати та організовувати завдання. Інтерфейс також може пристосовуватися до особистих уподобань користувача.

З його допомогою можна створювати, організовувати та відстежувати свої завдання і події, встановлювати нагадування, додавати коментарі та вкладати файли до задач, а також спільно працювати над проектами з колегами. Додаток також має інтеграцію з календарем, можливість створення списків, планування повторюваних завдань і пріоритетів, а також синхронізується між різними платформами, що дозволяє вам отримувати доступ до вашого списку завдань з будь-якого пристрою.

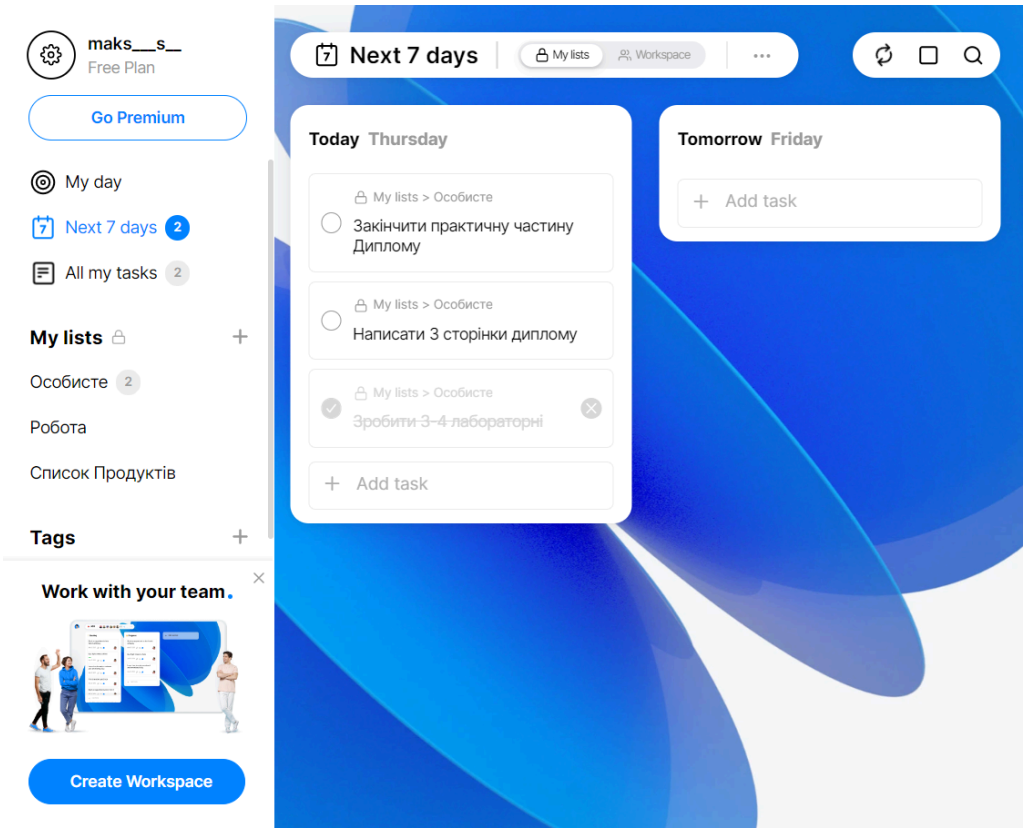


Рисунок 1.4. Інтерфейс *Any.do*

Серед переваг присутні:

- кросплатформений доступ, що дозволяє вам мати доступ до своїх завдань з будь-якого пристрою;
- інтеграція з іншими сервісами;
- простота використання;
- керування завданнями в реальному часі: користувач може оновлювати, редагувати та міняти статуси завдань у реальному часі, що сприяє кращій організації та продуктивності.

До недоліків можна віднести обмежені можливості в безкоштовні версії додатку.

1.3. Висновки до розділу

У першому розділі було розглянуто поняття веб–додатку, що дозволило зрозуміти його особливості та характеристики. Було досліджено основні аспекти додатків, такі як архітектура, функціональність, можливості та переваги порівняно з іншими типами програмного забезпечення.

В ході аналізу сучасних веб-додатків для планування дня та важливих подій місяця були вивчені різноманітні аспекти, пов’язані з цією темою. Було проведено детальний огляд найбільш популярних додатків для планування, зокрема *Google Calendar, Microsoft Outlook Calendar* і *Any.do*. Кожен з них має свої унікальні особливості і функціональні можливості, які були детально проаналізовані. Завершуючи аналіз, було визначено загальні переваги та недоліки сучасних веб-додатків для планування.

Загалом, аналіз сучасних веб-додатків планування виявив широкий спектр функцій та можливостей, які допомагають ефективно організовувати робочий та особистий час. Вибір певного веб-додатку залежить від індивідуальних потреб користувача, його пріоритетів та вимог до функціональності.

Додатки для планування часу користуються значною популярністю серед користувачів. В сучасному світі, де людям потрібно керувати багатьма завданнями та подіями, такі додатки стали необхідними інструментами для ефективного управління часом. Це й не дивно адже такі додатки надають зручність та організацію, дозволяючи користувачам легко створювати, редагувати та відстежувати свої завдання та події.

РОЗДІЛ 2. ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ ВЕБ-ДОДАТКІВ

2.1. Тенденції розробки веб-додатків

В останні роки популярність Інтернету та мобільних пристроїв зросла. Мобільні пристрої стали важливим компонентом повсякденного життя людей, оскільки вони все більше взаємопов’язані. Це надає величезну можливість створювати веб-програми, які функціонують на різних платформах і доступні в будь-який час з будь-якого місця.

Веб-розробка, галузь, яка постійно розвивається, надає розробникам численні можливості для створення ефектних та винахідливих веб-додатків за допомогою нових технологій, фреймворків та інструментів. З розвитком і прогресом протоколів і стандартів веб-програми стають безпечнішими, швидшими та краще адаптованими.

Сучасні користувачі багато чого очікують від веб-програм — безпечне середовище, сумісність із різними пристроями, швидкість і, звісно, зручний інтерфейс, у якому легко орієнтуватися. Вимоги користувачів постійно змінюються, тому розробники повинні бути готові адаптуватися до нових вимог та надавати інноваційні функції та можливості в своїх додатках.

Завдяки постійним змінам у технологіях та вимогах користувачів, розробники веб-додатків повинні бути готові швидко впроваджувати нові функції та технології у своїх додатках. Це може включати використання нових бібліотек, фреймворків, розширень або методів розробки. Розробники повинні бути в курсі останніх трендів та знати, як використовувати їх для покращення своїх додатків та задоволення потреб користувачів.

Загалом, сучасні тенденції розробки веб-додатків визначаються зростанням використання Інтернету та мобільних пристроїв, технологічними змінами, вимогами користувачів та швидким впровадженням нових функцій та технологій. Розробники повинні бути готові адаптуватися до цих змін, використовувати сучасні інструменти та практики, щоб створювати інноваційні та ефективні веб-додатки, які задовольняють потреби користувачів у сучасному цифровому світі.

2.2. Клієнтська частина

У веб-розробці «клієнтська частина» стосується всього у веб-додатку, що відображається або відбувається на клієнті (пристрої кінцевого користувача). Це включає те, що бачить користувач, як-от текст, зображення та іншу частину інтерфейсу користувача, а також будь-які дії, які виконує програма в браузері.

Мови розмітки, такі як *HTML* і *CSS*, інтерпретуються браузером на стороні клієнта. Крім того, багато сучасних розробників включають процеси на стороні клієнта в архітектуру своїх програм і відходять від виконання всього на стороні сервера; бізнес-логіка для динамічних веб-сторінок, наприклад, зазвичай працює на стороні клієнта в сучасній веб-програмі. Процеси на стороні клієнта майже завжди написані на *JavaScript*. Розглянемо ці технології:

- *HTML* мова розмітки гіпертексту, використовується для структурованого впорядкування сегментів, цитат, гіперпосилань, заголовків і абзаців;
- *CSS* (*Cascading Style Sheets* - каскадні таблиці стилів) - одна з базових технологій у сучасному Інтернеті. *CSS*-код - це список інструкцій для браузера, як і де відображати елементи веб-сторінки. Під “елементами” зазвичай маються на увазі теги *HTML* і їх вміст. Основна ідея *CSS* в тому, щоб відокремити дизайн документа від його вмісту. *CSS* відповідає за оформлення і зовнішній вигляд, а *HTML* - за зміст і логічну структуру документа;

- *JavaScript* — це інтерпретована високорівнева мова програмування, яка підтримує імперативний, функціональний і подієво-орієнтовані стилі, та використовується для розробки динамічних веб-додатків та скриптів на стороні клієнта. Він дозволяє створювати динамічні елементи на веб-сторінках, контролювати події, змінювати вміст сторінки, взаємодіяти з користувачем та здійснювати асинхронні запити до сервера.

  JavaScript також може використовуватися на стороні сервера за допомогою платформи *Node.js*. В цьому випадку він використовується для розробки серверних додатків, *API* та інших серверних рішень. *Node.js* дозволяє виконувати *JavaScript* поза браузером, що розширює можливості мови та дозволяє використовувати її для розробки повноцінних веб-додатків.

2.3. Популярні фреймворки та бібліотеки

Фреймворк - це програмне середовище, яке спрощує та прискорює процес створення програмного забезпечення. Використовуючи фреймворки, розробники можуть сфокусуватись на реалізації логіки, специфічної для свого продукту, і зосередитись на самому програмуванні. Існують завдання, спільні для різних веб-сайтів, веб-програм та інших продуктів. Це маршрутизація, взаємодія з базою даних, автентифікація користувачів, підтримка сеансів, захист від веб-атак, кешування, відокремлення логіки від представлення та багато іншого. Створювати код для виконання цих завдань щоразу – це невиправдана трата часу та зусиль. Код для виконання таких завдань реалізований у фреймворках.

Бібліотека в програмуванні – це набір зумовлених функцій, класів і ресурсів, які можна використовувати для вирішення завдань із певної області. Бувають бібліотеки для роботи з датою та часом, випадковими числами, файловою системою, *HTTP*-запитами.

Бібліотека не визначає структуру програми. Під час роботи з нею користувач вирішує, що і коли викликати. Натомість фреймворк визначає архітектуру програми

та забезпечує взаємодію між її компонентами. Він містить різні бібліотеки та використовує їх для створення каркасу програми.

- *React* – це декларативна, ефективна і гнучка *JavaScript*-бібліотека, призначена для створення інтерфейсів користувача. Вона дозволяє компонувати складні інтерфейси з невеликих, окремих частин коду – “компонентів”, які відповідають за відображення конкретних частин інтерфейсу. Кожен компонент має свою внутрішню логіку та може бути повторно використаний в різних частинах додатку;

- *Express.js* – швидкий, гнучкий, мінімалістичний фреймворк для веб-додатків, побудованих на *Node.js*. Розробники можуть зручно створювати веб-додатки та *API*, використовуючи нескладний інтерфейс і потужні можливості *Express.js*. Також фреймворк підтримує *middleware* функції. Ці функції допомагають керувати різними завданнями, такими як обробка форм, обробка помилок і автентифікація шляхом виконання до або після обробки запитів.

  Його простота, швидкість і гнучкість роблять його ідеальним вибором для створення програм на стороні сервера, не занурюючись у технічні деталі. Завдяки мінімалістичному дизайну Express.js пропонує розробникам широку свободу розширювати свої можливості різними модулями та бібліотеками;

- *Angular* – фронтенд-фреймворк з відкритим кодом, написаний на *TypeScript*, який розробляють *Google*, під керівництвом *Angular Team*. Однією з основних особливостей *Angular* є його компонентна архітектура. Він використовує концепцію компонентів, які представляють окремі частини користувацького інтерфейсу, такі як кнопки, форми, таблиці і т.д. Кожен компонент має свою власну логіку та представлення, що сприяє модульності та повторному використанню коду.

2.4. Серверна частина

Серверна частина веб-додатку відповідає за обробку запитів користувачів, взаємодію з базою даних, виконання бізнес-логіки та надання відповідей клієнтській стороні. Це частина додатку, яка виконується на сервері і відповідає за обробку та збереження даних, а також забезпечує взаємодію з клієнтською стороною через мережу.

Існує багато платформ та мов програмування для реалізації серверної частини:

- *PHP* – скриптова мова програмування загального призначення, інтенсивно застосовується для розробки веб-додатків. В даний час підтримується переважною більшістю хостинг-провайдерів і є одним з лідерів серед мов програмування, що застосовуються для створення динамічних веб-сайтів;
- *Python* є популярною мовою програмування, яка широко використовується для серверної розробки веб-додатків. Вона має простий синтаксис та багатий набір бібліотек, що полегшує розробку. Flask та Django є двома популярними веб-фреймворками Python, які надають потужні можливості для створення серверної частини веб-додатків;
- *Java* – це мова програмування загального призначення, для розробки програмного забезпечення, вебсервісів, ігор і застосунків. Java є однією з найпопулярніших мов програмування у світі завдяки нескладному синтаксису, гнучкості, безпеці, портативності та масштабованості. До відчутних переваг можна віднести: кросплатформеність, надійність, та продуктивність;
- *Node.js* – програмне середовище виконання мови *JavaScript*, в якому можна запускати код *JS* замість браузера і працювати з ним. В основі *Node.js* лежить движок *V8* від *Google*. *Node.js* дозволяє *javascript* підключатися до різних пристроїв введення-виведення (камерам, мікрофону тощо), а ще до

бібліотек на різних мовах програмування, розширюючи можливості програми.

Найчастіше *Node.js* застосовують як *web* сервер. Вміння розподіляти ресурси сервера, в залежності від дії та бездіяльності, роблять *Node* унікальним рішенням.

Web-додаток, побудований з *Node.js*, буде легким, продуктивним і не вимогливим до ресурсів. Його серверна частина зможе обслуговувати величезну кількість звернень зі стабільною ефективністю.

2.5. Бази даних у веб-додатках

База даних (БД) – впорядкований набір логічно взаємопов'язаних даних, що призначений для задоволення інформаційних потреб користувачів. Головним завданням БД є гарантоване збереження значних обсягів інформації та надання доступу до неї користувачеві або ж прикладній програмі. Таким чином БД складається з двох частин : збереженої інформації та системи управління нею.

У веб-додатках бази даних є важливою складовою, оскільки вони дозволяють зберігати та управляти великим обсягом даних, забезпечують безпеку, ефективну роботу та дозволяють здійснювати складні операції обробки даних. Вони використовуються для зберігання інформації про користувачів, контент, статистику, транзакції та багато іншого, що дозволяє веб-застосункам працювати ефективно та надавати користувачам потрібну функціональність.

У сучасному світі існує велика кількість різних видів баз даних, всі вони мають певні схожості та водночас їхня поведінка та особливості можуть відрізнятися. Розглянемо деякі види баз даних та приклади що їм відповідають.

2.5.1 Реляційна база даних

Реляційна база даних – це структурована колекція даних, яка організована у вигляді таблиць. Кожна таблиця складається з рядків і стовпців, де рядки

представляють конкретні записи, а стовпці визначають типи даних, які можуть бути збережені. Реляційна база даних використовує спеціальні зв'язки між таблицями, щоб візуалізувати і зберігати залежності між даними.

Встановлення зв'язків між таблицями в базі даних здійснюється за допомогою використання ключів, які ефективно зв'язують і впорядковують дані між різними таблицями. Безпечне та надійне керування даними забезпечується реляційними базами даних, які пропонують *ACID*-властивості, що складаються з авторизації, атомарності, узгодженості та ізоляції. Один із способів взаємодії з реляційною базою даних – це використання мови структуризованих запитів *SQL (Structured Query Language)*. *SQL* надає набір команд, які можна використовувати для створення, модифікації і вибірки даних у реляційних базах даних.

Прикладом реляційної бази даних є *MySQL*. Це одна з найпопулярніших та широко використовувана відкрита реляційна база даних. До переваг через які ця база даних є такою широковживаною можна віднести простоту, продуктивність, стандартизація та безпека.

2.5.2 Нереляційна база даних

Нереляційна база даних (*NoSQL*) – зберігає дані без чітких зв'язків між собою та без чіткої структури. Замість структурованих таблиць, всередині бази знаходиться безліч різнорідних документів. *NoSQL* бази не підтримують *SQL* запити.

На відміну від реляційних, у нереляційних базах даних схема даних є динамічною та може змінюватись у будь який момент часу. До даних складніше отримати доступ. Проте такі СУБД відрізняються швидкістю та продуктивністю. Фізичні об'єкти у *NoSQL* зазвичай можна зберігати прямо у тому вигляді, у якому з ними потім працює додаток.

Бази даних *NoSQL* підходять для зберігання великих об'ємів неструктурованої інформації, а також для швидкої розробки та тестування гіпотез. У них можна зберігати дані будь яких типів та додавати нові в процесі роботи.

Прикладом нереляційної бази даних є *MongoDB*. Вона використовує модель документів, де дані зберігаються у вигляді *JSON*-подібних документів. *MongoDB*

широко використовується для розробки веб-додатків, аналітики даних, систем керування вмістом, додатків машинного навчання та багатьох інших випадків використання, де потрібна гнучкість та швидкість обробки даних.

2.5.3 Ієрархічна база даних

Ієрархічна модель даних – це модель даних, де використовується представлення бази даних у вигляді деревовидної (ієрархічної) структури, що складається з об'єктів (даних) різних рівнів (рис. 2.1).

Між об'єктами існують зв'язки, кожен об'єкт може включати в себе кілька об'єктів більш низького рівня. Такі об'єкти перебувають у відношенні предка до нащадку, при цьому можлива ситуація, коли об'єкт-предок не має нащадків або має їх декілька, тоді як в об'єкта-нащадка обов'язково тільки один предок.

До переваг можна віднести: розробку низькорівневих інструментів керування даними у зовнішньому сховищі, здатність створювати ефективні прикладні системи, економне використання пам'яті.

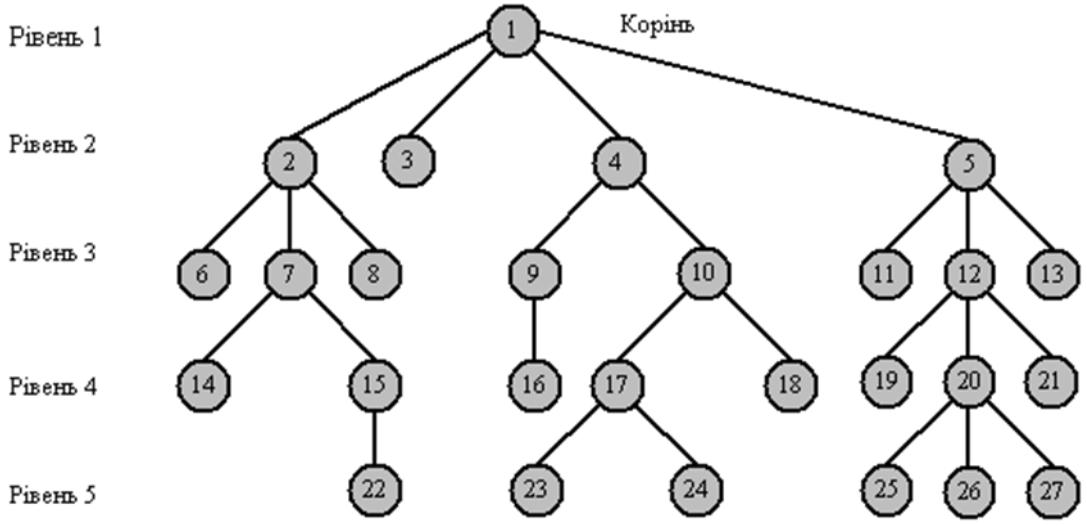


Рис. 2.1. Ієрархічна структура бази даних

2.5.4 Мережева база даних

Мережева модель даних представляє собою сукупність об’єктів різного рівня, де кожний об’єкт зв’язаний з іншим.

Мережева модель даних подібна до ієрархічної моделі тим, що вона має однакові базові компоненти (вузли, рівні, зв’язки), але характер зв’язків між ними принципово інший. У мережевих моделях допускається вільний зв'язок між елементами на різних рівнях.

Серед переваг даної моделі варто відзначити ефективність використання пам’яті та швидкість реагування, загальність (можливості мережевої моделі найширші в порівнянні з іншими моделями), можливість доступу до даних через декілька зв’язків (через будь-яке базове співвідношення), є відсутність дублювання даних. Що стосується недоліків, то варто підкреслити складність такої моделі, оскільки існує велика кількість концепцій, варіантів зв’язків та їх характеристик, що ускладнює розуміння та реалізацію моделі.

2.6. Висновки до розділу

У цьому розділі було проведено детальне дослідження технологій проектування веб-додатків. Розглянувши різні аспекти, від тенденцій веб-додатків до клієнтської та серверної частин, а також баз даних, було отримане глибоке розуміння процесу створення веб-додатків і факторів, які впливають на їх розробку та функціонування.

Усі технології та компоненти, що були розглянуті, використовуються для створення веб-додатків різної складності та функціональності. Вони дозволяють розробникам спростити розробку та підвищити продуктивність веб-додатків, зосередитися на реалізації логіки, забезпечуючи готові рішення для таких завдань, як маршрутизація, автентифікація, кешування та багато іншого.

Розуміння технологій проектування веб-додатків є важливим кроком у створенні успішних та функціональних проектів. Незалежно від того, чи ми використовуємо популярні фреймворки, клієнтські інструменти або різні види баз даних, можна сказати, що засвоєння технологій проектування веб-додатків, розуміння різних компонентів та їх взаємодії допоможе нам створити успішний та функціональний веб-додаток, який задовольняє потреби користувачів та відповідає сучасним вимогам.

РОЗДІЛ 3. СТВОРЕННЯ ВЕБ-ДОДАТКУ

3.1 Створення клієнтської частини

Фронтенд відповідає за створення користувацького інтерфейсу, з яким взаємодіє користувач. Це включає в себе дизайн, розташування елементів, взаємодію з користувачем, анімацію та інше. Гарний та інтуїтивно зрозумілий інтерфейс сприяє кращому взаємодії з користувачами та поліпшує їх досвід використання додатка. Саме тому створення клієнтської частини є важливим пунктом створення веб-додатка.

Для реалізації інтерфейсу користувача були використані такі технології як *HTML, CSS, JavaScript* та *Bootstrap*.

Розробка починається зі створення файлу *HTML*, який містить структуру та вміст веб-сторінки.

В самому файлі потрібно обов’язково заповнити meta теги для передачі додаткової інформації про веб сторінку та підключити файли для покращення організації та управління коду, такі як *CSS* – стилі та *JavaScript* – скрипти.

```
<head>
   <meta lang="uk" >
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="stylesheet" href="style.css">
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css"
```

```
    integrity="sha512-
iecdLmaskl7CVkqkXNQ/ZH/XLlvWZOJyj7Yy7tcenmpD1ypASozpmT/E0iPtmFIB46ZmdtA
c9eNBvH0H/ZpiBw=="
    crossorigin="anonymous" referrerpolicy="no-referrer" />
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.0.1/css/bootstrap.min.css">
    <title>Calendar</title>
</head>
```

Створюємо модальне вікно реєстрації та логіну користувача (рис. 3.1). Це спливаюче вікно, яке з'являється поверх вмісту сторінки та перекриває його, знадобиться для отримання імені та паролю користувача, які ми використаємо з часом для аутентифікації і в подальшому для підключення до бази даних.

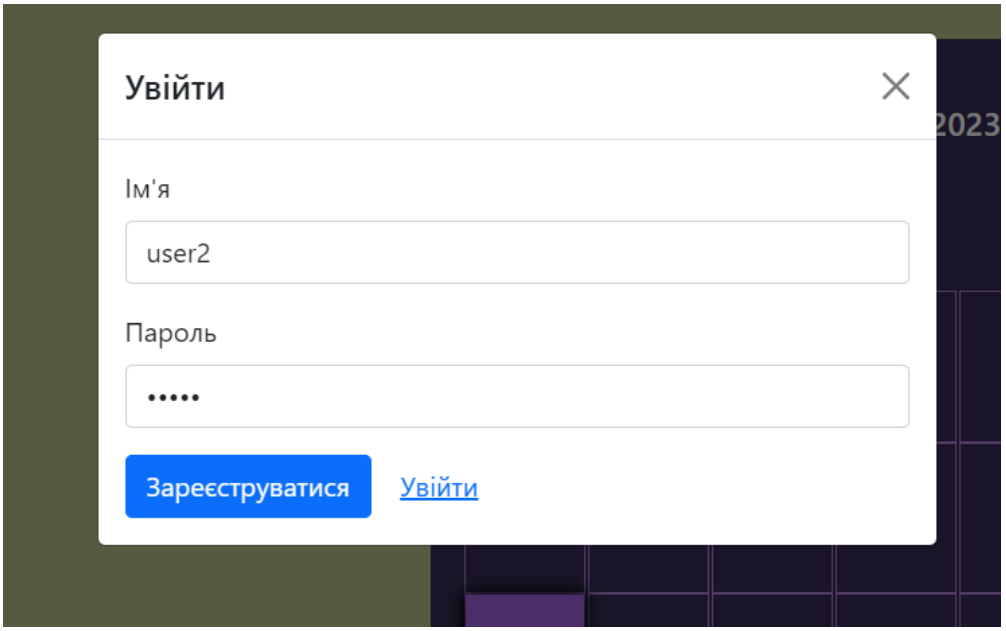


Рис. 3.1. Вікно реєстрації та логіну

У вказаному коді використовується *Bootstrap*, популярний фреймворк для розробки веб-інтерфейсів. Він надає набір готових стилів, компонентів та інструментів, які значно спрощують процес розробки та покращують зовнішній вигляд веб-сторінок.

Bootstrap забезпечує зручні класи та *CSS*-стилі для стилізації та розміщення елементів у модальному вікні. Наприклад, клас *modal* вказує, що елемент є модальним вікном, а клас *fade* додає ефект зникнення/з'явлення при зміні видимості вікна.

```
<div class="modal fade" id="loginModal" tabindex="-1" aria-labelledby="loginModalLabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header"></div>
      <div class="modal-body"></div>
    </div>
  </div>
</div>
```

Далі представлено *HTML*-код який буде відображати структуру сторінки для календаря з завданнями до кожного активного дня:

```
<div class="conteiner">
    <div class="left">
        <div class="today-date"> </div>
        <div class="events"></div>
        <div class="add-event-wrapper ">
            <div class="add-event-header"></div>
            <div class="add-event-body"></div>
            <div class="add-event-footer">
                <button class="add-event-btn">Зберегти</button>
            </div>
        </div>
    </div>
<div class="right">
        <div class="calendar">
            <div class="month"></div>
```

```
        <div class="weekdays">
            <div>пн</div>
            <div>вт</div>
            <div>ср</div>
            <div>чт</div>
            <div>пт</div>
            <div>сб</div>
            <div>нд</div>
        </div>
        <div class="days"></div>
        <div class="goto-today">
            <div class="goto">
                <input type="text" placeholder="mm/yyyy" class="date-input">
                <button class="goto-btn">go</button>
            </div>
            <button class="today-btn">today</button>
        </div>
    </div>
</div>
```

Без застосування стилів *CSS* веб-додаток має дуже сирий вигляд. Тому є обов'язковим їх написання. *CSS* дозволяє створювати класи та ідентифікатори, які можна застосовувати до багатьох елементів на сторінці. Це спрощує кодування та забезпечує повторне використання стилів на різних сторінках, що сприяє швидкості розробки та зменшує обсяг коду.

Після додавання інструкцій до елементів, таких як кольори, відступи, шрифти, розташування контенту та багато іншого, наш веб-застосунок має вже приємний для користувача інтерфейс. В порівняння з тим що було до підключення стилів.

Інтерфейс додатку до підключення *CSS* (рис. 3.4) та після (рис. 3.5).

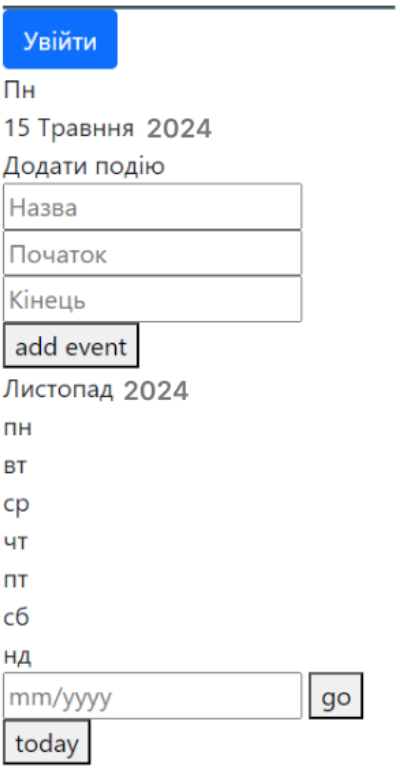


Рис. 3.4. Інтерфейс до підключення *CSS*

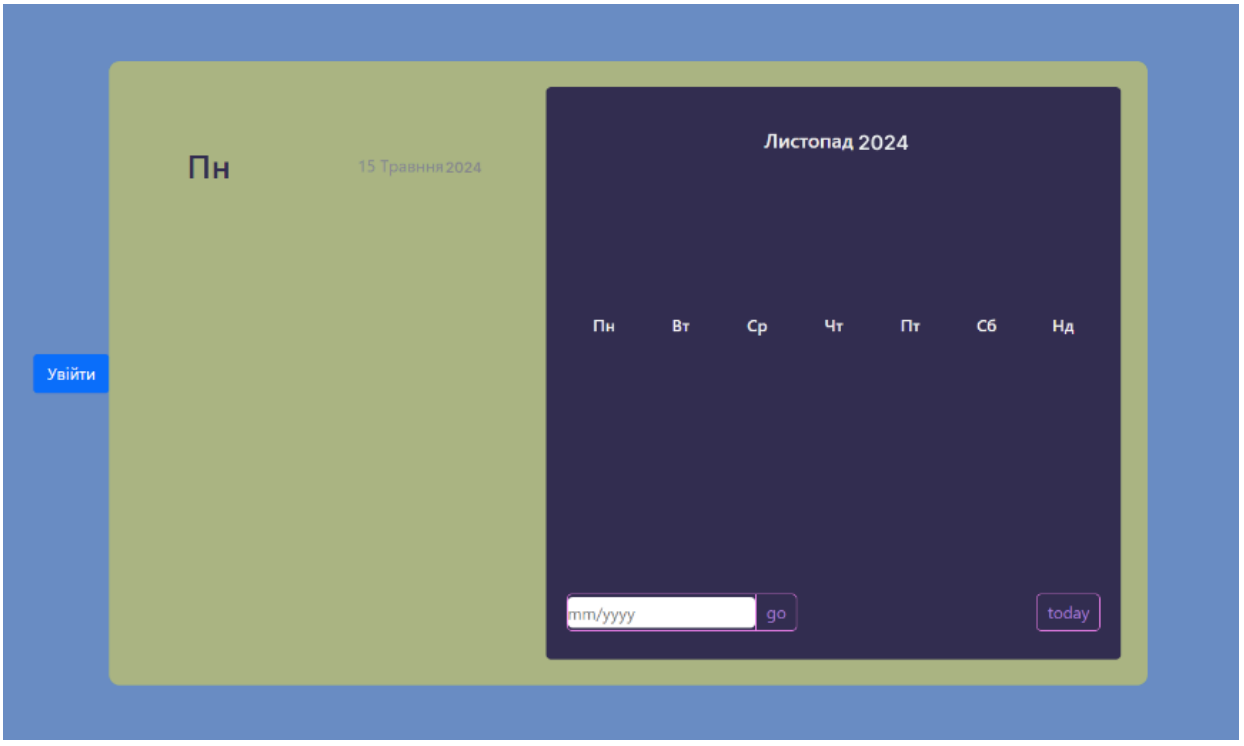


Рис. 3.5. Інтерфейс після підключення *CSS*

Наступним важливим кроком в реалізації додатку буде додавання основної функціональності календаря, до якої входить додавання та відображення подій, реалізація кнопок, та багато іншого за допомогою *JavaScript*.

Нижче представленні основні зміні як відповідають за календар, його ініціалізацію, відображення днів поточного, наступного та попереднього місяців, кнопки для переходу сьогоднішньої дати.

```
const calendar=document.querySelector(".calendar"),
   date = document.querySelector(".date"),
   daysContainer = document.querySelector(".days"),
   prev = document.querySelector(".prev"),
   next = document.querySelector(".next"),
   todayBtn = document.querySelector(".today-btn"),
   gotoBtn = document.querySelector(".goto-btn"),
   dateInput = document.querySelector(".date-input"),
   eventDay=document.querySelector(".event-day"),
   eventDate=document.querySelector(".event-date"),
   eventsContainer=document.querySelector(".events");
   addEventSubmit=document.querySelector(".add-event-btn");
   let today = new Date();
   let activeDay;
   let month=today.getMonth();
   let year=today.getFullYear();
   const months=[
      "січень", "лютий", "березень",
      "квітень", "травень", "червень", "липень",
      "серпень", "вересень", "жовтень", "листопад", "грудень"
   ];
```

При створенні скриптів використовується *document.querySelector()* що є методом *JavaScript*, який використовується для отримання посилання на перший елемент у документі, що відповідає вказаному селектору *CSS*. Селектор *CSS* - це рядок, який використовується для вибору елементів на сторінці на підставі їх класів, ідентифікаторів, типів або інших атрибутів.

Отримання посилань на елементи сторінки дозволяє взаємодіяти з ними за допомогою *JavaScript*. Я можу додавати події, слухачі, змінювати їх значення або стилі.

Веб-додатки повинні бути адаптивними, тобто здатними коректно відображатися на різних пристроях та екранах різного розміру. Використання медіа-запитів дозволяє налаштувати стилі для різних пристроїв, забезпечуючи кращу відповідність та коректне відображення змісту.

Це дуже важливо для забезпечення зручності використання веб-додатку на різних пристроях, від мобільних пристроїв до настільних комп'ютерів. Завдяки належному налаштуванню стилів за допомогою медіа-запитів, я забезпечу, зручність для користувачів та ефективність додатку на різних пристроях з різними розмірами екранів.

```
@media screen and (max-width: 768px) {
  body {
    align-items: flex-start;
    justify-content: flex-start;
  }
  body .conteiner {
    min-height: 100vh;
    flex-direction: column;
    border-radius: 0;
  }
@media screen and (max-width: 500px) {
  .calendar .weekdays {    height: 50px;  }
  .calendar .days .day {    height: 40px;    font-size: 0.8rem;  }
  .right .today-date {
padding: 20px;  }
}
```

Приклад вигляду календаря з екрану мобільного пристрою *iPhone 12Pro* наведено на рисунку 3.6.

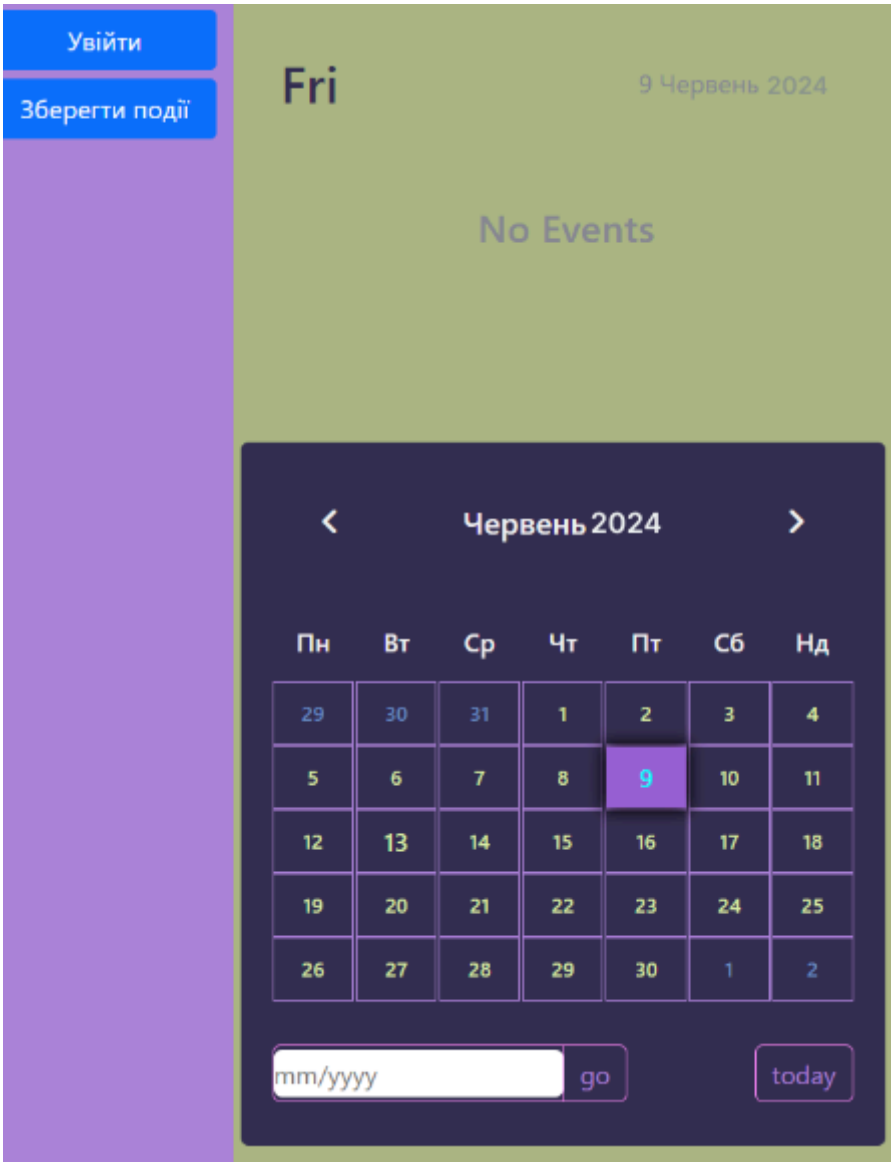


Рисунок 3.6. Вигляд календаря при ширині екрану в 390*px*

На даному етапі розробки можна сказати що клієнтська частина додатку вже закінчена. Технологій *HTML*, *Bootstrap*, *CSS* та *JavaScript* цілком вистачило для опису всіх необхідних деталей та функціоналу на клієнтській частині нашого застосунку.

Приклад роботи веб-додатка зображено на рисунку 3.7.

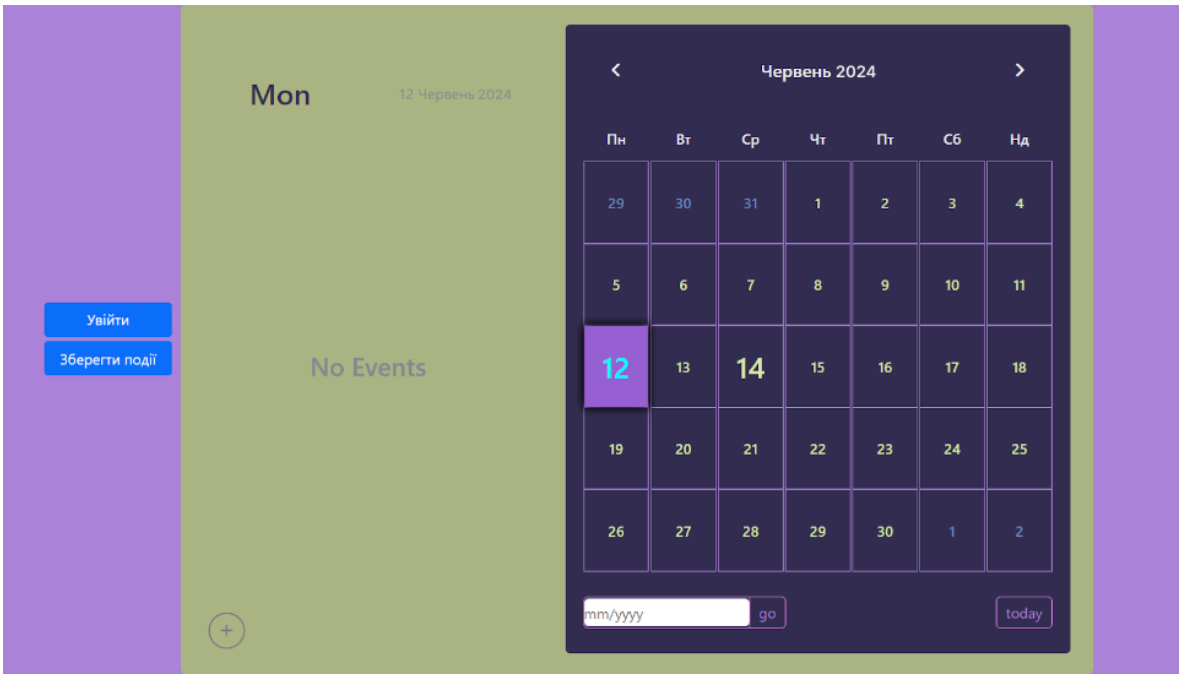


Рисунок 3.7. Веб-додаток

1.2. Підключення серверної частини веб-додатку

Налаштування серверної частини є важливим етапом у створенні веб-додатка, оскільки сервер відповідає за обробку запитів від клієнта, взаємодію з базою даних та надання відповідей клієнту забезпечує роботу додатка в мережі та забезпечує його функціональність.

Налаштовувати сервер будемо за допомогою *Node.js* та *Express.js*, що дозволяє зручно створювати маршрути, обробляти запити, працювати з базою даних та забезпечувати зручну комунікацію з фронтендом мого додатка.

Для створення проекту *Node.js* в терміналі каталога, де я хочу створити новий проект ввожу команду *npm init -y* .

npm автоматично створить файл *package.json* зі значеннями за замовчуванням. Файл *package.json* є файлом конфігурації для проектів *Node.js*. Він зберігає інформацію про проект, таку як назва проекту, версія, залежності, скрипти для запуску та інші налаштування.

```
{ "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2",
    "mongoose": "^7.2.2",
    "nodemon": "^2.0.22"
  }}
```

Express.js є популярним фреймворком для створення серверних додатків на *Node.js*. Завантажемо фреймворк за допомогою команди *npm install express*. Потім створемо файл *app.js* для створення та налаштування сервера.

Перше що слід зробити для налаштування це імпорт *Express.js* в файлі сервера:

const express = require('express')

Потім створюю екземпляр додатку *Express.js*:

const app = express()

Налаштування маршрутів для сервера, які будуть відповідати на різні типи запитів будуть зберігатися в файлі *authRouter.js*

Для запуску самого сервера використовується функція *app.listen()*

3.3. Підключення бази даних

У веб-додатку використовуємо базу даних *MongoDB Atlas*, яка є хмарним сервісом бази даних *MongoDB*.

Він використовується для зберігання та керування даними веб-додатків. До переваг можна віднести простоту, адже *MongoDB Atlas* має інтуїтивно зрозумілий інтерфейс, який спрощує створення та налаштування кластерів бази даних. Це дозволяє розробникам швидко налаштовувати та розгортати базу даних без необхідності займатися фізичною інфраструктурою. Іншим важливим аспектом є масштабованість сервісу. Розробники можуть легко масштабувати свою базу даних вгору або вниз, додавати або видаляти вузли, щоб забезпечити потрібну продуктивність та доступність. Це дає можливість легко реагувати на зростаючі потреби вашого додатку та забезпечувати оптимальну продуктивність.

Створюємо новий проект та кластер на *MongoDB Atlas*, налаштовуючи параметри кластера, такі як його розмір та типи вузлів. Потім я отримаю рядок підключення до кластера на *MongoDB Atlas*. Цей рядок підключення містить *URL*, який дозволяє веб-додатку з'єднатися з базою даних (рис. 3.8.).

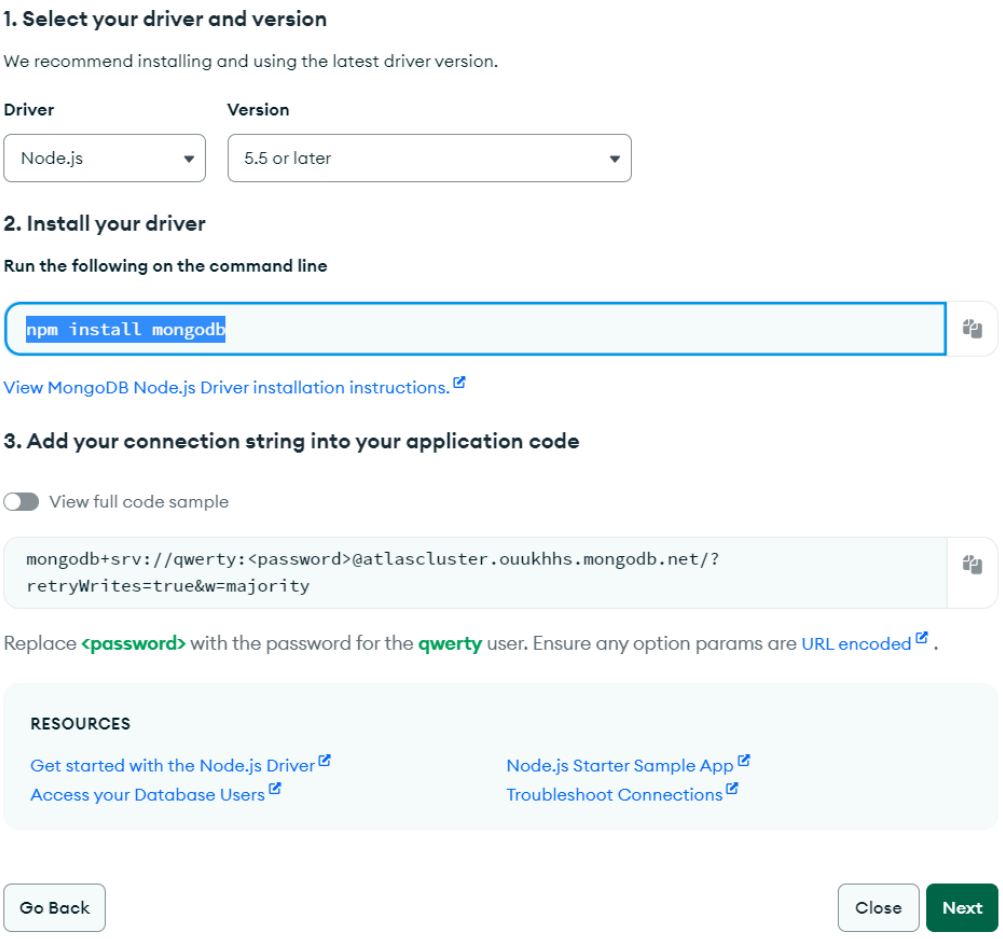


Рисунок 3.8. Створення нового проекту *MongoDB Atlas*

Після отримання рядка підключення, встановлюємо пакет *mongoose* – об'єктно-реляційний драйвер для *MongoDB*, який допомагає взаємодіяти з базою даних:

npm install mongoose

Далі я підключаю mongoose, додаючи рядок:

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

Щоб встановити з'єднання з базою даних я використовую функцію:

```
await mongoose.connect()
```

Функція *start()* підключається до бази даних та запускає сервер, що дозволяє веб-додатку працювати з даними з бази даних та обробляти *HTTP*-запити:

```
const start = async ()=>{
   try{
                                                                    await
mongoose.connect('mongodb+srv://qwerty:qwerty123@atlascluster.ouukhhs.mongodb.net
/Calendar')
      app.listen(PORT, () => console.log(`server sterted on PORT: ${PORT}`))
   } catch (e){
      console.log(e);
   }
}
start();
```

Моделі даних які використовуватимуться для користувачів і подій у базі даних описані в папці *modals*, а саме *User.js* та *dayEvents.js*.

Модель користувача:

```
const {Schema, model} = require('mongoose')
const User = new Schema({
   username: {type: String, unique: true, required: true},
   password: {type: String, required: true},
})
```

```
module.exports = model('User', User)
```

Медель події:

```
const mongoose = require('mongoose');
const eventsSchema = new mongoose.Schema({
  day: Number ,
  month: Number,
  year: Number,
  events:[{
    title: String,
    time: String
  }],
  userid: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }
});
const dayEvents = mongoose.model('dayEvents', eventsSchema);
module.exports = dayEvents;
```

3.4. Реалізація серверної логіки

В цьому розділі представлено детальний опис ключових компонентів, які відповідають за обробку запитів та взаємодію з базою даних у серверній частині додатку. Цей розділ розкриває архітектуру та функціональні можливості файлів налаштування маршрутів та запитів до бази даних.

Перш за все, представлено опис маршрутів, що визначають доступні шляхи та методи *HTTP* для взаємодії з сервером. Кожен маршрут має свою унікальну адресу та зв'язаний з ним обробник, який виконує певні дії при отриманні відповідного запиту.

```
router.post('/registration',[
   check('username',"Ім'я користувача не може бути пустим").notEmpty(),
   check('password',"Пароль має бути не менше 4 символів").isLength({min:4})
   ],controller.registration )
```

```
router.post('/login',controller.login)
router.get('/users',controller.getUsers)
router.get('/events',controller.getdayEvents)
router.get('/user',controller.getUser)
router.get('/deleteEvents',controller.DeleteEvents)
router.post('/addEvents',controller.addayEvents)
```

Другий компонент – контролер, який відповідає за обробку запитів та виконання потрібних операцій у базі даних. Кожен метод контролера реалізує певну функціональність, таку як реєстрація користувача, автентифікація, отримання та збереження подій, видалення даних тощо.

Приклад реалізації методу аутентифікації:

```
async login(req,res){
    try{
        const {username, password} = req.body;
        const user = await User.findOne({username});//провіряємо чи є в базі даних такий користувач
        if (!user) {
            return res.status(400).json({message: `Користувач ${username} не знайдений`})
        }
        const validPassword = bcrypt.compareSync(password,user.password);
        if (!validPassword) {
            return res.status(400).json({message: `Введений пароль невірний`})
        }
        session.username=user.username;
        return res.json({message:`Користувач ${user.username} Id користувача: ${user.id}, Назва сесії ${session.username}`});

    }catch(e){
        console.log(e)
```

res.status(400).json({message: `Login errorCurrrent session id ${session.id}`})

} }

Для перевірки запитів використовуємо надзвичайно корисну програму – *Postman* (рис. 3.9). Вона спрощує взаємодію з *API* та дозволяє легко відправляти *HTTP*-запити та отримувати відповіді.

Завдяки *Postman* швидко та зручно тестувати різні типи запитів (*GET, POST, PUT, DELETE)* та передавати параметри, тіла запиту. Це дозволяє переконатися, що запити до сервера працюють належним чином та отримують очікувані результати.

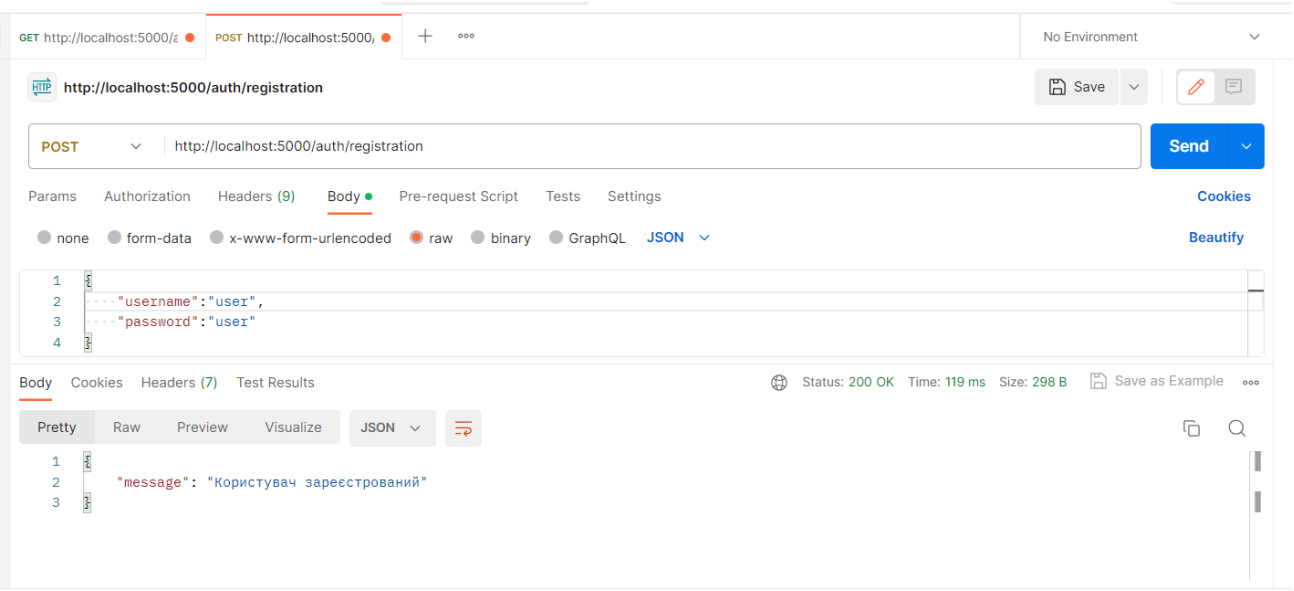


Рисунок 3.9. Робота програми Postman

3.5. Взаємодія між клієнтом та сервером

Взаємодія між клієнтською та серверною частинами відіграє вирішальну роль у створенні ефективного та функціонального веб-додатку.

Передусім, варто зазначити, що взаємодія між клієнтом і сервером дозволяє обмінюватись даними, виконувати запити та отримувати відповіді. Клієнтська частина, зазвичай реалізована з використанням *HTML, CSS* та *JavaScript*, взаємодіє з сервером, що обробляє ці запити і повертає відповіді.

Важливим аспектом взаємодії є використання функції *Fetch()*. Ця функція в JavaScript надає потужний механізм для виконання *HTTP*-запитів до сервера прямо з клієнтського браузера. Вона дозволяє відправляти різні типи запитів, такі як *GET, POST, PUT, DELETE*, і отримувати відповіді в різних форматах, таких як *JSON* або *HTML*.

Під час розробки веб-додатку використовуємо *fetch* функції для здійснення запитів до сервера. Нижче наведений код реєстрації з використанням *POST*- запиту та код функції для отримання списку подій з сервера за допомогою *GET*-запиту:

```
function Registration(username, password) {
    const reg = JSON.stringify({ username, password });
    fetch('/registration', {
      method: 'post',
      headers: { 'Content-Type': 'application/json', },
      body: reg,
    })
      .then(response => response.json())
      .then(data => { console.log('Ви Зареєструвалися до бази даних:', log);  })
      .catch(error => { console.log('Сталася помилка при реєстрації:', error); });
      alert('Успішна реєстрація ');
      var loginModal = document.getElementById('loginModal');
      var modalBackdrop = document.getElementsByClassName('modal-backdrop')[0];
      document.body.classList.remove('modal-open');
      loginModal.classList.remove('show');
      modalBackdrop.classList.remove('show');
      Login(username,password)
  }
function getEvents() {
fetch('/events')
.then(  response => response.json())
.then(data => {
```

```
    if (!data) {
      console.log(`${data}`);
      return;
    }else{
    eventsArr = eventsArr.concat(data);
    }
  });
}
```

3.6. Приклади роботи готового застосунку

Початкова сторінка календаря подій з відображенням сьогоднішнього дня наведено на рисунку 3.10.

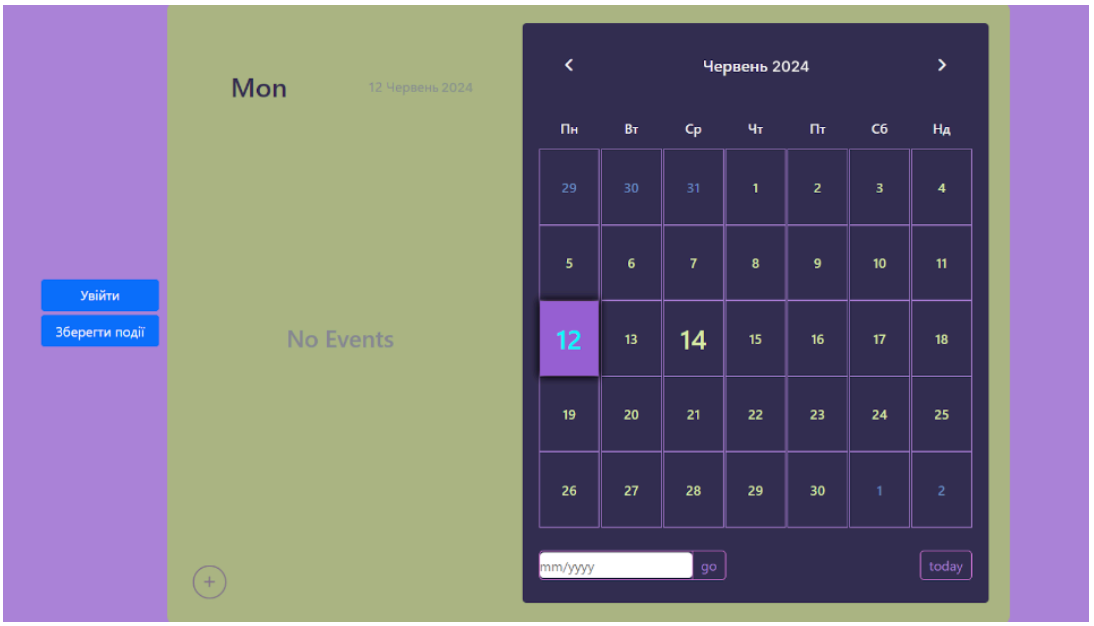


Рисунок 3.10. Початкова сторінка

Початкова сторінка календаря подій з відображенням сьогоднішнього дня та записаними подіями наведено на рисунку 3.11.

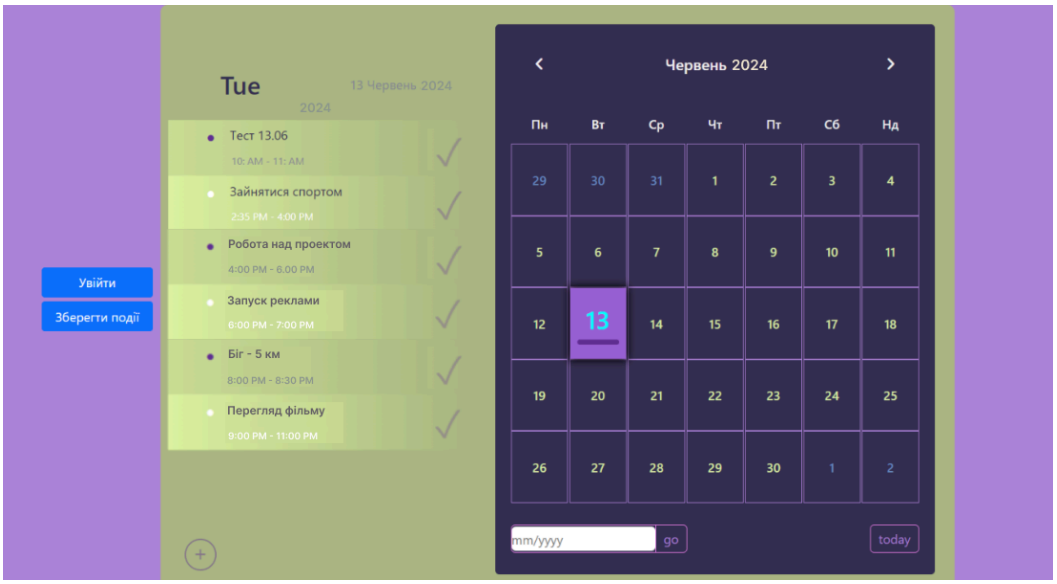


Рисунок 3.11. Початкова сторінка з подіями

Модальне вікно логіну та реєстрації наведено на рисунках 3.12 та 3.13 відповідно.

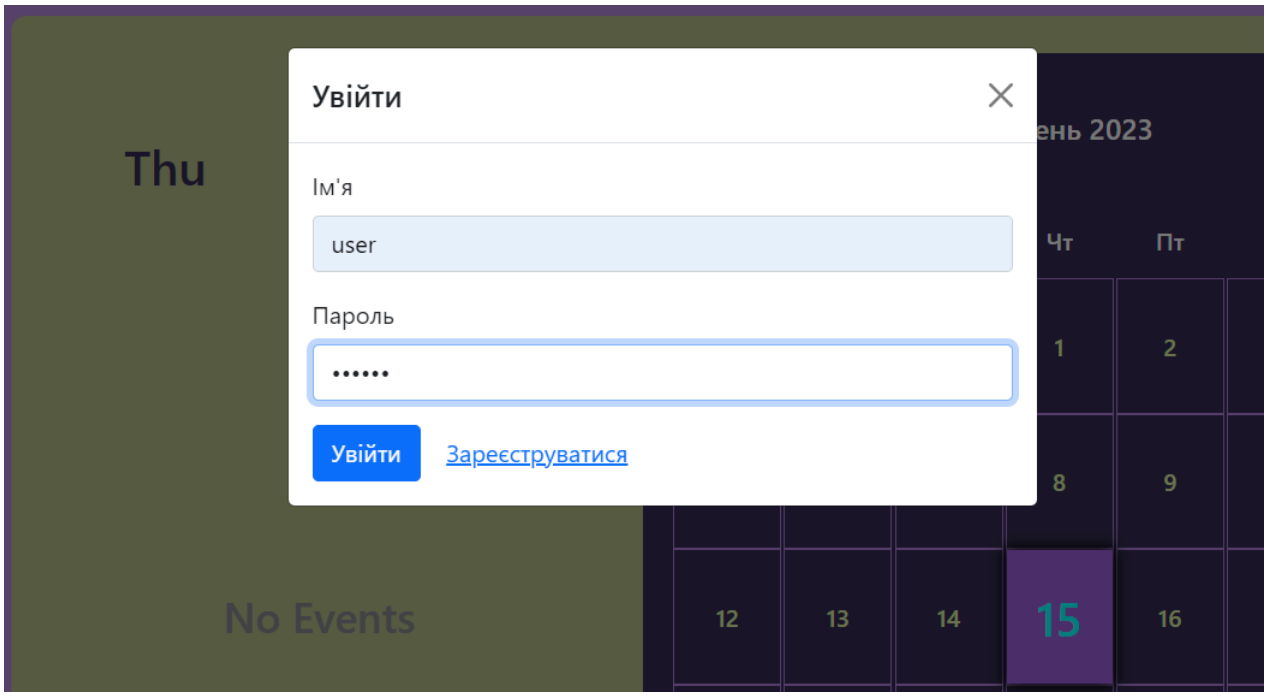


Рисунок 3.12. Вікно логіну

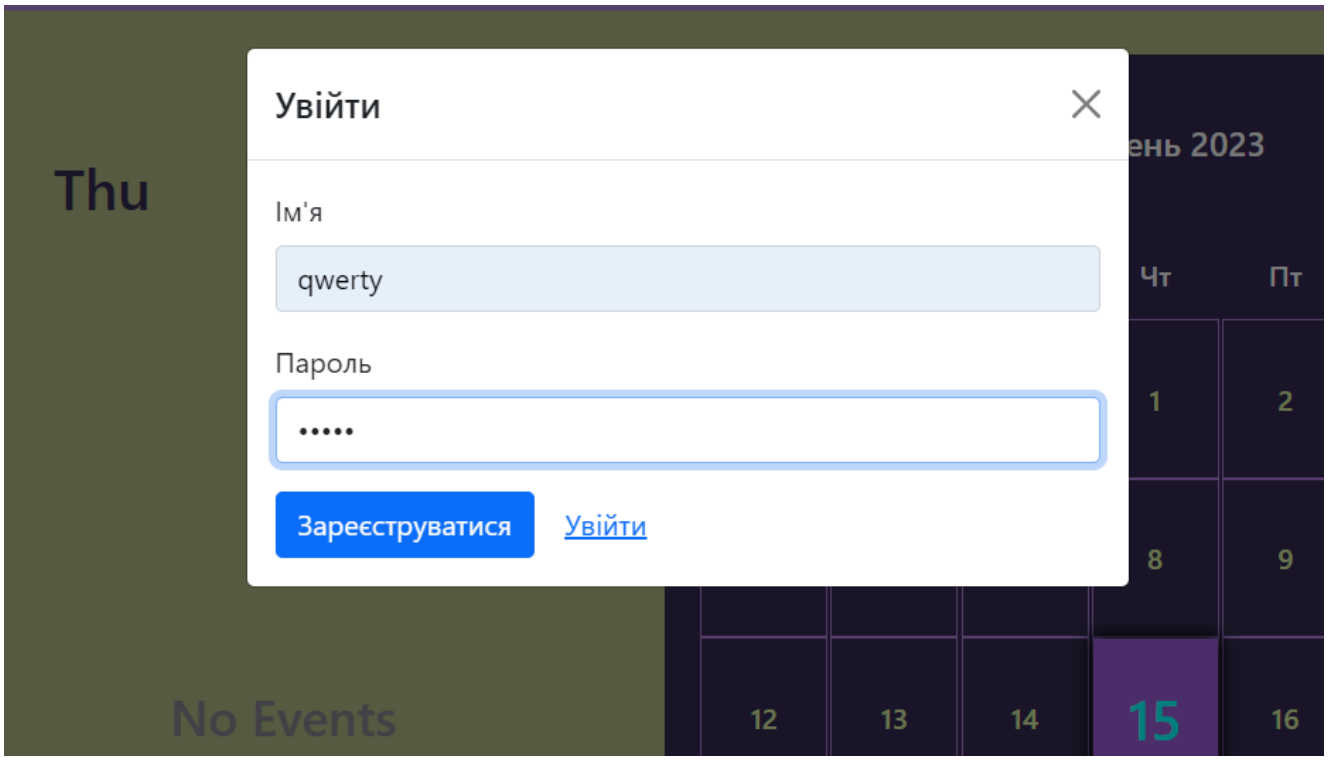


Рисунок 3.13. Вікно реєстрації

3.6. Висновки до розділу

Розділ «Створення веб-додатку» був присвячений процесу розробки повноцінного веб-додатку з використанням клієнтської та серверної частин, а також інтеграції з базою даних. Було розглянуто створення клієнтської частини додатку, а також наведено приклади коду які демонструють процес взаємодії користувача з додатком шляхом обробки подій та здійснення запитів до сервера з використанням функції *Fetch()*.

Було детально розглянуто серверну частину додатку, використовуючи фреймворк *Express.js*. Описано підключення сервера до клієнтської частини за допомогою *app.use(express.static('Client'))*, яке дозволяє надавати клієнту доступ до статичних файлів.

Описані приклади коду надають практичний уявлення про реалізацію цих процесів. Загалом цей розділ створює основу для подальшого розвитку та вдосконалення веб-додатку, дозволяючи користувачам взаємодіяти з додатком та здійснювати необхідні дії.

ВИСНОВКИ

У даному дипломному проекті було проведено детальне дослідження веб-додатків, зокрема зосереджене на плануванні та управлінні важливими подіями. Під час аналізу сучасних веб-додатків було розглянуто поняття веб-додатку та проведено огляд найбільш популярних рішень в галузі планування. Це дозволило отримати важливі дані щодо функціональності, дизайну та користувацького досвіду, що відтворюються в цих додатках.

У другому розділі було розглянуто технології проектування веб-додатків, популярні фреймворки та бібліотеки, серверну частину та використання баз даних. Зазначені технології є ключовими складовими у створенні веб-додатків планування, оскільки вони забезпечують ефективну розробку, швидкість та масштабованість системи. Було виділено сучасні тенденції у галузі та розглянуто інструменти, що допомагають покращити якість та продуктивність веб-додатків.

В останньому розділі присвяченому процесу створення веб-додатку планування, було розглянуто ключові його етапи, такі як розробка клієнтської частини, підключення серверної частини та бази даних, реалізація серверної логіки та взаємодія між клієнтом та сервером. Крім того, були надані приклади роботи готового веб-додатку, розробленого в рамках проекту. Це дозволило продемонструвати практичне застосування отриманих знань та навичок у реальному проекті.

В цілому, дана робота дала можливість ознайомитися з сучасними веб-додатками планування, вивчити ключові технології їх розробки та успішно реалізувати власний веб-додаток. Результати дослідження та розробки дають підстави вважати, що веб-додаток планування має великий потенціал у покращенні організації часу та управлінні подіями. Застосування сучасних технологій та розроблений функціонал дозволяють користувачам зручно та ефективно планувати свої дії та події. Дана робота відкриває шлях для подальшого розвитку та вдосконалення веб-додатків планування з метою забезпечення кращого користувацького досвіду та задоволення потреб сучасного суспільства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко С. В., Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. / С. В. Бойченко, О. В. Іванченко. – Київ: НАУ, 2017. – 63 с.

2. ДСТУ 3008-95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення».

3. Пасічник В. В., Веб-технології та веб-дизайн. Книга 1. Веб-технології. / Пасічник В. В., Пасічник О. В., Угрин Д. І. – Львів: Магнолія , 2021.- 336с.

4. Фрімен Е., *Head First*. Програмування на *JavaScript*. / Фрімен Е., Робсон Е. – Харків: Фабула , 2022 – 672с.

5. *What do client side and server side mean* [Електроний ресурс] – Режим доступу: *https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/* (Дата звернення: 25.05.2023) – Назва з екрана.

6. Що таке веб додаток? Різниця між сайтом, веб-додатком, *SPA* і *PWA* [Електронний ресурс] – Режим доступу: *https://webcase.com.ua/uk/blog/cho-takoe-web-prilozhenie-vse-vidy/* (дата звернення: 25.05.2023) – Назва з екрана

7. Огляд фреймворків *JavaScript*. Що, для чого і коли використовувати [Електронний ресурс] – Режим доступу: *https://dou.ua/forums/topic/34739/* (дата звернення: 27.05.2023) – Назва з екрана.

8. *Google calendar reviwer* [Електронний ресурс] – Режим доступу: *https://www.techradar.com/reviews/google-calendar* (дата звернення: 27.05.2023) – Назва з екрана.

9. *Any.do* [Електронний ресурс] – Режим доступу: *https://www.techradar.com/reviews/anydo* (дата звернення: 27.05.2023) – Назва з екрана.

10. Реляційна база даних [Електронний ресурс] – Режим доступу: *https://ua5.org/database/189-reljacjjna-baza-danikh.html/* (дата звернення: 28.05.2023) – Назва з екрана.

11. *Express. 4.x API* [Електронний ресурс] – Режим доступу: *https://expressjs.com/uk/4x/api.html#express* (дата звернення: 30.05.2023) – Назва з екрана.

12. *Npm. bcrypt.js* [Електронний ресурс] – Режим доступу: *https://www.npmjs.com/package/bcryptjs* (дата звернення: 30.05.2023) – Назва з екрана.

13. *How to use username in Session* [Електронний ресурс] – Режим доступу: *https://www.tabnine.com/code/javascript/functions/express-session/Session/username* (дата звернення: 30.05.2023) – Назва з екрана.

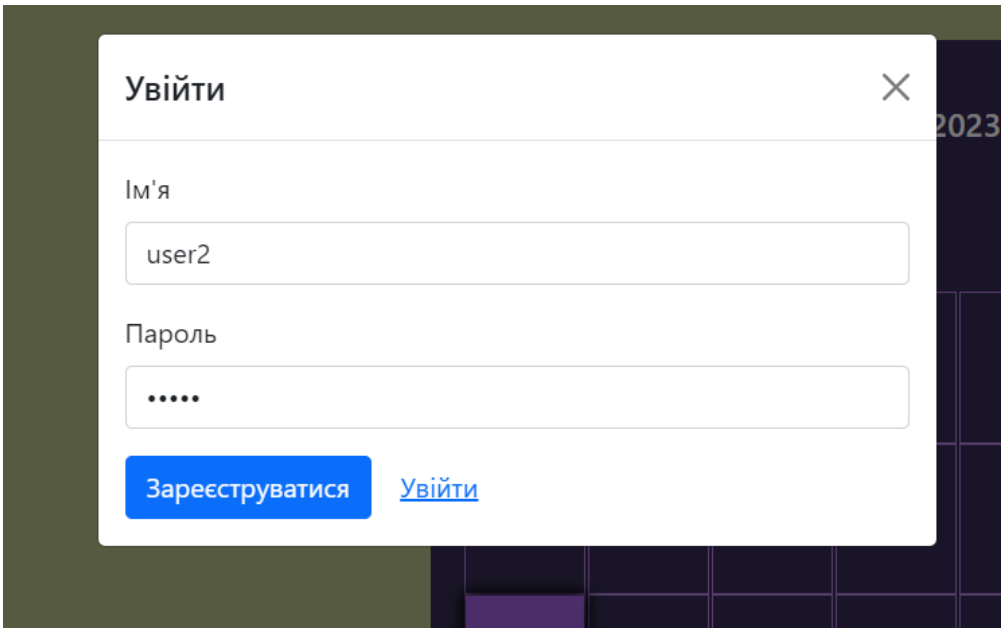
14. Вступ до *Mongoose* для *MongoDB* та *Node.js* [Електронний ресурс] – Режим доступу: *https://code.tutsplus.com/uk/articles/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527* (дата звернення: 30.05.2023) – Назва з екрана.

15. *Using the Fetch API* [Електронний ресурс] – Режим доступу: *https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch* (дата звернення: 02.06.2023) – Назва з екрана.

ДОДАТКИ

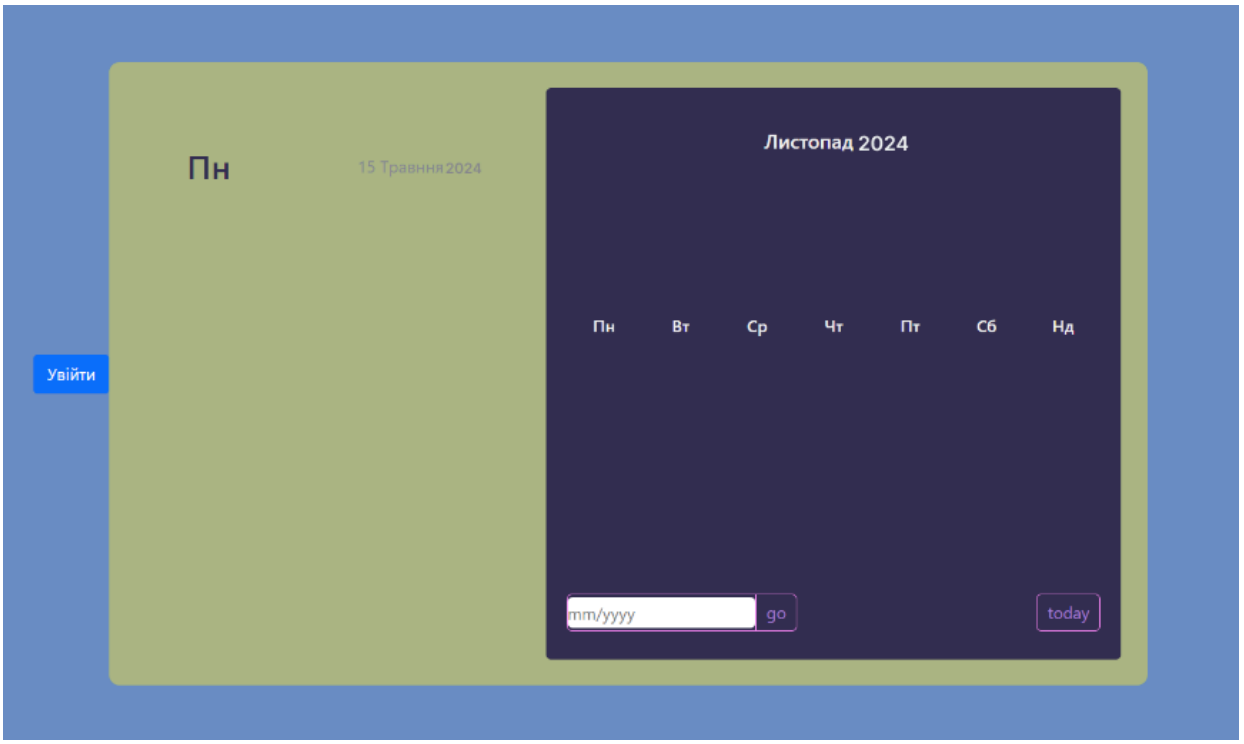
ДОДАТОК А

Вікно авторизації



ДОДАТОК Б

Незаповнений календар



ДОДАТОК В

Заповнений календар з подіями