ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: **«розробка веб додатку "TaskMaster" за допомогою React, Node.js, MongoDB»**

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Євгеній ДВУРЕЧЕНСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав: здобувач вищої освіти гр. КНД-41
Євгеній ДВУРЕЧЕНСЬКИЙ
(Ім'я, ПРІЗВИЩЕ)

Керівник: Олена ШИКУЛА
Науковий ступінь, вчене звання *(Ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
Науковий ступінь, вчене звання *(Ім'я, ПРІЗВИЩЕ)*

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти «Бакалавр»

Спеціальність 122 «Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Двуреченський Євгеній Анатолійович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка веб додатку "TaskMaster" за допомогою React, Node.js, MongoDB
керівник кваліфікаційної роботи Олена ШИКУЛА д.ф.-м.н, професор
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» 02.2024 р. №36
2. Строк подання кваліфікаційної роботи «31» травня 2024 р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, документація по React, Redux toolkit, MongoDB та Nodejs, параметри налаштування кластеру.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Опис та аналіз інснуючих аналогів.
    2. Створення та налаштування клієнтської частини(React).
    3. Створення та налаштування серверної частини(Node.js).
    4. Налаштування взаємодії між клієнтською і серверною частиною.
    5. Опис функціоналу.
5. Перелік графічного матеріалу:
    1. Приклади програмного коду

2. Вибір параметрів налаштування кластеру
3. Екранні форми додатку
4. Презентація проекту

6. Дата видачі завдання «27» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Літературний огляд теми	27.02-20.03.24	Виконано
2	Формулювання проблеми та мети дослідження	20.03-24.03.24	Виконано
3	Аналіз та вивчення інструментів розробки	24.03-10.04.24	Виконано
4	Розробка веб додатку	10.04-11.05.24	Виконано
5	Написання тексту дипломної роботи	11.05-18.05.24	Виконано
6	Редагування та оформлення дипломної роботи	18.05-24.05.24	Виконано
7	Створення презентації	24.05-25.05.24	Виконано

Здобувач вищої освіти _____ Євгеній ДВУРЕЧЕНСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Олена ШИКУЛА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 58 сторінок, 34 рисунків, 20 джерел.

Мета роботи – розробка веб-додатку "TaskMaster" з використанням сучасних технологій React, Node.js та MongoDB для управління завданнями, покращення продуктивності користувачів та забезпечення зручності у використанні.

Об'єкт дослідження – процеси розробки та впровадження веб-додатків для управління завданнями.

Предмет дослідження – інструменти та методи розробки веб-додатків, включаючи фреймворк React, серверну платформу Node.js та базу даних MongoDB.

Короткий зміст роботи:

Досліджуються та реалізуються основні етапи розробки веб-додатку "TaskMaster". Веб-додаток призначений для управління завданнями і включає функціонал створення, редагування, видалення та перегляду завдань. Для досягнення цієї мети використовуються такі інструменти та технології, як React для клієнтської частини, Node.js та Express для серверної частини, а також MongoDB для зберігання даних. Після чого створюється взаємодія між частинами.

Надається опис функціоналу веб додатку і представлений інтерфейс.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, УПРАВЛІННЯ ЗАВДАННЯМИ, REACT, NODE.JS, MONGODB

ABSTRACT

Text part of the qualification work for the bachelor's degree: 58 pages, 34 pictures, 20 sources.

The aim of the work – developing the web application "TaskMaster" using modern technologies such as React, Node.js, and MongoDB for task management, improving user productivity, and ensuring ease of use.

The object of research – the processes of development and implementation of web applications for task management.

The subject of research – the tools and methods for developing web applications, including the React framework, the Node.js server platform, and the MongoDB database.

Summary of the work:

In this work, the main stages of developing the "TaskMaster" web application are explored and implemented. The web application is designed for task management and includes functionality for creating, editing, deleting, and viewing tasks.

To achieve this goal, such tools and technologies as React for the client-side, Node.js and Express for the server-side, and MongoDB for data storage are used.

A description of the functionality of the web application and the presented interface are provided.

KEYWORDS: WEB APPLICATION, TASK MANAGEMENT, REACT, NODE.JS, MONGODB

ЗМІСТ

ВСТУП

У сучасному світі ефективне управління завданнями є ключовим фактором успішної діяльності. Веб-додатки для управління завданнями стають все більш популярними завдяки їх здатності забезпечувати зручний і швидкий доступ до інформації. Однак багато існуючих рішень мають свої недоліки, що обумовлює необхідність створення нових, більш функціональних та зручних додатків.

Розробка веб-додатку "TaskMaster" має на меті створення інструменту для ефективного управління завданнями, який би задовольняв сучасні вимоги користувачів. Основні технології, що використовуються в проекті, включають React для фронтенду, Node.js для бекенду та MongoDB для управління базою даних. Ці технології дозволяють забезпечити високу продуктивність, масштабованість та зручність використання додатку.

Об'єктом дослідження є процес управління завданнями за допомогою веб-додатків, що використовують сучасні технології фронтенд та бекенд розробки. Предметом дослідження є програмні засоби та технології, за допомогою яких можна реалізувати ефективну систему управління завданнями. Це включає вивчення таких технологій, як React, Node.js та MongoDB, а також аналіз їх переваг і недоліків у контексті розробки веб-додатків.

Мета роботи полягає у створенні функціонального та надійного веб-додатку для управління завданнями, який би відповідав сучасним вимогам користувачів та ринку. Додаток "TaskMaster" має забезпечити користувачів зручним інтерфейсом для створення, редагування, видалення та перегляду завдань, а також ефективною системою збереження та обробки даних.

Для досягнення поставленої мети необхідно виконати такі завдання:

Провести огляд та аналіз ринку існуючих рішень для управління завданнями, визначити їх переваги та недоліки.

Визначити проблеми, які має вирішити новий додаток, та обґрунтувати вибір технологій для його розробки.

Детально описати процес розробки додатку, включаючи створення та налаштування проекту, розробку клієнтської та серверної частин, інтеграцію всіх компонентів системи.

Розробити архітектуру додатку та описати основні функції та можливості "TaskMaster".

Оцінити досягнуті результати, аналізувати переваги розробленого додатку, а також обговорити перспективи його подальшого розвитку.

Таким чином, дана робота представляє собою комплексний аналіз та розробку веб-додатку для управління завданнями, що використовує сучасні технології для забезпечення високої функціональності та зручності у використанні. Вона охоплює всі етапи від аналізу ринку і вибору технологій до опису архітектури системи та її функціональних можливостей, що дозволяє створити ефективний і надійний інструмент для управління завданнями.

1 ОГЛЯД ТА АНАЛІЗ ТЕМАТИЧНОЇ ОБЛАСТІ

1.1 Огляд ринку та аналогічних рішень

У сучасному світі ефективне управління завданнями є ключовим фактором успішної діяльності. Веб-додатки для управління завданнями, або так звані "to-do" додатки, стають все більш популярними завдяки їх здатності забезпечувати зручний і швидкий доступ до інформації та розумне управління своїми справами і як наслідок часом, тому "to-do" є невід'ємним інструментом для успішного тайм-менеджменту.

Веб-додаток "TaskMaster" був створений за принципом додатків "to-do", тож розглянемо найбільш популярні рішення на ринку:

Todoist - це один з найпопулярніших "to-do" додатків, який надає користувачам можливість створювати та організовувати завдання. Основні функції Todoist включають:

Створення і редагування завдань

Встановлення термінів виконання завдань

Пріоритезація завдань

Додавання описів та вкладень до завдань

Сортування завдань за різними критеріями

Підтримка колективної роботи

Сильні сторони Todoist включають зручний інтерфейс, гнучкість в організації завдань та інтеграцію з багатьма іншими сервісами. Однак, деякі розширені функції доступні тільки у платній версії.

Microsoft To Do - це ще один популярний "to-do" додаток, який інтегрований з іншими продуктами Microsoft, такими як Outlook та Office 365. Основні можливості Microsoft To Do включають:

Створення і редагування завдань

Встановлення термінів виконання та нагадувань

Пріоритезація завдань

Додавання нотаток до завдань

Сортування завдань за різними критеріями

Підтримка колективної роботи

Сильні сторони Microsoft To Do полягають у його інтеграції з іншими продуктами Microsoft, що робить його зручним для користувачів екосистеми Microsoft. Проте, для користувачів, які не використовують інші продукти Microsoft, цей додаток може виявитися менш привабливим.

Google Keep - це простий і зручний "to-do" додаток від Google, який надає базові функції для управління завданнями. Основні функції Google Keep включають:

Створення і редагування завдань

Встановлення нагадувань

Додавання нотаток і списків

Сортування завдань

Інтеграція з іншими продуктами Google

Сильні сторони Google Keep включають простоту використання та інтеграцію з іншими сервісами Google. Однак, цей додаток має обмежені можливості порівняно з більш потужними інструментами для управління завданнями.

Any.do - це ще один популярний "to-do" додаток, який пропонує комплексний набір функцій для управління завданнями. Основні функції Any.do включають:

Створення і редагування завдань

Встановлення термінів виконання та нагадувань

Пріоритезація завдань

Додавання нотаток і вкладень до завдань

Сортування завдань за різними критеріями

Підтримка колективної роботи

Інтеграція з календарями та іншими сервісами

Сильні сторони Any.do включають багатофункціональність, зручний інтерфейс та можливість інтеграції з іншими сервісами, що робить його зручним для комплексного управління завданнями та часом.

Основні переваги ринку "to-do" додатків включають великий попит на такі рішення, оскільки вони допомагають користувачам ефективно організовувати свої завдання та підвищувати продуктивність. Крім того, завдяки зростаючій цифровізації та постійному вдосконаленню технологій, цей ринок має велику перспективу для розвитку. Інновації у функціональності, інтеграція з іншими сервісами та покращення користувацького досвіду забезпечують постійне зростання та розширення ринку "to-do" додатків.

1.2 Аналіз проблемної області

У попередньому розділі ми розглянули основні "to-do" додатки на ринку, такі як Todoist, Microsoft To Do, Google Keep та Any.do. Незважаючи на їх популярність і багатофункціональність, існує низка проблем, які не повністю вирішуються цими додатками. В цьому розділі ми проаналізуємо проблемну область, щоб зрозуміти, які саме аспекти потребують покращення і як "TaskMaster" може вирішити ці проблеми.

Проблеми зручності використання: однією з основних проблем багатьох "to-do" додатків є складність інтерфейсу для нових користувачів. Багато додатків мають багатофункціональні інтерфейси, які можуть бути заплутаними і непростими для освоєння. Наприклад, користувачам може бути важко швидко знаходити необхідні функції або інструменти, що знижує ефективність роботи з додатком. Важливість інтуїтивно зрозумілого та простого інтерфейсу важко переоцінити, особливо для користувачів, які лише починають користуватися подібними інструментами.

Обмежені функції у безкоштовних версіях: багато популярних "to-do" додатків мають обмежені можливості у безкоштовних версіях, що може стати серйозною перепоною для користувачів, які не готові платити за розширений функціонал. Наприклад, у Todoist деякі розширені функції доступні тільки у платній версії, що обмежує можливості користувачів без підписки. Це створює бар'єр для доступу до повного функціоналу додатку і знижує його привабливість для широкої аудиторії.

Відсутність функцій сортування і пріоритезації: деякі "to-do" додатки мають обмежені можливості сортування та пріоритезації завдань. Наприклад, можливість сортування завдань за датою створення, пріоритетністю або алфавітним порядком може бути важливою для багатьох користувачів. Відсутність таких функцій знижує ефективність управління завданнями і може викликати незручності у роботі. Забезпечення можливості гнучкого сортування і пріоритезації завдань є важливим аспектом для будь-якого "to-do" додатку.

Аналіз проблемної області "to-do" додатків виявив кілька ключових проблем, які потребують вирішення. Веб-додаток "TaskMaster" спрямований на усунення цих проблем, пропонуючи простий і інтуїтивно зрозумілий інтерфейс, відсутність платної версії та гнучкі можливості сортування і пріоритезації завдань. Це дозволить "TaskMaster" у преспективі зайняти свою нішу на ринку "to-do" додатків і забезпечити користувачам ефективне управління завданнями та підвищення продуктивності.

1.3 Цільова аудиторія

Розуміння цільової аудиторії є ключовим аспектом успішної розробки будь-якого додатку. Веб-додаток "TaskMaster" створений для задоволення потреб широкого спектру користувачів, яким необхідно ефективно управляти своїми завданнями. Враховуючи аналіз ринку та проблемної області, ми можемо визначити кілька основних груп користувачів, для яких наш додаток буде особливо корисним.

Індивідуальні користувачі: однією з основних цільових аудиторій для "TaskMaster" є індивідуальні користувачі, які шукають простий і зручний інструмент для управління своїми повсякденними завданнями. Ця група включає студентів, фрілансерів, домогосподарок та будь-кого, хто хоче покращити свій тайм-менеджмент. Для них важливо мати інтуїтивно зрозумілий інтерфейс, можливість швидкого створення, редагування та сортування завдань, а також функції пріоритезації.

Професіонали та фрілансери: професіонали та фрілансери, які працюють над кількома проектами одночасно, також є важливою цільовою аудиторією. Для них

критично важливо мати можливість ефективно управляти своїми завданнями, встановлювати пріоритети та слідкувати за дедлайнами. "TaskMaster" забезпечує функціонал, необхідний для організації роботи професіоналів, включаючи сортування завдань і встановлення пріоритетів. Важливо, що додаток підтримує збереження даних у хмарній базі даних, що дозволяє користувачам отримувати доступ до своїх завдань з будь-якого пристрою.

Освітні установи та студенти: освітні установи та студенти також можуть стати користувачами "TaskMaster". Студенти можуть використовувати додаток для управління своїми навчальними завданнями, планування навчального процесу та підготовки до екзаменів. Викладачі та адміністрація навчальних закладів можуть застосовувати "TaskMaster" для організації робочих процесів, планування занять та відстеження виконання навчальних планів.

Люди, які шукають інструменти для особистої продуктивності: існує велика група людей, які активно шукають інструменти для покращення особистої продуктивності. Ці користувачі часто експериментують з різними методами тайм-менеджменту, такими як метод "Pomodoro", GTD (Getting Things Done) та інші. "TaskMaster" пропонує функціонал, який може бути адаптований під різні методики управління завданнями, що робить його привабливим для цієї аудиторії. Наприклад, користувачі можуть створювати спеціальні списки завдань для різних методик, встановлювати пріоритети та сортувати завдання відповідно до обраного методу.

Цільова аудиторія веб-додатку "TaskMaster" є досить широкою і включає індивідуальних користувачів, професіоналів та фрілансерів, освітні установи та студентів, людей, які шукають інструменти для особистої продуктивності. Завдяки своїм функціональним можливостям, зручному інтерфейсу, "TaskMaster" здатний задовольнити потреби цих різних груп користувачів, надаючи їм ефективний інструмент для управління завданнями та підвищення продуктивності.

2 АНАЛІЗ ТА ШЛЯХИ РОЗВ'ЯЗКУ ПРОБЛЕМИ

2.1 Огляд та вибір технологій для розробки веб-додатку

У сучасній веб-розробці існує безліч технологій та інструментів, які дозволяють створювати функціональні та ефективні додатки. Вибір правильних технологій є критично важливим для успіху будь-якого проекту. У цьому розділі ми розглянемо найбільш популярні технології для фронтенд, бекенд розробки та управління базами даних, а також обґрунтуємо вибір стека технологій для розробки веб-додатку "TaskMaster".

Фронтенд розробка є важливою частиною створення веб-додатків, оскільки вона відповідає за користувацький інтерфейс і взаємодію з користувачами. Найбільш популярні фронтенд технології включають:

React: Це бібліотека JavaScript для створення користувацьких інтерфейсів, розроблена компанією Facebook. React відомий своєю гнучкістю, високою продуктивністю та компонентним підходом до розробки.

Angular: Це фреймворк JavaScript, розроблений компанією Google. Angular забезпечує повний набір інструментів для створення динамічних веб-додатків, включаючи двосторонню прив'язку даних та вбудовані рішення для управління станом додатку.

Vue.js: Це прогресивний фреймворк JavaScript, який надає можливість створювати інтуїтивно зрозумілі та легко масштабовані інтерфейси. Vue.js поєднує в собі кращі риси React та Angular, що робить його популярним серед розробників.

Переваги та недоліки

React: Переваги включають простоту використання, високу продуктивність завдяки віртуальному DOM, великий набір сторонніх бібліотек та активну спільноту. Недоліки можуть включати необхідність додаткових бібліотек для повноцінної розробки.

Angular: Переваги включають повний набір інструментів для розробки, вбудовані рішення для прив'язки даних та управління станом. Недоліки включають високу складність і круту криву навчання.

Vue.js: Переваги включають простоту та легкість освоєння, гнучкість і можливість поступового впровадження у проекти. Недоліки включають меншу кількість готових рішень у порівнянні з React та Angular.

Бекенд розробка відповідає за серверну логіку, обробку запитів та взаємодію з базою даних. Найпопулярніші бекенд технології включають:

Node.js: Це середовище виконання JavaScript, яке дозволяє виконувати JavaScript-код на сервері. Node.js відомий своєю високою продуктивністю та здатністю обробляти велику кількість одночасних запитів завдяки неблокуючій моделі вводу/виводу.

Django: Це фреймворк для веб-розробки на мові Python, який забезпечує швидку розробку та простоту використання. Django включає в себе багато готових рішень для реалізації різних завдань.

Ruby on Rails: Це фреймворк для веб-розробки на мові Ruby, який надає можливість швидкого створення додатків завдяки високому рівню абстракції та великій кількості готових рішень.

Переваги та недоліки

Node.js: Переваги включають високу продуктивність, можливість використання одного і того ж коду на клієнті та сервері, а також велику кількість модулів та пакетів. Недоліки включають складність управління асинхронністю.

Django: Переваги включають швидкість розробки, простоту використання та наявність великої кількості вбудованих інструментів. Недоліки включають високу вагу фреймворка і відносно низьку продуктивність.

Ruby on Rails: Переваги включають швидкість розробки, високий рівень абстракції та велика кількість готових рішень. Недоліки включають складність масштабування та високу вагу фреймворка.

Управління базами даних є важливим аспектом розробки будь-якого веб-додатку, оскільки воно забезпечує збереження та доступ до даних. Найпопулярніші технології управління базами даних включають:

MongoDB: Це документно-орієнтована база даних NoSQL, яка забезпечує високу гнучкість та масштабованість. MongoDB зберігає дані у форматі JSON-подібних документів.

MySQL: Це реляційна база даних, яка забезпечує високий рівень надійності та продуктивності. MySQL є однією з найпоширеніших баз даних у світі.

PostgreSQL: Це реляційна база даних з відкритим кодом, яка забезпечує високу продуктивність та підтримку розширених функцій.

Переваги та недоліки

MongoDB: Переваги включають високу гнучкість, можливість горизонтального масштабування та простоту використання. Недоліки включають відсутність підтримки складних транзакцій та обмежені можливості для складних запитів.

MySQL: Переваги включають високу продуктивність, надійність та підтримку складних транзакцій. Недоліки включають обмежені можливості горизонтального масштабування та складність роботи з великими обсягами даних.

PostgreSQL: Переваги включають високу продуктивність, підтримку розширених функцій та можливість обробки складних запитів. Недоліки включають відносну складність налаштування та адміністрування.

Після аналізу основних технологій для фронтенд, бекенд розробки та управління базами даних, для розробки веб-додатку "TaskMaster" було обрано наступний стек основних технологій:

React для фронтенд розробки, Node.js для бекенд розробкиб, MongoDB для управління базою даних.

Цей вибір був зроблений з урахуванням переваг цих технологій над їх аналогами і моїм рівнем володіння і знань над ними. Далі ми більш детально розглянемо обраний стек технологій і менш важливі, допоміжні технології які я активно використовував.

2.1.1 Огляд React, Node.js, MongoDB

React – це популярна бібліотека для розробки користувацьких інтерфейсів, створена компанією Facebook. Вона використовується для побудови динамічних і інтерактивних веб-додатків. React є відомим завдяки своїй простоті, гнучкості та високій продуктивності. Ось детальний огляд основних аспектів та переваг використання React у розробці веб-додатку "TaskMaster".[1]

Основні концепції React

Компонентний підхід: однією з основних концепцій React є компонентний підхід до розробки. Компоненти в React – це незалежні, повторно використовувані шматки коду, які відповідають за рендеринг частини користувацького інтерфейсу. Кожен компонент може мати свій власний стан і поведінку. Це дозволяє розробникам створювати складні інтерфейси, розбиваючи їх на дрібніші, керовані частини.

Віртуальний DOM: React використовує концепцію віртуального DOM (Document Object Model) для підвищення продуктивності додатку. Віртуальний DOM – це легка копія реального DOM, яка зберігається в пам'яті. Коли стан компонента змінюється, React оновлює віртуальний DOM, порівнює його з попередньою версією (цей процес називається "дифінг") і визначає мінімальний набір змін, який необхідно внести в реальний DOM. Це значно покращує продуктивність, оскільки мінімізує кількість операцій з реальним DOM.

Односпрямований потік даних: ще однією важливою концепцією React є односпрямований потік даних. Це означає, що дані передаються від батьківських компонентів до дочірніх через властивості (props). Такий підхід робить структуру додатку передбачуваною і спрощує відстеження змін у стані компонентів.

Переваги використання React

Висока продуктивність: завдяки використанню віртуального DOM і оптимізації оновлень, React забезпечує високу продуктивність додатків. Це особливо важливо для великих та складних веб-додатків, які потребують швидкого рендерингу та реагування на дії користувачів.

Повторне використання компонентів: компонентний підхід React дозволяє повторно використовувати компоненти в різних частинах додатку, що значно спрощує розробку і підтримку коду. Це також сприяє зменшенню дублювання коду і покращенню його якості.

Активна спільнота та екосистема: React має велику і активну спільноту розробників, яка постійно розвиває та вдосконалює бібліотеку. Існує багато сторонніх бібліотек і інструментів, які інтегруються з React і розширюють його можливості. Це включає в себе такі інструменти, як React Router для маршрутизації, Redux для управління станом додатку, а також численні компоненти інтерфейсу користувача.

Легка інтеграція з іншими технологіями: React легко інтегрується з іншими бібліотеками і фреймворками. Це дозволяє використовувати його разом з різними інструментами для створення повноцінних веб-додатків. Наприклад, React може бути інтегрований з Node.js для створення серверної частини додатку, або з GraphQL для оптимізації запитів до сервера.

Використання React у "TaskMaster"

У контексті розробки "TaskMaster", React забезпечує декілька важливих переваг:

Інтерактивність: React дозволяє створювати динамічні і інтерактивні інтерфейси, що робить додаток зручним і привабливим для користувачів.

Продуктивність: Завдяки віртуальному DOM і оптимізації оновлень, React забезпечує високу продуктивність додатку навіть при великій кількості завдань і користувачів.

Компонентний підхід: Розбиття інтерфейсу на компоненти спрощує розробку, тестування і підтримку коду. Це також дозволяє легко додавати нові функції або змінювати існуючі.

Масштабованість: React дозволяє створювати масштабовані додатки, які можуть розширюватися разом із зростанням потреб користувачів.

Інтеграція: React легко інтегрується з іншими технологіями, що дозволяє використовувати його в поєднанні з іншими інструментами для створення повноцінних веб-додатків.

Вибір React для розробки фронтенд-частини веб-додатку "TaskMaster" обґрунтований його високою продуктивністю, гнучкістю, зручністю використання та активною спільнотою. Ці переваги роблять React ідеальним вибором для створення динамічних, інтерактивних і масштабованих веб-додатків, що відповідають сучасним стандартам розробки.

Node.js – це середовище виконання JavaScript на сервері, створене на основі движка V8 від Google Chrome. Node.js дозволяє виконувати JavaScript-код поза браузером, що відкриває нові можливості для розробки веб-додатків, зокрема для серверної частини. Node.js відомий своєю високою продуктивністю, здатністю обробляти велику кількість одночасних запитів та неблокуючою моделлю вводу/виводу. Ось детальний огляд основних аспектів та переваг використання Node.js у розробці веб-додатку "TaskMaster".[2]

Основні концепції Node.js

Подієво-орієнтована архітектура: однією з основних концепцій Node.js є подієво-орієнтована архітектура. Node.js використовує неблокуючу модель вводу/виводу, що дозволяє обробляти велику кількість одночасних запитів без необхідності створення нових потоків для кожного запиту. Це досягається за рахунок використання подій і зворотних викликів (callbacks), що дозволяє серверу продовжувати обробку інших запитів під час виконання тривалих операцій вводу/виводу.

Єдиний стек технологій: Node.js дозволяє використовувати JavaScript як на клієнтській, так і на серверній частині додатку, що спрощує розробку та підтримку коду. Це означає, що розробники можуть використовувати одну і ту ж мову програмування для створення всього додатку, що підвищує ефективність команди та зменшує час на освоєння нових технологій.

Пакетний менеджер npm: Node.js постачається з вбудованим пакетним менеджером npm (Node Package Manager), який є одним з найбільших і

найактивніших репозиторіїв пакетів у світі. npm дозволяє легко встановлювати та керувати сторонніми бібліотеками та модулями, що значно спрощує розробку додатків. Велика кількість доступних пакетів дозволяє швидко додавати нові функціональності та інтегрувати додаток з іншими сервісами.

Переваги використання Node.js

Висока продуктивність: завдяки неблокуючій моделі вводу/виводу та використанню подієво-орієнтованої архітектури, Node.js забезпечує високу продуктивність та здатність обробляти велику кількість одночасних запитів. Це особливо важливо для веб-додатків, які повинні працювати під високим навантаженням та забезпечувати швидкий відгук на дії користувачів.

Масштабованість: Node.js дозволяє легко масштабувати додатки горизонтально, додаючи нові сервери до кластеру. Завдяки подієво-орієнтованій архітектурі, додатки на Node.js можуть ефективно використовувати ресурси системи та забезпечувати високу продуктивність навіть при зростанні кількості користувачів та запитів.

Швидкість розробки: Node.js дозволяє швидко створювати прототипи та готові додатки завдяки використанню JavaScript на обох кінцях розробки (клієнтська та серверна частина), а також завдяки великій кількості доступних пакетів у npm. Це значно скорочує час розробки та дозволяє швидше впроваджувати нові функціональності.

Спільнота та підтримка: Node.js має велику та активну спільноту розробників, що забезпечує постійне вдосконалення платформи та наявність великої кількості ресурсів для навчання та підтримки. Це включає документацію, онлайн-курси, форуми та конференції, де розробники можуть обмінюватися досвідом та знайомитися з новими інструментами та підходами.

Використання Node.js у "TaskMaster"

У контексті розробки "TaskMaster", Node.js є чудовим вибором оскільки він має декілька важливих переваг:

Обробка великої кількості запитів: Завдяки неблокуючій моделі вводу/виводу, Node.js може ефективно обробляти велику кількість одночасних запитів, що робить його ідеальним для високонавантажених додатків.

Єдиний стек технологій: Використання JavaScript на обох кінцях (фронтенд та бекенд) спрощує розробку та підтримку коду, підвищує ефективність команди та зменшує час на освоєння нових технологій.

Швидкий розвиток: Завдяки великій кількості доступних пакетів та модулів у npm, розробники можуть швидко додавати нові функціональності та інтегрувати додаток з іншими сервісами.

Масштабованість: Node.js дозволяє легко масштабувати додаток, додаючи нові сервери до кластеру, що забезпечує високу продуктивність навіть при зростанні кількості користувачів та запитів.

Активна спільнота: Велика та активна спільнота розробників Node.js забезпечує постійну підтримку та вдосконалення платформи, що дозволяє швидко вирішувати проблеми та знайомитися з новими інструментами та підходами.

Вибір Node.js для розробки серверної частини веб-додатку "TaskMaster" обґрунтований його високою продуктивністю, здатністю обробляти велику кількість одночасних запитів, масштабованістю та єдиним стеком технологій з фронтенд-частиною. Ці переваги роблять Node.js ідеальним вибором для створення високонавантажених, масштабованих та продуктивних веб-додатків, що відповідають сучасним стандартам розробки.

MongoDB – це документно-орієнтована база даних NoSQL, створена для зберігання великих обсягів даних у зручному форматі JSON-подібних документів. MongoDB є одним з найпопулярніших рішень у сфері NoSQL баз даних завдяки своїй гнучкості, масштабованості та продуктивності.[3] Ось детальний огляд основних аспектів та переваг використання MongoDB у розробці веб-додатку "TaskMaster".

Основні концепції MongoDB

Документно-орієнтована модель: однією з основних концепцій MongoDB є документно-орієнтована модель зберігання даних. Дані зберігаються у вигляді

документів у форматі BSON (бінарне представлення JSON), що дозволяє легко працювати з вкладеними структурами даних. Документи об'єднуються у колекції, що є аналогом таблиць у реляційних базах даних.

Гнучка схема: MongoDB не має жорстко визначеної схеми, що дозволяє зберігати документи з різною структурою в одній колекції. Це забезпечує високу гнучкість при розробці додатків, оскільки структуру даних можна змінювати без необхідності модифікації всієї бази даних.

Масштабованість: MongoDB підтримує горизонтальне масштабування за допомогою шардингу (розподілу даних між кількома серверами). Це дозволяє обробляти великі обсяги даних та забезпечувати високу продуктивність при зростанні кількості користувачів та запитів.

Інтеграція з іншими технологіями: MongoDB легко інтегрується з іншими інструментами та фреймворками, такими як Node.js, що дозволяє створювати повноцінні додатки, використовуючи єдиний стек технологій. MongoDB також підтримує вбудовані індекси та агрегації, що спрощує обробку та аналіз даних.

Переваги використання MongoDB

Висока продуктивність: завдяки використанню документно-орієнтованої моделі та горизонтального масштабування, MongoDB забезпечує високу продуктивність при обробці великих обсягів даних. Це дозволяє швидко виконувати складні запити та аналізувати дані в реальному часі.

Гнучкість: гнучка схема MongoDB дозволяє легко змінювати структуру даних без необхідності переробляти всю базу даних. Це особливо корисно при розробці додатків, де вимоги до даних можуть змінюватися з часом.

Масштабованість: підтримка горизонтального масштабування за допомогою шардингу дозволяє MongoDB обробляти великі обсяги даних та забезпечувати високу продуктивність при зростанні навантаження. Це робить MongoDB ідеальним рішенням для додатків, які повинні масштабуватися разом із зростанням кількості користувачів та даних.

Інтеграція та підтримка: MongoDB має активну спільноту та підтримку, що забезпечує постійне вдосконалення та розвиток платформи. Існує багато

інструментів та бібліотек, які інтегруються з MongoDB, що спрощує розробку та підтримку додатків.

Використання MongoDB у "TaskMaster"

У контексті розробки "TaskMaster", MongoDB забезпечує декілька важливих переваг:

Гнучка схема: можливість зберігати документи з різною структурою дозволяє легко адаптувати базу даних до змінних вимог додатку.

Висока продуктивність: завдяки документно-орієнтованій моделі та підтримці індексів, MongoDB забезпечує швидке виконання запитів та обробку даних.

Масштабованість: підтримка горизонтального масштабування дозволяє MongoDB ефективно обробляти великі обсяги даних та забезпечувати високу продуктивність при зростанні кількості користувачів.

Інтеграція з Node.js: MongoDB легко інтегрується з Node.js, що дозволяє використовувати єдиний стек технологій для розробки серверної частини додатку.

Активна спільнота та підтримка: велика кількість ресурсів, інструментів та бібліотек спрощує розробку та підтримку додатку.

Вибір MongoDB для управління базою даних у веб-додатку "TaskMaster" обґрунтований її високою продуктивністю, гнучкістю, масштабованістю та легкістю інтеграції з іншими технологіями. Ці переваги роблять MongoDB ідеальним вибором для створення сучасних, масштабованих та ефективних веб-додатків, що відповідають вимогам зростаючого ринку та потребам користувачів.

2.1.2 Інші технології

Окрім основних технологій, таких як React, Node.js та MongoDB, у розробці веб-додатку "TaskMaster" використовуються й інші важливі інструменти, що забезпечують високу продуктивність, зручність та гнучкість додатку. Серед них особливо виділяються TailwindCSS, React Router DOM, Express, Redux, Redux Toolkit та MongoDB Atlas. Нижче наведено детальний огляд цих технологій та їхні переваги для розробки проекту.

TailwindCSS – це утилітарний CSS фреймворк, який надає велику кількість класів для створення стилів без написання власного CSS. TailwindCSS відрізняється від традиційних CSS фреймворків, таких як Bootstrap, своєю гнучкістю та можливістю налаштування під конкретні потреби проекту.[4]

Основні концепції TailwindCSS

Утилітарні класи: TailwindCSS використовує утилітарні класи для створення стилів. Це означає, що кожен клас відповідає за один конкретний стиль, наприклад, text-center для центрування тексту або bg-blue-500 для встановлення синього кольору фону. Це дозволяє швидко створювати складні дизайни без написання власного CSS.

Повна кастомізація: TailwindCSS дозволяє повністю налаштовувати теми, кольори, розміри та інші параметри через конфігураційний файл. Це забезпечує високу гнучкість та можливість створення унікальних дизайнів, які відповідають потребам проекту.

Переваги використання TailwindCSS

Швидкість розробки: Використання утилітарних класів значно прискорює процес розробки, оскільки розробники можуть швидко створювати стилі без необхідності писати власний CSS.

Зменшення розміру CSS файлів: TailwindCSS надає інструменти для видалення невикористаних класів у фінальній збірці, що зменшує розмір CSS файлів та покращує продуктивність додатку.

Гнучкість та кастомізація: Можливість повного налаштування стилів через конфігураційний файл дозволяє створювати унікальні та адаптивні дизайни, які відповідають вимогам проекту.

React Router DOM – це бібліотека для маршрутизації у додатках на базі React. Вона забезпечує можливість створення динамічних односторінкових додатків (SPA) з багатосторінковою навігацією, дозволяючи користувачам переміщуватися між різними частинами додатку без перезавантаження сторінки.[5]

Основні концепції React Router DOM

Декларативна маршрутизація: React Router DOM використовує декларативний підхід до визначення маршрутів, що робить код більш зрозумілим та легким для підтримки. Розробники визначають маршрути та компоненти, які повинні відображатися при переході за певними URL.

Динамічна маршрутизація: React Router DOM дозволяє створювати динамічні маршрути, які можуть змінюватися залежно від стану додатку або дій користувача. Це забезпечує гнучкість та дозволяє створювати складні навігаційні структури.

Переваги використання React Router DOM

Проста інтеграція з React: React Router DOM легко інтегрується з додатками на базі React, що дозволяє швидко налаштувати маршрутизацію та забезпечити зручну навігацію у додатку.

Декларативний підхід: Декларативний підхід до визначення маршрутів робить код більш зрозумілим та легким для підтримки, що сприяє покращенню якості коду та зниженню кількості помилок.

Підтримка динамічних маршрутів: Можливість створення динамічних маршрутів дозволяє створювати більш складні та гнучкі навігаційні структури, що покращує користувацький досвід.

Express – це мінімалістичний та гнучкий веб-фреймворк для Node.js, який спрощує розробку серверних додатків та API. Express забезпечує простий інтерфейс для створення маршрутів, обробки запитів та відповіді, а також підтримку проміжних обробників (middleware).[6]

Основні концепції Express

Маршрутизація: Express дозволяє легко створювати маршрути для обробки HTTP-запитів. Це дозволяє розробникам визначати різні маршрути для обробки різних типів запитів та відповіді на них.

Проміжні обробники (middleware): Express підтримує використання проміжних обробників для обробки запитів на різних етапах їхнього проходження через додаток. Це дозволяє розділяти логіку обробки запитів на окремі компоненти, що робить код більш модульним та зручним для підтримки.

Переваги використання Express

Простота використання: Express має простий і зрозумілий інтерфейс, що робить його легким для освоєння та використання. Це дозволяє розробникам швидко створювати серверні додатки та API.

Гнучкість: Express забезпечує високу гнучкість завдяки підтримці проміжних обробників, що дозволяє розробникам легко розширювати та налаштовувати додаток під конкретні потреби проекту.

*Активна спільнота та підтримк*а: Express має велику та активну спільноту, що забезпечує постійну підтримку та розвиток фреймворку. Існує багато ресурсів, документації та бібліотек, які допомагають розробникам швидко вирішувати проблеми та впроваджувати нові функціональності.

Redux – це популярна бібліотека для управління станом додатків JavaScript, особливо корисна в поєднанні з React. Вона дозволяє централізовано зберігати стан додатка, роблячи його більш передбачуваним і полегшуючи відстеження змін. Основна ідея Redux полягає в тому, що весь стан додатка зберігається в одному об'єкті, який називається стор (store).[7]

Єдиний джерело істини: Весь стан додатка зберігається в одному об'єкті, що дозволяє легко відслідковувати зміни і знижує ймовірність помилок.

Стан тільки для читання: Стан додатка не можна змінити безпосередньо. Замість цього, для внесення змін використовуються дії (actions), які описують, що має статися.

Зміни через чисті функції: Для обробки дій і внесення змін до стану використовуються ред'юсери (reducers) – чисті функції, які отримують поточний стан і дію, а повертають новий стан.

Переваги використання Redux

Передбачуваність: Централізоване управління станом робить поведінку додатка передбачуваною і полегшує налагодження.

Легка інтеграція: Redux легко інтегрується з різними бібліотеками та фреймворками, особливо з React.

Інструменти розробки: Redux DevTools дозволяють відстежувати дії, перевіряти стан і відновлювати попередні стани, що значно спрощує процес розробки та налагодження.

Redux Toolkit – це офіційний пакет для ефективного використання Redux. Він спрощує і прискорює розробку за допомогою надання набору інструментів та шаблонів для створення дій, ред'юсерів та стору.[8]

Основні концепції Redux Toolkit

configureStore(): Спрощує створення стору з налаштованими середовищами розробки, такими як redux-thunk для асинхронних операцій та інструменти розробки.

createSlice(): Дозволяє створювати ред'юсери та дії одночасно, спрощуючи структуру та зменшуючи обсяг коду.

createAsyncThunk(): Полегшує роботу з асинхронними операціями, автоматично створюючи дії для різних етапів запиту (запит, успіх, невдача).

Переваги використання Redux Toolkit

Простота: Зменшує кількість коду та складність конфігурації, спрощуючи процес створення Redux-додатків.

Стандартизація: Забезпечує єдиний підхід до роботи з Redux, що полегшує читання та підтримку коду.

Підтримка TypeScript: Redux Toolkit має вбудовану підтримку TypeScript, що дозволяє використовувати статичну типізацію для покращення якості коду.

Використання Redux та Redux Toolkit у "TaskMaster"

У контексті розробки "TaskMaster", використання Redux та Redux Toolkit забезпечує декілька важливих переваг:

Централізоване управління станом: Використання Redux дозволяє централізовано зберігати та управляти станом додатку, що робить його більш передбачуваним і полегшує відстеження змін.

Зменшення складності: Redux Toolkit спрощує створення дій, ред'юсерів та стору, що зменшує обсяг коду та підвищує ефективність розробки.

Покращення процесу розробки: Використання Redux DevTools та інших інструментів для розробки полегшує налагодження та тестування додатку, підвищуючи загальну якість коду.

MongoDB Atlas – це хмарна служба баз даних, яка надає повністю керовану MongoDB інфраструктуру. Atlas забезпечує автоматичне масштабування, резервне копіювання, моніторинг та безпеку, що спрощує управління базами даних та забезпечує високу надійність і продуктивність.

Основні концепції MongoDB Atlas

Хмарна інфраструктура: MongoDB Atlas надає хмарну інфраструктуру для управління базами даних, що дозволяє розробникам зосередитися на розробці додатків без необхідності керувати фізичними серверами.

Автоматичне масштабування: MongoDB Atlas автоматично масштабує базу даних відповідно до навантаження, що забезпечує високу продуктивність та надійність додатку.

Переваги використання MongoDB Atlas

Простота управління: MongoDB Atlas забезпечує повністю керовану інфраструктуру, що спрощує управління базами даних та забезпечує високу надійність і продуктивність.

Автоматичне масштабування: Завдяки автоматичному масштабуванню, MongoDB Atlas забезпечує високу продуктивність додатку навіть при зростанні кількості користувачів та даних.

Безпека та резервне копіювання: MongoDB Atlas забезпечує високу безпеку даних та автоматичне резервне копіювання, що гарантує збереження та захист даних.

У контексті розробки "TaskMaster", використання TailwindCSS, React Router DOM, Express та MongoDB Atlas забезпечує декілька важливих переваг:

Зручність та швидкість розробки: Використання утилітарних класів TailwindCSS та проміжних обробників Express дозволяє швидко створювати та налаштовувати стилі та серверні маршрути, що значно прискорює процес розробки.

Гнучка маршрутизація: React Router DOM забезпечує гнучку та зручну маршрутизацію у додатку, дозволяючи створювати динамічні маршрути та покращувати користувацький досвід.

Масштабованість та продуктивність: Використання MongoDB Atlas для управління базою даних забезпечує високу продуктивність та надійність додатку завдяки автоматичному масштабуванню та резервному копіюванню.

Інтеграція та безпека: MongoDB Atlas забезпечує високу безпеку даних та простоту інтеграції з іншими компонентами додатку, що робить його ідеальним вибором для сучасних веб-додатків.

Вибір TailwindCSS, React Router DOM, Express, MongoDB Atlas, Redux та Redux Toolkit для розробки веб-додатку "TaskMaster" обґрунтований їхньою зручністю, гнучкістю та здатністю покращувати продуктивність додатку. Ці технології забезпечують швидку розробку, зручну маршрутизацію, високу продуктивність та надійність, що робить їх ідеальним вибором для створення сучасних та ефективних веб-додатків. Використання Redux та Redux Toolkit дозволяє централізовано управляти станом додатку, зменшуючи складність коду та підвищуючи ефективність розробки. Ці інструменти забезпечують передбачуваність поведінки додатку та спрощують налагодження, що значно покращує загальну якість та зручність у підтримці коду.

2.2 Процес розробки

Після детального аналізу технологій та вибору інструментів для розробки веб-додатку "TaskMaster", я приступив до створення самого додатку. Процес розробки включає декілька ключових етапів, кожен з яких є важливим для досягнення кінцевого результату. Основними етапами розробки це створення та налаштування проекту, розробка клієнтської частини та розробка серверної частини.[9]

Слід зауважити, що розробка будь-якого програмного забезпечення потребує ретельного планування. Перш ніж розпочати безпосереднє написання коду, важливо визначити архітектуру додатку та встановити основні цілі та вимоги до

функціональності. Для цього важливо проаналізувати схожі роботи більш досвідчених програмістів. Незамінним інструментом для вирішення цих проблем був штучний інтелект, зокрема ChatGPT.

2.2.1 Створення та налаштування проекту

Процес створення та налаштування проекту "TaskMaster" розпочався з підготовки середовища розробки. Я обрав Visual Studio Code (VSCode) як основний інструмент для написання коду завдяки його зручності та підтримці великої кількості розширень, що підвищують продуктивність.

Для створення проекту React я використав команду npx create-react-app client, яка автоматично налаштовує середовище розробки з мінімальними налаштуваннями. Після чого я розділив структуру мого проекту на клієнтську і серверну частину. Детально розглянути структуру проекту можна на рисунку 2.1 .

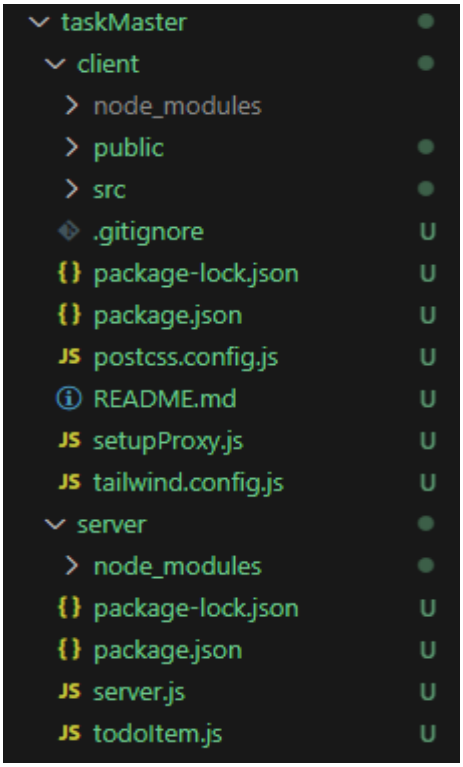


Рисунок 2.1 – Структура веб-додатку Task Master

Після створення основної структури проекту я підключив всі необхідні технології за допомогою npm. Нижче наведено команди для встановлення основних технологій.

Підключення TailwindCSS: Для підключення TailwindCSS до клієнтської частини проекту я увів в командний рядок наступну команду:

npm install tailwindcss postcss autoprefixer

npx tailwindcss init

Після недовгого процесу завантаження всі залежності TailwindCSS зберігаються у папці node_modules. Далі необхідно імпортувати необхідні файли у src/index.css .

Підключення React Router DOM: Для маршрутизації у клієнтській частині проекту я використав React Router DOM, щоб підключити його потрібно ввести команду npm install react-router-dom у командний рядок. За схожим алгоритмом підключаємо і настуані технології:

Підключення Express та MongoDB - npm install express mongoose dotenv

Підключення Redux та Redux Toolkit - Toolkit:npm install @reduxjs/toolkit react-redux

Щоб дізнатись які залежності встановлені і перевіріти поточну версію, потрібно зайти у файл package.json рисунок 2.2 де ви знайдете вю необхідну інформацію.

Слід зауважити що просто встановити залежності недостатньо, для кожної технології потрібен свій підхід. Більш детально це розглянемо у процесі розробки.

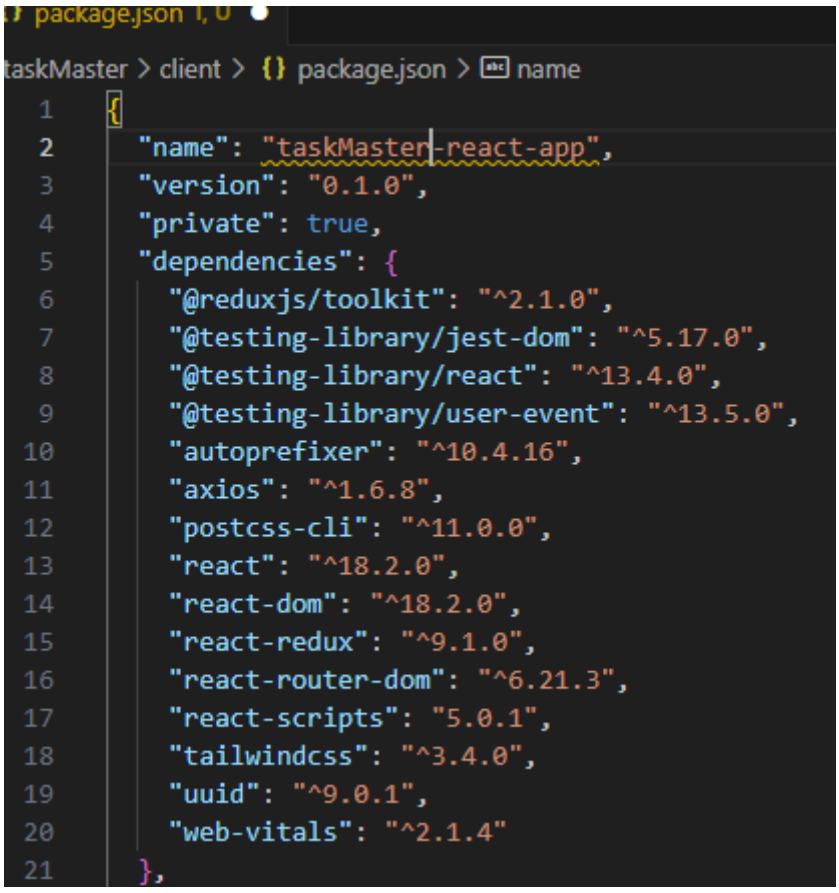


Рисунок 2.2 – Перелік встановлених залежностей

2.2.2 Розробка клієнтської частини

Після налаштування середовища розробки та створення проекту, я перейшов до розробки клієнтської частини додатку "TaskMaster". Під час написання коду саме клієнтська частина виявилась найбільш трудомістким і часозатратним етапом. Основне завдання цього етапу це створення логіки додатку і зовнішній вигляд[10], також необхідно визначети який компонент за що відповідає та успішно все це обєднати у цілий функціональний додаток.

Архітектура клієнтської частини: архітектура клієнтської частини базується на компонентному підході, що дозволяє розділити інтерфейс на незалежні, повторно використовувані компоненти. Це значно спрощує розробку, тестування та підтримку коду.[11],[12],[13]

Основними компонентами клієнтської частини являються: Header, TodoInput, TodoList, TodoPage.

Компонент Header: компонент Header є одним із найважливіших компонентів додатку, оскільки він відповідає за відображення заголовка та навігації і кінця додатку(footer) де заходяться контакти розробника. Компонент Header не несе ніякої логічної нагрузки. Основні його функції це:

Навігація: Він містить навігаційне меню з посиланням на сторінку зі списком завдань (TodoList). Для цього використовується компонент Link з бібліотеки react-router-dom.

Декор: Стилі компоненту надають йому привабливий вигляд. Використовуються CSS класи для налаштування кольорів, розмірів та тіней елементів. Головний елемент компоненту містить заголовок "TaskMaster" та навігаційну кнопку (Рисунок 2.3).

Footer: Компонент також включає футер з контактною інформацією, яка містить посилання на Telegram, GitHub та LinkedIn(Рисунок 2.4).

Рисунок 2.3 – Вигляд компонента Header

Рисунок 2.4 – Вигляд footer частини

Компонент TodoInput: компонент TodoInput є важливою частиною додатку "TaskMaster" (Рисунок 2.5). Він відповідає за введення нових завдань користувачем. Основні функці компонету це:

Введення завдань: Користувач вводить заголовок, опис завдання, дату та час дедлайну.

Пріоритезація: Користувач обирає пріоритет завдання з трьох можливих варіантів: A, B, C (Рисунок 2.6).

Сортування: Користувач може сортувати завдання за датою, алфавітом або пріоритетом(Рисунок 2.7).

Додавання завдання: Користувач додає завдання до списку завдань, натискаючи кнопку "Add task".

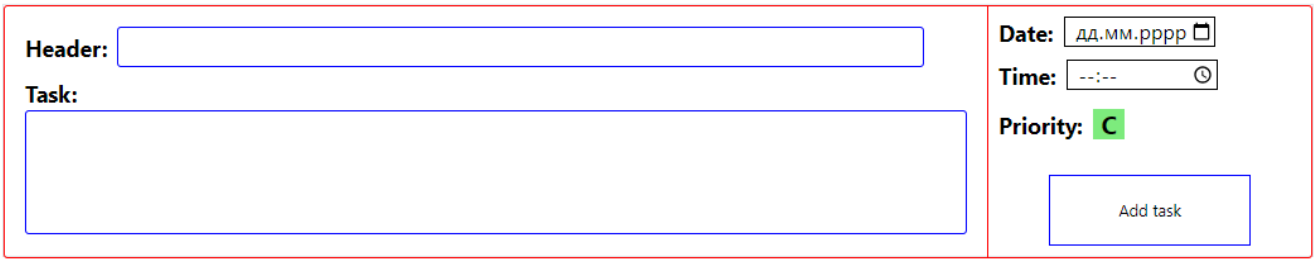


Рисунок 2.5 – Вигляд footer частини

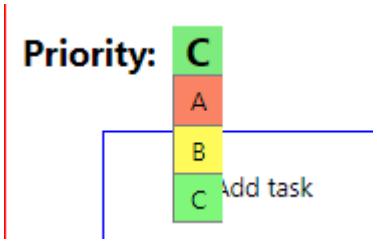


Рисунок 2.6 – Вибір пріорітетності задачі

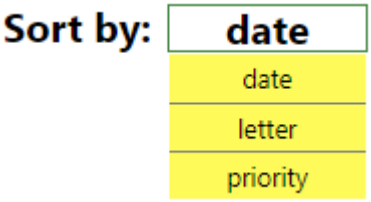


Рисунок 2.7 – Вибір методу сортування

Компонент TodoList: компонент TodoList є компонентом де відображаються завдання. Основні функції TodoList це:

Відображення списку завдань: Всі завдання відображаються у вигляді списку(Рисунок 2.8).

Редагування завдань: Користувач може редагувати завдання, натискаючи кнопку "Edit".

Видалення завдань: Користувач може видаляти завдання, натискаючи кнопку "Delete".

Рисунок 2.8 – Вигляд списку задач

Зміна статусу завдань: Користувач може змінювати статус завдання (виконано/не виконано) за допомогою чекбоксу.

Стани та функції компоненту

Стан компоненту: Компонент використовує стани для зберігання масиву завдань та статусу сортування. Я використовував реакт хуки useState і useSelector(Redux).[14]

Функції: Компонент має кілька функцій для обробки дій користувача, таких як зміна статусу завдання, редагування та видалення завдань. Для цього було використано хук useEffect і імпортовано функції із логічного файлу додатку.

Компонент TodoPage: компонент TodoPage є центральною частиною додатку "TaskMaster". Він об'єднує інші компоненти, такі як TodoInput і TodoList, забезпечуючи загальну структуру сторінки. Саме TodoPage імпортується в heder для

навігації. Сам компонент не несе ніякої логічності або функціоналу, TodoPage це велике вікно, в яке помістили всі інші функціональні компоненти todo.

todoStorySlice: файл todoStorySlice є центральною частиною логіки всього додатку "TaskMaster" (Рисунок 2.9). Він відповідає за управління станом додатку, реалізованим за допомогою Redux. Цей файл містить усі необхідні дії та редюсери, які забезпечують функціонування додатку, а також забезпечує зв'язок із сервером за допомогою fetch запитів.

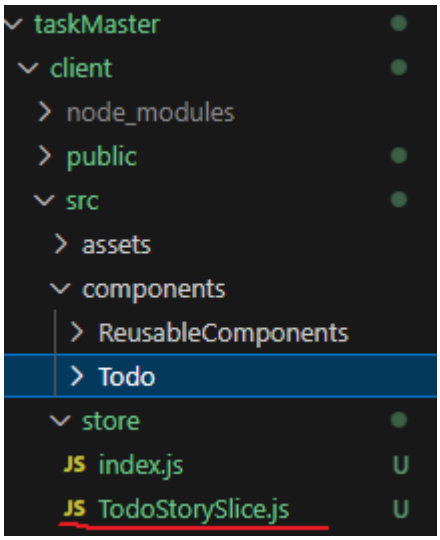


Рисунок 2.9 – Місцезнаходження файлу TodoStorySlice.js в архітектурі додатку

Основні функції todoStorySlice це:

Управління станом: файл todoStorySlice використовує Redux для управління станом додатку. Це включає в себе зберігання, оновлення та отримання даних про завдання.

Fetch запити: компонент відповідає за взаємодію з сервером за допомогою fetch запитів. Це включає в себе запити на отримання, додавання, редагування та видалення завдань.

todoStorySlice має наступні fetch запити:

Отримання завдань(GET): компонент використовує fetch запити для отримання списку завдань з сервера. Це забезпечує актуальність даних у додатку та дозволяє синхронізувати зміни між клієнтом і сервером.

Додавання завдань(POST): для додавання нового завдання до списку використовується відповідний fetch запит, який відправляє дані про нове завдання на сервер.

Редагування завдань(POST): компонент забезпечує можливість редагування існуючих завдань шляхом відправки відповідного fetch запиту з оновленими даними.

Видалення завдань(POST): для видалення завдання зі списку компонент відправляє на сервер fetch запит з інформацією про завдання, яке необхідно видалити.

Суміжний файл index.js: файл index.js є важливим для функціонування todoStorySlice, оскільки він забезпечує підключення Redux до додатку. У цьому файлі створюється store та підключається провайдер Redux, що дозволяє використовувати стан додатку у всіх компонентах(Рисунок 2.10).

Якщо говорити про логіку самого додатку, то вона є наступною – в todoStorySlice створюється пустий масив в initialState під назвою todoArray. Цей масив є головним і в ньому зберігаються всі завдання. Цей масив за допомогою fetch запитів відправляє і бере дані із бд, тому при перезапуску додатку всі дані зберігаються. За допомогою redux ми беремо дані із масива і динамічно показуємо їх в компоненті todoList де користувач взаємодіє із завданнями.

Рисунок 2.10 – Вміст файлу index.js для успішного підключення redux/toolkit

2.2.3 Розробка серверної частини

Для створення повноцінно функціонального додатку, сервер є однією із основних частин. Для більш легкого і ефективного написання коду для серверу я використовував такі залежності як axios і express[15],[16],[17], а для успішного підключення бд – mongodb і mongoose(Рисунок 2.11).

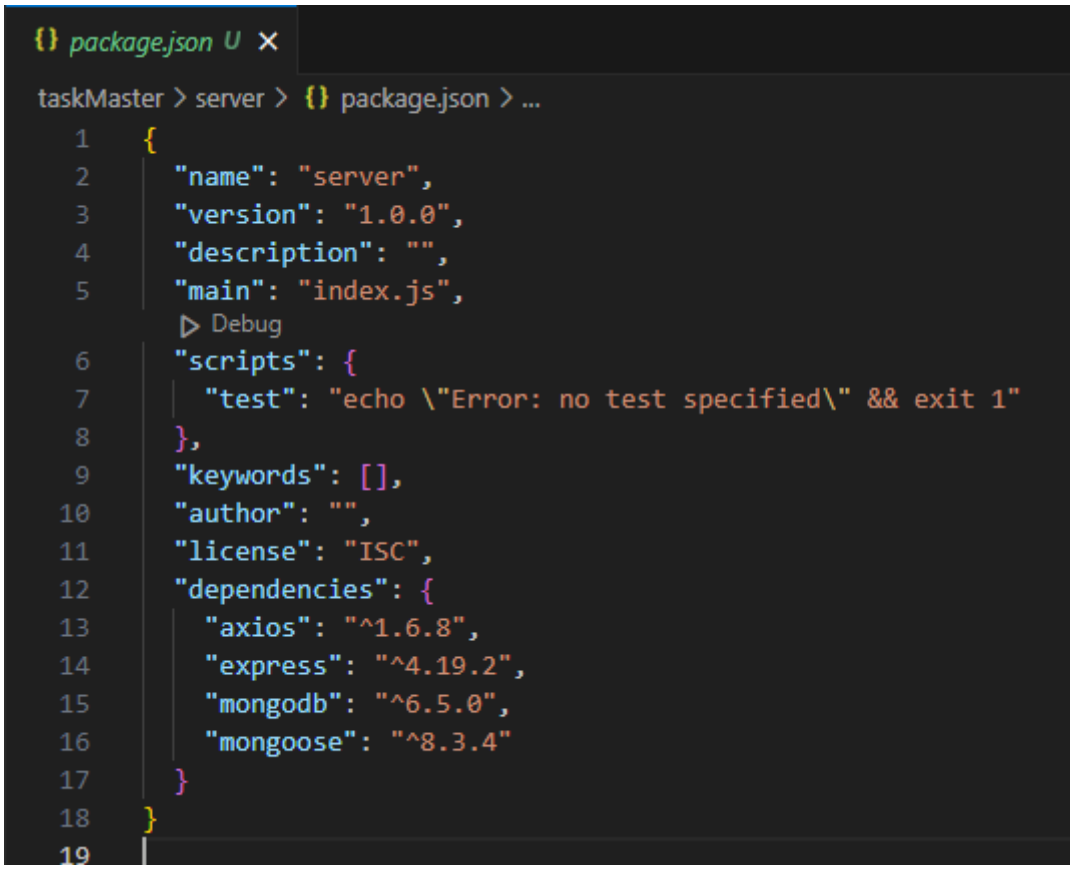


Рисунок 2.11 – Перелік всіх залежностей встановлених на серверній частині додатку та їх поточної версії

Головна задача сервера це приймати запити із клієнтської частини, обробляти їх і при необхідності брати та змінювати дані в бд. Для мене найважчим моментом при розробці сервера стало створення маршрутів між клієнтом і серверо[18]. Цю проблему я вирішив завдяки створенню файлу проксі setupProxy.js(Рисунок 2.12).

Рисунок 2.12 – Код файлу setupProxy.js

Завдякі файлу setupProxy.js ми отримали маршрут /api, після чого ми можемо індивідуально налаштувати маршрут для кожного запиту. У моєму випадку він виглядав наступним чином http://localhost:3000/api/TodoArray .

Файл server.js: файл server.js є основним файлом серверної частини додатку. Він виконує кілька ключових функцій:

Ініціалізація сервера: сервер ініціалізується за допомогою Express.js та налаштовується для прослуховування запитів на визначеному порту. У коді визначено змінну PORT, яка встановлюється з оточення або за замовчуванням використовується порт 3000. Сервер запускається на вказаному порту і починає прослуховувати запити.

Підключення до бази даних: за допомогою Mongoose сервер підключається до бази даних MongoDB Atlas. Підключення здійснюється через URL-з'єднання, який містить інформацію про користувача, пароль та назву бази даних. Після підключення до бази даних, обробляються можливі помилки з'єднання. Для запуску сервера потрібно прописати команду node server .js . Якщо з'єднання встановлено успішно, виводиться відповідне повідомлення у консолі(Рисунок 2.13).

Налаштування CORS: для забезпечення крос-доменних запитів сервер налаштовує CORS-заголовки. Це дозволяє клієнтським додаткам, що розташовані

на інших доменах, надсилати запити до сервера. Зокрема, дозволяються методи GET, POST, PUT, DELETE та встановлюються необхідні заголовки.[19]

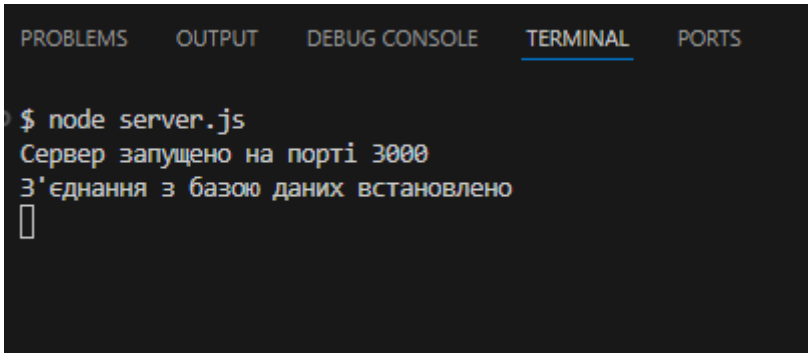


Рисунок 2.13 – Інформація із командного рядку про успішно запущений сервер і встановлення з’єднання з бд

Розбір тіла запиту: Express.js middleware дозволяє серверу обробляти JSON-дані, що надходять у запитах. Це забезпечує правильне розпізнавання та обробку даних, відправлених з клієнтської частини додатку.

Файл server.js реалізує два основні методи для роботи з масивом завдань:

POST /api/TodoArray: Метод отримує масив завдань від клієнта. Перш за все, очищається колекція завдань у базі даних, після чого нові завдання додаються до колекції. Це дозволяє підтримувати актуальний список завдань.

GET /api/TodoArray: Метод отримує масив завдань з бази даних та відправляє його клієнту. Це забезпечує синхронізацію даних між клієнтом і сервером, надаючи користувачам актуальну інформацію.

Файл todoItem.js: файл todoItem.js містить схему завдання та модель для роботи з базою даних. Він виконує такі функції:

Опис схеми завдання: визначає структуру документів у колекції TodoItem в базі даних.

Створення моделі: використовуючи схему, створює модель TodoItem, яка використовується для взаємодії з колекцією завдань у базі даних.

Опис схеми завдання: схема визначає поля, які має містити кожен документ у колекції завдань. Схема включає такі поля:

checkbox: Boolean, що вказує, чи виконано завдання.

creationTime: String, що зберігає час створення завдання.

deadline: Об'єкт, що містить дату та час дедлайну.

header: String, заголовок завдання.

id: String, унікальний ідентифікатор завдання.

priority: String, що вказує на пріоритет завдання.

task: String, опис завдання.

Створення моделі: модель TodoItem створюється на основі схеми та експортується для використання в інших частинах додатку. Модель забезпечує методи для взаємодії з колекцією завдань у базі даних, такі як створення, оновлення, видалення та отримання завдань(Рисунок 2.14).

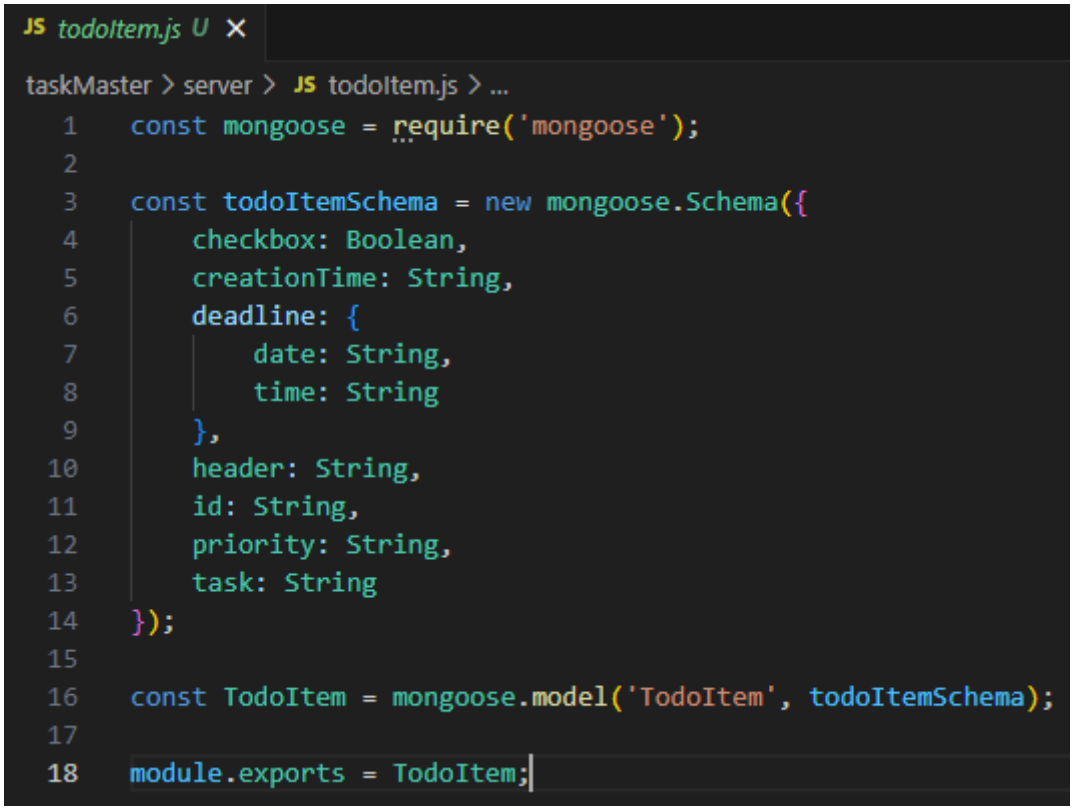


Рисунок 2.14 – Схема todo завдання яке зберігається в бд

MongoDB: MongoDB – це сучасна, потужна і високо масштабована база даних, яка відноситься до категорії NoSQL баз даних. Вона особливо популярна завдяки своїй гнучкості та можливості ефективно зберігати великі обсяги даних, що робить її ідеальним вибором для веб-додатків, таких як "TaskMaster"[20]. У моєму

проекті я використав хмарне рішення MongoDB Atlas, яке надає всі переваги MongoDB у хмарі, дозволяючи зосередитися на розробці додатку без необхідності управління інфраструктурою.

MongoDB Atlas забезпечує автоматичне масштабування, високу доступність та надійність, що є критичними для сучасних веб-додатків. Використовуючи MongoDB Atlas, ми можемо легко підключити наш додаток до бази даних, зберігати та отримувати дані, а також мати доступ до потужних інструментів для моніторингу та управління.

Для успішного підключення до бд ми маємо створити всю необхіду інфраструктуру. Я це виконував у вікні браузера за допомогою офіційного сайту від команди MongoDB.

Крок 1 – вхід на офіційному сайті(Рисунок 2.15).

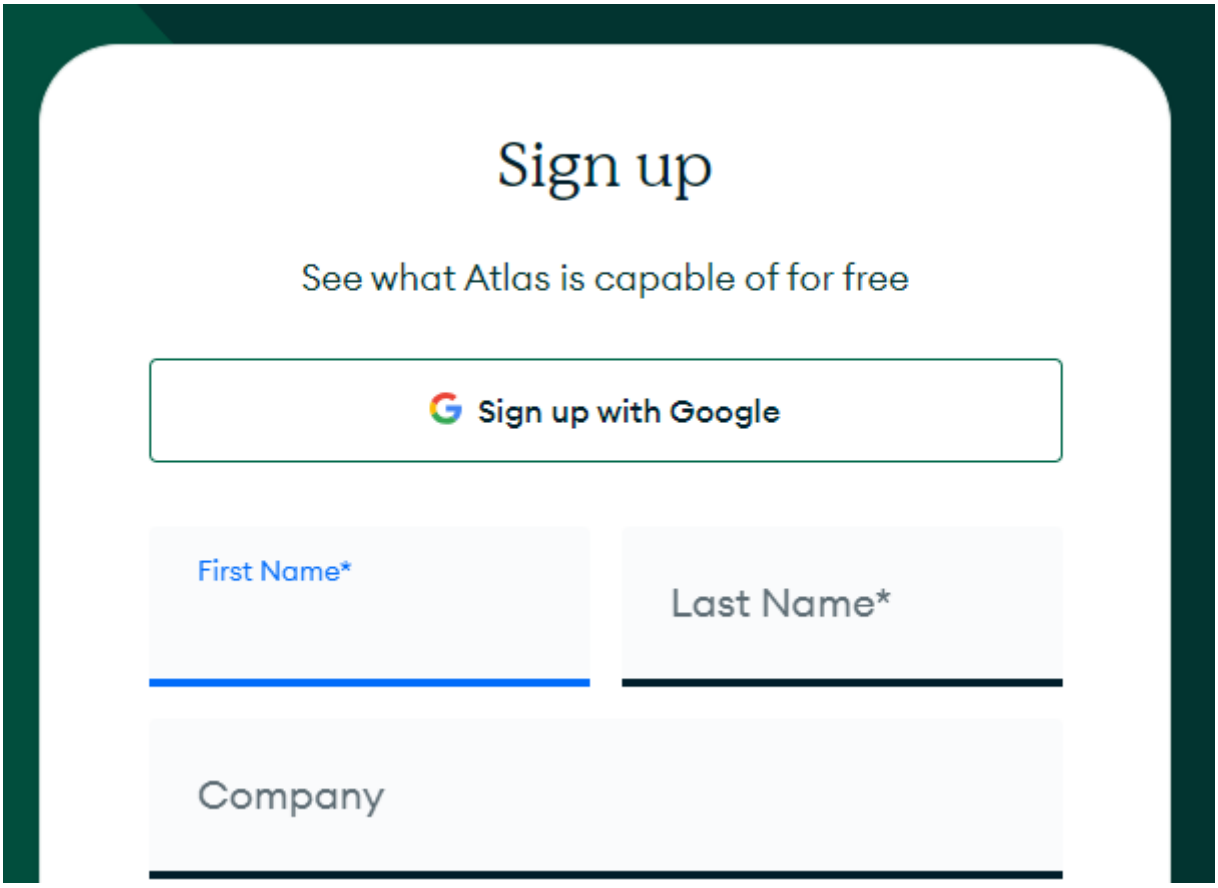


Рисунок 2.15 – Вікно входу в MongoDB Atlas

Я успішно увійшов натиснувши Sing up with Google. Після входу необхідно створити свій кластер і вірно його налаштувати. Для початку необхідно обрати хмарний сервіс із яким працювати, я обрав AWS(Рисунок 2.16).

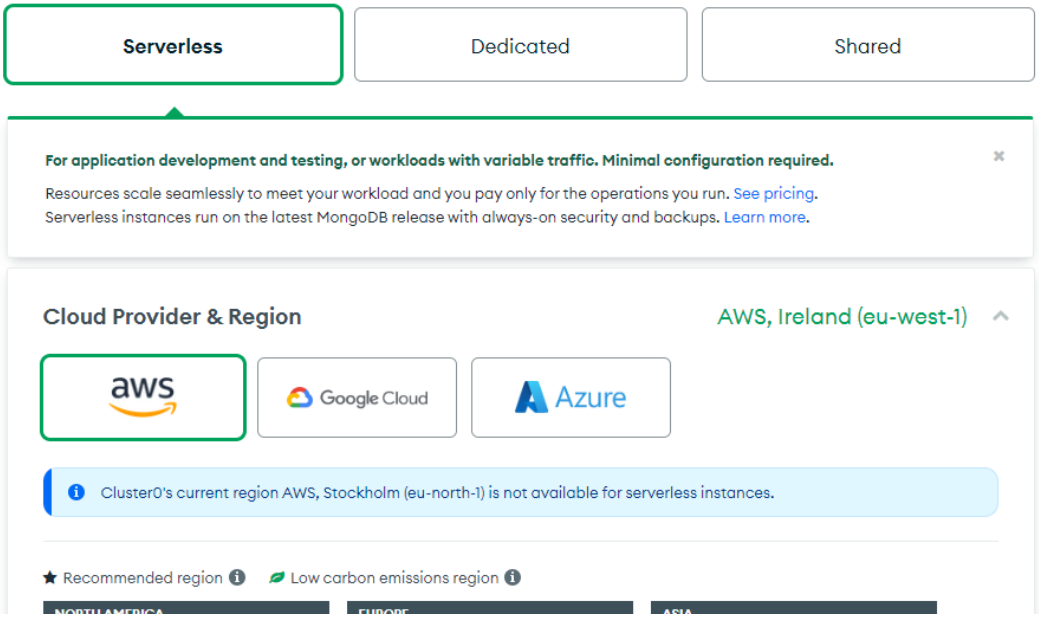


Рисунок 2.16 – Вибір хмарних сервісів

Наступний крок це обрання регіону де будуть зберігатись дані і вибір плану для резервного копіювання(Рисунок 2.17). Я обрав як регіон Ірландію і резервне копіювання раз в 72 години.

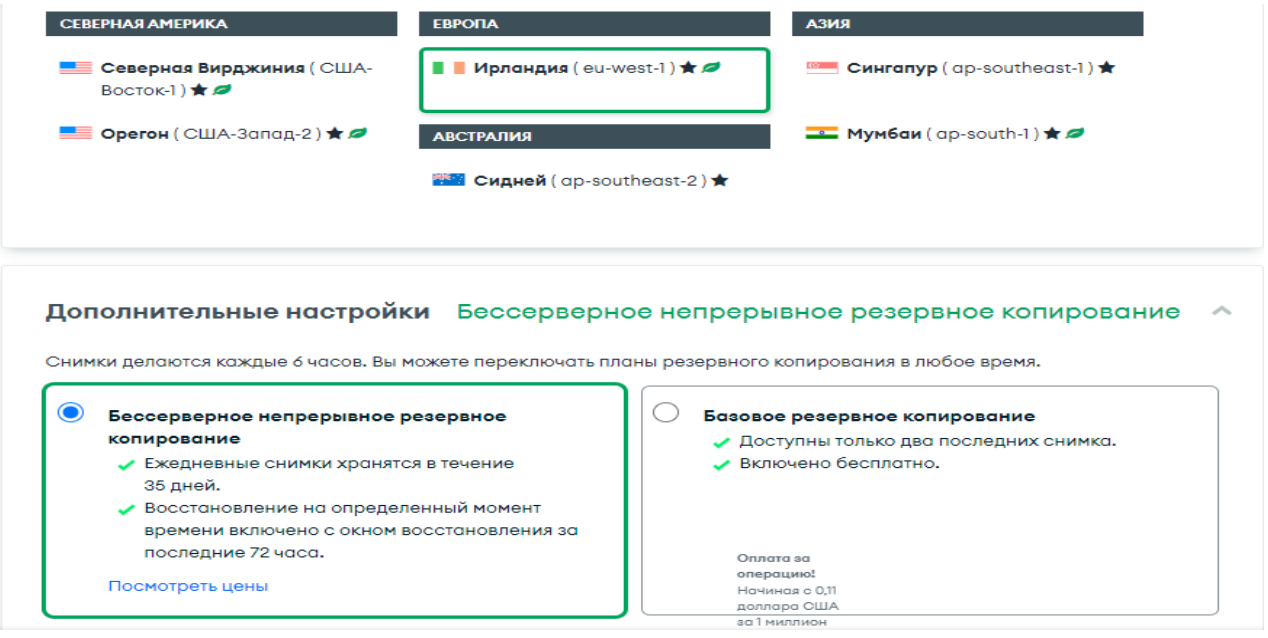


Рисунок 2.17 – Вибір регіону і плану резервного копіювання

Тепер необхідно налаштувати конфігурації кластера. Вони є різні і чім більшими та ресурсозатратними вони є тим більше вони коштують. Рисунок 2.18 У моєму випадку я обрав безкоштовну конфігурацію на 512 мб(Рисунок 2.19), що цілком достатньо для старту додатку.

• Additional hardware configurations available for specialized workloads

Tier	RAM	Storage	vCPU	Price
M30	8 GB	40 GB	2 vCPUs	from **$0.58**/hr
M40 •	16 GB	80 GB	4 vCPUs	from **$1.12**/hr
M50 •	32 GB	160 GB	8 vCPUs	from **$2.16**/hr
M60 •	64 GB	320 GB	16 vCPUs	from **$4.27**/hr
M80 •	128 GB	750 GB	32 vCPUs	from **$7.90**/hr
M140	192 GB	1000 GB	48 vCPUs	from **$11.88**/hr
M200 •	256 GB	1500 GB	64 vCPUs	from **$15.78**/hr
M300 •	384 GB	2000 GB	96 vCPUs	from **$23.63**/hr
M400 •	512 GB	3000 GB	64 vCPUs	from **$24.24**/hr
M700	768 GB	4096 GB	96 vCPUs	from **$36.00**/hr

Рисунок 2.18 – Список доступних конфігурацій і їх вартість

Cluster Tier M10 (2 GB RAM, 10 GB Storage)
1 000 IOPS, Encrypted, Auto-expand Storage

Hourly price is for a MongoDB replica set with **3 data bearing servers.**

Dedicated Clusters for development environments and low-traffic applications

Tier	RAM	Storage	vCPU	Price
M10	2 GB	10 GB	2 vCPUs	from **$0.08**/hr
Storage	*10 GB is included in the base price.* 10 GB — 128 GB			10 GB
Auto-scale	☐ Cluster Tier Scaling View docs ☑ Storage Scaling			
IOPS	1000 IOPS			
Additional Info	1500 max connections | Up to 5 Gigabit network performance			
M20	4 GB	20 GB	2 vCPUs	from **$0.22**/hr

Рисунок 2.19 – Вибір конфігурації для моєї бд

При налаштуванню кластеру можна увімкнути функцію моніторінгу (рисунок 2.20). Ця функція дозволить переглядати вашу кількість підключень, що дуже зручно при першій спробі налаштування додатка із бд, оскільки ми завжди знаємо чи сервер підключився до бд і в який час. Слід зазначити що дана функція є безкоштовною для обраного плану конфігурацій.

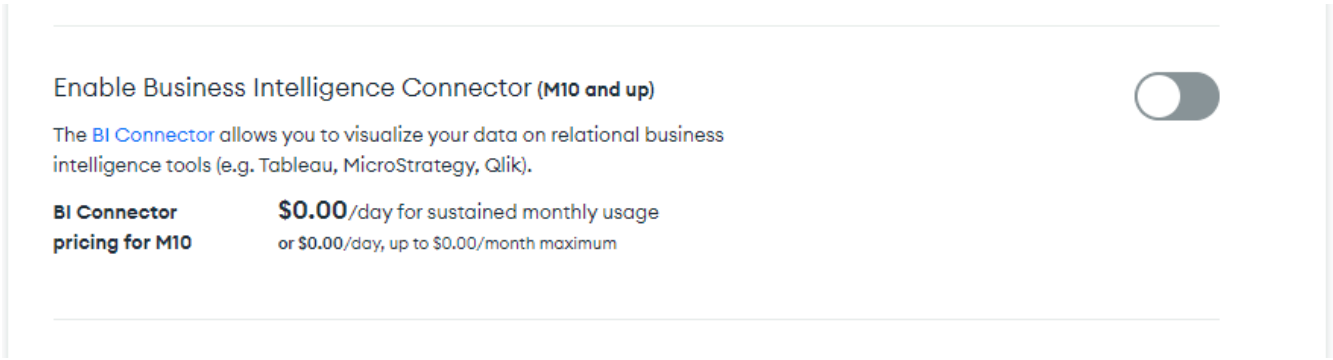


Рисунок 2.20 – Функція моніторінгу

Після налаштування кластеру ми переходимо на головний інтерфейс MongoDB Atlas (Рисунок 2.21). Наступна задача це створити базу даних де і будуть зберігатись завдання. Для створення бд необхідно зайти в колекції і натиснути кнопку create database, після чого відкриється меню створення бд(Рисунок 2.22). Необхідно ввести імя бд, для мене це TodoServe. Оскільки я вже прописав назву колекції даних в коді, окремо її створювати непотрібно. Якщо все вірно функціонує колекція даних створюється автоматично, навіть якщо вручну видалити її.

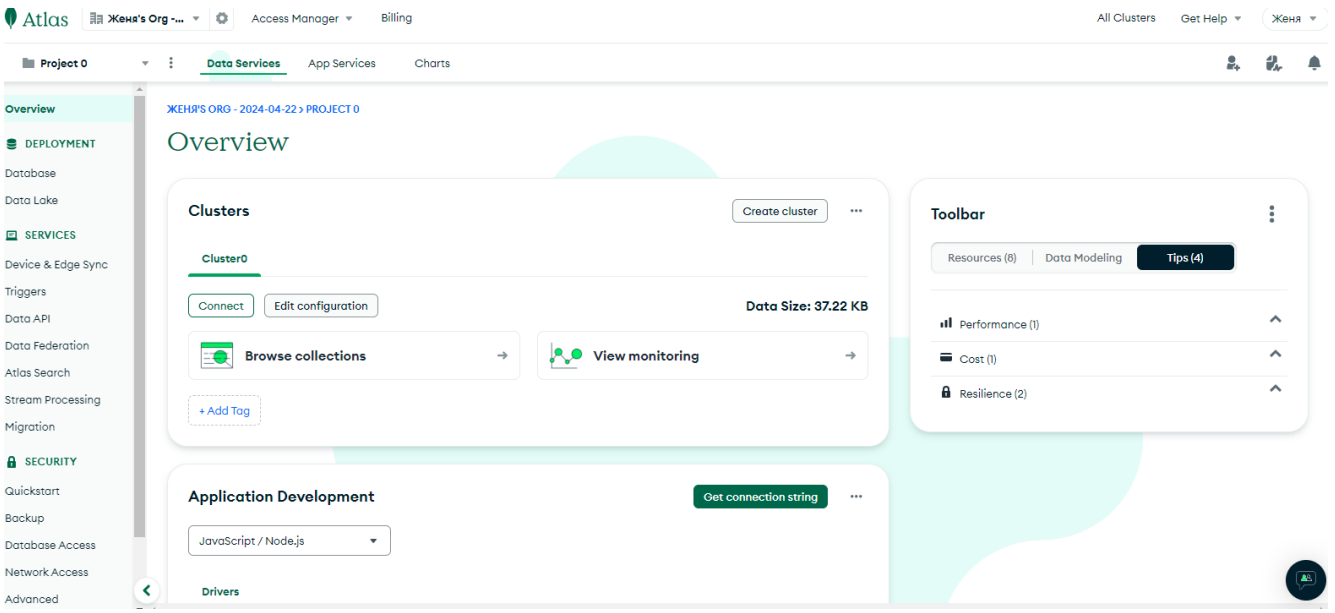


Рисунок 2.21 – Вигляд головного інтерфейсу MongoDB Atlas

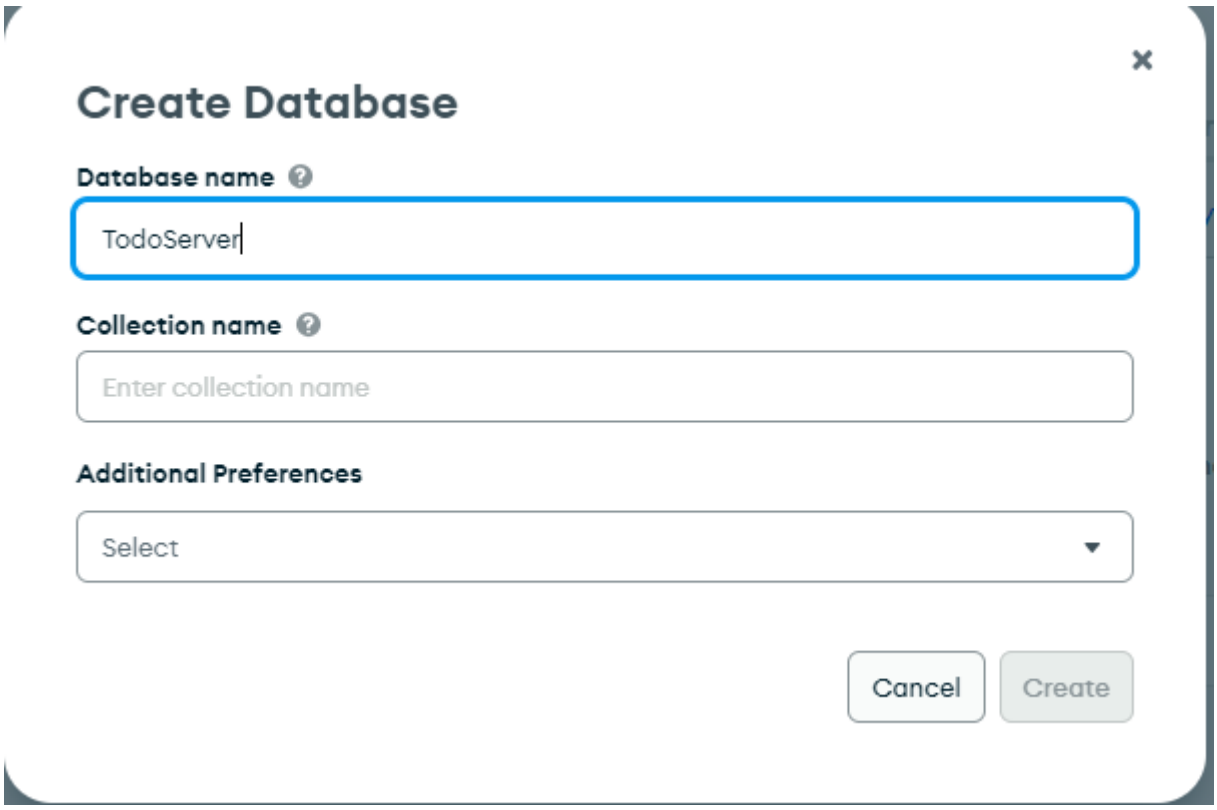


Рисунок 2.22 – Вікно для створення бд

Щоб успішно звязати сервер додатку і бд необхідно ввести ключ, саме він зєднує додаток і сервер. Також важливим кроком є налаштування доступу до бд. Для цього я створив користувача з адмінськими дозволами(Рисунок 2.23) і додав IP свого пристрою до дозволених підключень(Рисунок 2.24).

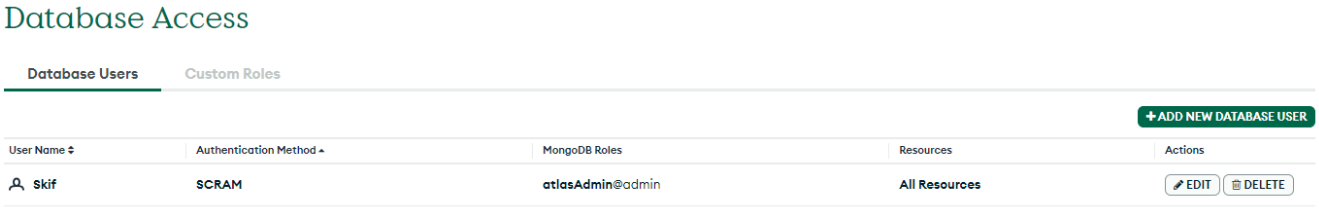


Рисунок 2.23 – Створений користувач із адмінськими правами

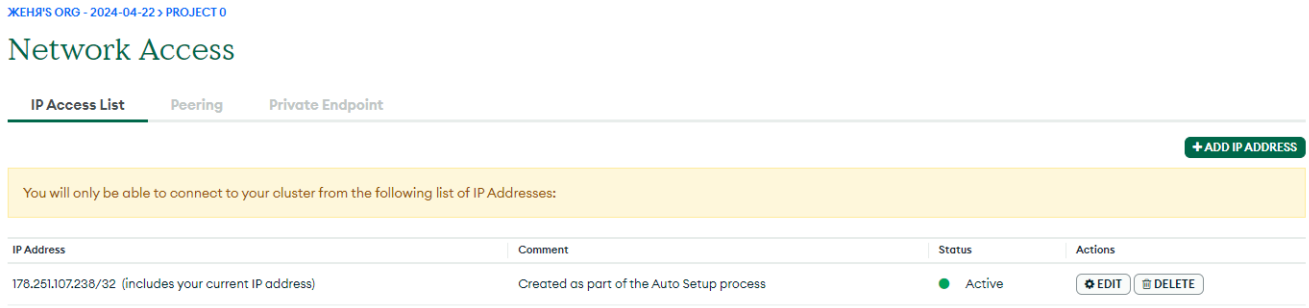


Рисунок 2.24 – IP адрес який до додав до списку дозволених для підключення

Після всіх кроків я спробував підключитись до бази даних. Після напису в консолі про успішне підключення від сервера, я перейшов до методів моніторінгу трафіку бд(Рисунок 2.25). Моніторінг показав 2 підключення, що являється нормою оскільки це є особливостями роботи самого MongoDB.

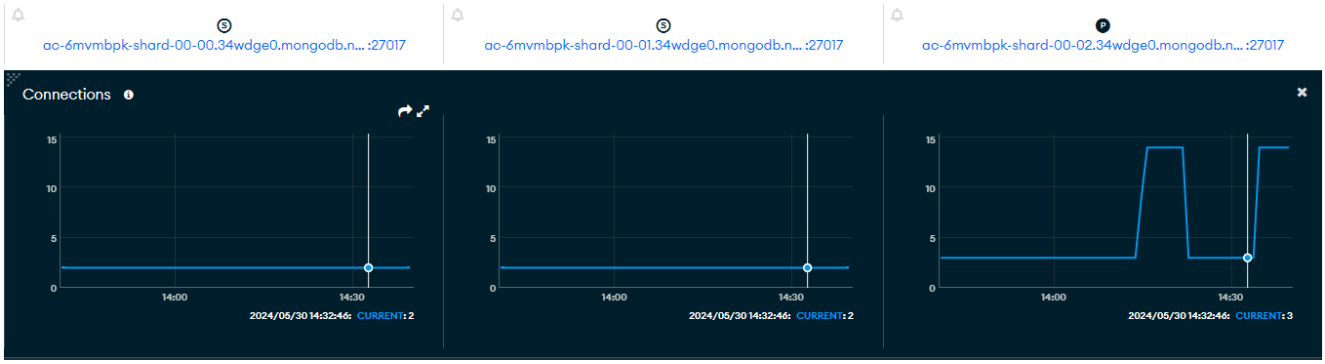


Рисунок 2.25 – Інтерфейс моніторінгу БД

3 ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1 Опис основних функцій та можливостей додатку

Після завершення розробки та налаштування всіх компонентів, додаток "TaskMaster" нарешті готовий до використання. На цьому етапі важливо протестувати його функціональні можливості та переконатися, що він виконує всі заплановані завдання. Давайте детально розглянемо основні функції та можливості додатку.

При вході в додаток, користувач опиняється на головній сторінці (Рисунок 3.1). Для використання додатку потрібно перейти у вікно TodoList, для цього потрібно нажати пкм по кнопці TodoList, після чого наш адрес із http://localhost:3001/ зміниться на http://localhost:3001/todolist. У вікні TodoList ми маємо всі функції, які нам потрібні для створення і перегляду всіх задач.

Рисунок 3.1 – Вигляд головної сторінки

3.1.1 Створення завдань

Для створення нової задачі користувачу необхідно ввести наступне:

Заголовок: поле для вводу заголовку робить всі введені літери жирними і великими (Рисунок 3.2).

Завдання: поле для вводу завдання займає найбільшу кількість місця серед всіх елементів, тому у користувача є достатньо місця для вводу тексту (Рисунок 3.3).

Дата: дату можна ввести ввручну або використати календарь (Рисунок 3.4).

Час: час дату можна ввести ввручну або використати меню для вибору часу (Рисунок 3.5).

Пріорітет: потрібно натиснути пкм і обрати пріорітет для задачі (Рисунок 3.6).

Після вводу всіх необхідних даних для задачі потрібно нажити кнопку Add task для внесення задачі у список справ.

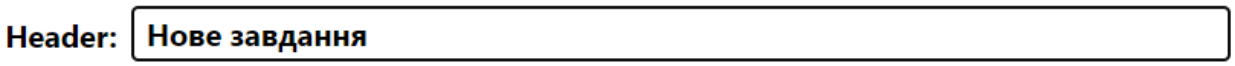


Рисунок 3.2 – Поле для вводу заголовку завдання

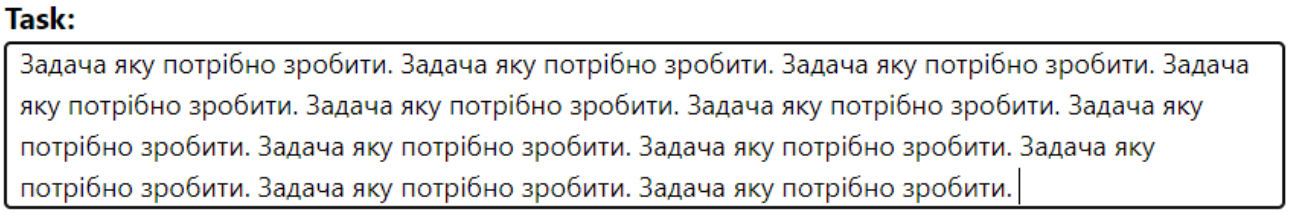


Рисунок 3.3 – Поле для вводу задачі

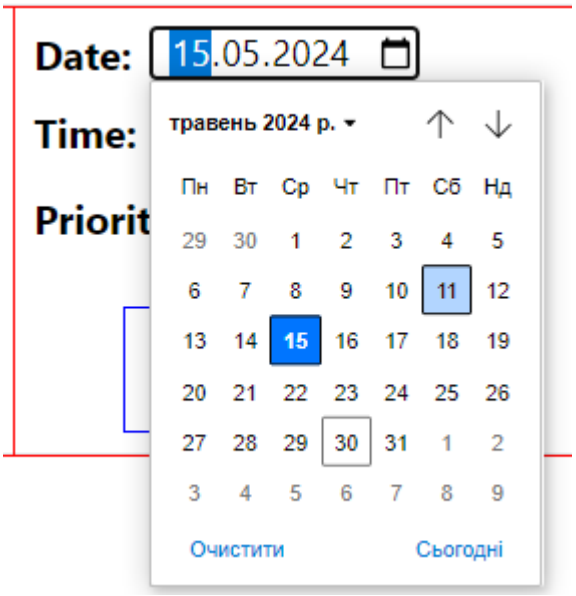


Рисунок 3.4 – Календарь для обрання дати

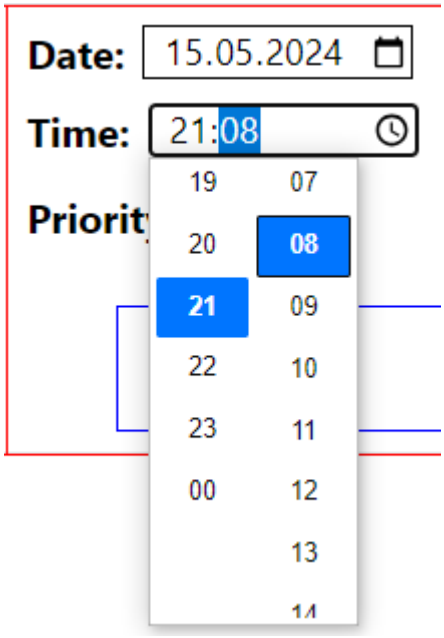


Рисунок 3.5 – Інтерфейс для вибору часу

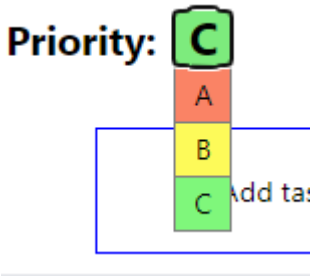


Рисунок 3.6 – Вигляд меню для виставлення пріорітету задачі

3.1.2 Перегляд завдань

Після створення задачі, вона додається до списку TodoList (Рисунок 3.7). Основна функція списку задач це перегляд всіх завдань які додав користувач. Також список задач можна сортувати за датою створення, алфавітним порядком і пріорітетом (Рисунок 3.8). А самі задачі мають наступні функції (Рисунок 3.9):

Статус завдання: при виконанні завдання можна видалити його або встановити галочку, що завдання виконане.

Видалення: завдання можна видалити із списку задач натиснувши кнопку Delete.

Рисунок 3.7 – Вигляд списку задач

Редагування: якщо необхідно, задачу можна редагувати натиснувши кнопку Edit, при цьому завдання видаляється із списку задач, а всі данні які були введені у завданні переходять у місце для вводу нових задач. При даній функції користувач може без проблем виправити або додати нові дані при цьому не переписуючі все з нуля.

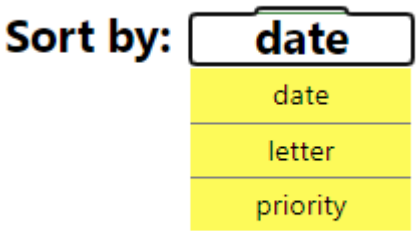


Рисунок 3.8 – Вибір методу сортування

Рисунок 3.9 – Вигляд задачі

ВИСНОВКИ

У ході розробки веб-додатку "TaskMaster" на початкових етапах було проведено аналіз ринку і предметної області, описані існуючі аналоги їх переваги і недоліки. Були проаналізовані сучасні технології для створення веб додатку.

Налаштоване середовище розробки та створення базової структури проекту. Реалізовано клієнтську та серверну частини додатку. Клієнтська частина включає в себе компоненти для введення, відображення та управління завданнями, тоді як серверна частина забезпечує збереження та обробку даних у базі даних. Завдяки використанню Redux, було досягнуто ефективного управління станом додатку, що дозволило забезпечити стабільну роботу навіть при великій кількості завдань.

Переваги розробленого додатку:

Висока продуктивність: Завдяки використанню віртуального DOM у React та оптимізованого рендерингу, додаток швидко реагує на дії користувачів.

Гнучкість та масштабованість: Компонентний підхід дозволяє легко додавати нові функції та змінювати існуючі без значних зусиль.

Використання сучасних технологій: Використання React, Node.js, MongoDB та інших сучасних інструментів забезпечує високу надійність та ефективність додатку.

Хмарне зберігання даних: Завдяки MongoDB Atlas дані зберігаються в хмарі, що забезпечує їх доступність та безпеку.

Управління станом: Використання Redux дозволяє централізовано управляти станом додатку, що спрощує його підтримку та розширення.

У майбутньому планується розширення функціональності додатку "TaskMaster" шляхом додавання нових можливостей та оптимізації існуючих. Деякі з можливих напрямків розвитку включають:

Інтеграція з іншими сервісами: Додавання інтеграцій для нагадування із gmail або telegram.

Мобільна версія: Розробка мобільної версії додатку для iOS та Android дозволить користувачам керувати завданнями з будь-якого пристрою.

Покращення інтерфейсу користувача: Постійне вдосконалення UI/UX дизайну додатку для забезпечення ще більшої зручності та інтуїтивності.

Розробка додатку "TaskMaster" значно покращить розподіл власного часу і допоможе організувати задачі.

ПЕРЕЛІК ПОСИЛАНЬ

1. React [Електронний ресурс] / React Documentation - Режим доступу: https://react.dev/

2. Node.js [Електронний ресурс] / Node.js Documentation - Режим доступу: https://nodejs.org/en

3. MongoDB Atlas [Електронний ресурс] / MongoDB - Режим доступу: https://www.mongodb.com/cloud/atlas/register

4. TailwindCSS [Електронний ресурс] / TailwindCSS Documentation - Режим доступу: https://tailwindcss.com/

5. React Router [Електронний ресурс] / React Router Documentation - Режим доступу: https://reactrouter.com/en/main

6. Express.js [Електронний ресурс] / Express Documentation - Режим доступу: https://expressjs.com/

7. Redux [Електронний ресурс] / Redux Documentation - Режим доступу: https://redux.js.org/

8. Redux Toolkit [Електронний ресурс] / Redux Toolkit Documentation - Режим доступу: https://redux-toolkit.js.org/

9. JavaScript: Understanding the Weird Parts [Електронний ресурс] / Udemy - Режим доступу: https://www.udemy.com/course/understand-javascript/

10. Tailwind CSS: From Zero to Production [Електронний ресурс] / Egghead.io - Режим доступу: https://egghead.io/courses/tailwind-css-from-zero-to-production-32ce582c

11. Building Applications with React and Redux [Електронний ресурс] / Pluralsight - Режим доступу: https://www.pluralsight.com/courses/react-redux-react-router-es6

12. Learning React: Functional Web Development with React and 18. Redux [Електронний ресурс] / O'Reilly Media - Режим доступу: https://www.oreilly.com/library/view/learning-react-2nd/9781492051718/

13. Full-Stack Web Development with React [Електронний ресурс] / Coursera - Режим доступу: https://www.coursera.org/learn/full-stack-react

14. React Hooks [Електронний ресурс] / React Documentation - Режим доступу: https://legacy.reactjs.org/docs/hooks-intro.html

15. Mastering Node.js [Електронний ресурс] / Packt Publishing - Режим доступу: https://www.packtpub.com/product/mastering-node-js/9781785888960

16. Advanced Node.js Development [Електронний ресурс] / Udemy - Режим доступу: https://www.udemy.com/course/advanced-nodejs-development/

17. Express in Action: Writing, Building, and Testing Node.js Applications [Електронний ресурс] / Manning Publications - Режим доступу: https://www.manning.com/books/express-in-action

18. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node [Електронний ресурс] / Apress - Режим доступу: https://www.apress.com/gp/book/9781484243906

19. HTTP Methods [Електронний ресурс] / Avior API Documentation - Режим доступу: https://www.contrive.mobi/aviorapi/HTTPMETHODS.html

20. MongoDB: The Definitive Guide [Електронний ресурс] / O'Reilly Media - Режим доступу: https://www.oreilly.com/library/view/mongodb-the-definitive/9781491954454/

ДОДАТОК А. ФРАГМЕНТИ ПРОГРАМНОГО КОДУ

```
import React, { useState } from "react";
import { useDispatch, useSelector } from "react-redux";
import { swichNumb, sortArray, addTodoItem } from "../../store/TodoStorySlice";
import { v4 as uuidv4 } from 'uuid';

const TodoInput = () => {
    const item = useSelector(state => state.TodoStorySlice.editItem);
    const swichNumber = useSelector(state => state.TodoStorySlice.swichNumber);
    const [isOpen, setIsOpen] = useState(false); // Стан для видимості меню
    const [isOpenSort, setIsOpenSort] = useState(false);
    const [selectedOptions, setSelectedOptions] = useState("C"); // Стан для обраних варіантів
    const [selectedSorts, setSelectedSorts] = useState("date");
    const dispatch = useDispatch();
    const [todoHeader, setTodoHeader] = useState("");
    const [todoTask, setTodoTask] = useState("");
    const [todoDate, setTodoDate] = useState("");
    const [todoTime, setTodoTime] = useState("");
    const [todoCreationTime, setTodoCreationTime] = useState("");
    // Функція для відкриття/закриття меню
    const togglePriorityMenu = () => {
        setIsOpen(!isOpen);
    };
    const toggleSortMenu = () => {
        setIsOpenSort(!isOpenSort);
    };
    // Функція для обрання варіанту відповіді
    const handleOptionClick = (option) => {
        setSelectedOptions(option);
        setIsOpen(false); // Закриття меню після обрання варіанту
    };
    const handleSortClick = (option) => {
        setSelectedSorts(option);
        setIsOpenSort(false);
        dispatch(sortArray({sort: option}))
    };
    const handlOnPressHeader = (event) => {
        setTodoHeader(event.target.value);
    };
    const handlOnPressTask = (event) => {
        setTodoTask(event.target.value);
    };
    const handlOnPressDate = (event) => {
        setTodoDate(event.target.value);
```

```
};
const handlOnPressTime = (event) => {
    setTodoTime(event.target.value);
};
if(swichNumber === 1){
    setTodoHeader(item.header);
    setTodoTask(item.task);
    setTodoDate(item.deadline.date);
    setTodoTime(item.deadline.time);
    setSelectedOptions(item.priority);
    setTodoCreationTime(item.creationTime);
    dispatch(swichNumb());
};
const handlOnclickButtunAdd = () => {
    const todoItem = {
        header: todoHeader,
        task: todoTask,
        deadline: {
            date: todoDate,
            time: todoTime
        },
        priority: selectedOptions,
        id: uuidv4(),
        checkbox: false,
        creationTime: todoCreationTime === "" ? new Date().toLocaleTimeString() : todoCreationTime,
    };
    dispatch(addTodoItem(todoItem));
    setTodoHeader("");
    setTodoTask("");
    setTodoDate("");
    setTodoTime("");
    setSelectedOptions("C");
    setTodoCreationTime("");

};
return (
    <div>
        <div className='w-[100%] h-[400px] flex justify-center items-center flex-row pt-[50px] gap-x-[50px] '>
            <div className='flex pt-[50px] justify-center'>
                <div className='w-[1300px] h-[250px] border-[1px]'>
                    <div className='border-[1px] border-[red] h-[250px] flex justify-start rounded'>
                        <div className='border-r-[1px] border-[red] w-[1000px] p-[20px]'>
                            <div className="flex">
                                <span className='mr-[10px] font-bold text-[22px] mt-[5px]'>Header: </span>
                                <input value={todoHeader} onChange={handlOnPressHeader} name="HeaderInput" className='border-[2px]
rounded border-[blue] mr-[40px] w-[800px] h-[40px] pl-[10px] text-[22px] font-bold' />
                            </div>
```

```
                    <div className="flex mt-[10px] flex-col">
                      <span className='mr-[10px] font-bold text-[22px]'>Task: </span>
                      <textarea value={todoTask} onChange={handlOnPressTask} className="text-[20px] pl-[10px] pr-[10px] border-[1px] rounded border-[blue] resize-none" name="task" rows={4} cols={40} />
                    </div>
                  </div>
                  <div className="h-[248px] ">
                    <div className='flex flex-col w-[320px] h-[158px] justify-start items-start p-[10px]'>
                      <div className="flex">
                        <span className='mr-[10px] mb-[10px] font-bold text-[22px]'>Date: </span>
                        <input value={todoDate} onChange={handlOnPressDate} name="DateInput" type="date" className='border-[1px] border-[black] mr-[20px] w-[150px] h-[30px] pl-[10px] text-[20px]' />
                      </div>
                      <div className="flex">
                        <span className='mr-[10px] mb-[10px] font-bold text-[22px]'>Time: </span>
                        <input value={todoTime} onChange={handlOnPressTime} name="TimeInput" type="time" className='border-[1px] border-[black] mr-[20px] w-[150px] h-[30px] pl-[10px] text-[20px]' />
                      </div>
                      <div className="mt-[5px] flex">
                        <div className="flex">
                          <span className='mr-[10px] font-bold text-[22px]'>Priority: </span>
                        </div>
                        <div className="z-10">
                          <button onClick={togglePriorityMenu} className="w-[32px] h-[30px] flex justify-center items-center bg-[#7deb7d] font-bold text-[25px] hover:bg-[#418641]">{selectedOptions}</button>
                            {isOpen && (
                              <ul className="border-[1px] border-[gray] w-[32px]">
                                <li onClick={() => handleOptionClick('A')} className=" w-[30px] h-[30px] flex justify-center items-center bg-[#fa8365] cursor-pointer hover:bg-[#ad5843]">A</li>
                                <li onClick={() => handleOptionClick('B')} className=" w-[30px] h-[30px] border-t-[1px] border-[gray] flex justify-center items-center bg-[#fdfa59] cursor-pointer hover:bg-[#a8a63c]">B</li>
                                <li onClick={() => handleOptionClick('C')} className=" w-[30px] h-[30px] border-t-[1px] border-[gray] flex justify-center items-center bg-[#7ff77b] cursor-pointer hover:bg-[#419b3e]">C</li>
                              </ul>
                            )}
                          </div>
                        </div>
                      </div>
                      <div className='h-[90px] w-[320px] flex justify-center items-start'>
                        <button onClick={handlOnclickButtunAdd} className='border-[1px] border-[blue] w-[200px] h-[70px] mt-[8px] hover:bg-[#d8d8d8]'>Add task</button>
                      </div>
                    </div>
                  </div>
                </div>
                <div className="h-[200px] flex-row mt-[20px] flex justify-center items-start">
```

```
            <span className="text-[25px] font-bold mr-[10px]">Sort by: </span>
            <div className="mt-[5px]">
              <button onClick={toggleSortMenu} className="w-[125px] h-[30px] flex justify-center items-center bg-[#ffffff] font-bold text-[25px] border-[1px] border-[#418641] focus:border-b-[white]">{selectedSorts}</button>
                {isOpenSort && (
                  <ul className="border-[1px] border-[#ffffff] h-[92px] w-[122px]">
                    <li onClick={() => handleSortClick('date')} className=" w-[122px]  h-[30px] flex justify-center items-center bg-[#fdfa59] cursor-pointer hover:bg-[#a8a63c]">date</li>
                    <li onClick={() => handleSortClick('letter')} className=" w-[122px] h-[30px] border-t-[1px] border-[gray] flex justify-center items-center bg-[#fdfa59] cursor-pointer hover:bg-[#a8a63c]">letter</li>
                    <li onClick={() => handleSortClick('priority')} className=" w-[122px] h-[30px] border-t-[1px] border-[gray] flex justify-center items-center bg-[#fdfa59] cursor-pointer hover:bg-[#a8a63c]">priority</li>
                  </ul>
                )}
            </div>
          </div>
        </div>
    )
}

export default TodoInput;

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import axios from 'axios';

// Створення асинхронної функції для отримання todoItems з сервера
export const fetchTodoItems = createAsyncThunk(
  'todoStory/fetchTodoItems',
  async () => {
    const response = await axios.get('http://localhost:3000/api/TodoArray');
    console.log(response.data);
    return response.data;
  }
);

// Створення асинхронної функції для додавання todoItem на сервер
export const addTodoItem = createAsyncThunk(
  'todoStory/addTodoItem',
  async (newTodoItem, { getState }) => {
    const state = getState();
    const todoArray = [...state.TodoStorySlice.todoArray, newTodoItem];
    await axios.post('http://localhost:3000/api/TodoArray', todoArray); // Оновіть URL на '/api/TodoArray'
    return todoArray;
  }
);
```

```
export const updatechecBoxAndDelete = createAsyncThunk(
  'todoStory/addTodoItem',
  async (_, { getState }) => {
    console.log("Work");
    const state = getState();
    const todoArray = [...state.TodoStorySlice.todoArray];
    await axios.post('http://localhost:3000/api/TodoArray', todoArray); // Оновіть URL на '/api/TodoArray'
    return todoArray;
  }
);

const TodoStorySlice = createSlice({
    name: 'TodoStorySlice',
    initialState: {
      text: 'Task list',
      todoArray: [],
      editItem: {
        header: '',
        task: '',
        deadline: {
          date: '',
          time: '',
        },
        priority: 'C',
        creationTime: '',
      },
      swichNumber: 0,
      sortStatus: 'date',
      creationTime: '',
      status: 'idle', // Статус для відстеження стану взаємодії з сервером
    },
    reducers: {

      changeChekbox(state, action){
        let id = state.todoArray.findIndex((item) => action.payload.id === item.id);
        state.todoArray[id].checkbox = !state.todoArray[id].checkbox;
      },
      deleteItem(state, action){
        state.todoArray = state.todoArray.filter((item) => action.payload.id !== item.id);
      },
      editItem(state, action){
        let id = state.todoArray.findIndex((item) => action.payload.id === item.id);
        state.editItem = {
          header: state.todoArray[id].header,
          task: state.todoArray[id].task,
          deadline: {
            date:  state.todoArray[id].deadline.date,
```

```
          time:  state.todoArray[id].deadline.time
        },
        priority: state.todoArray[id].priority,
        creationTime: state.todoArray[id].creationTime,

      }
      state.todoArray = state.todoArray.filter((item) => action.payload.id !== item.id);
      state.swichNumber++;
    },
    swichNumb(state){
      state.editItem = {
        header: "",
        task: "",
        deadline: {
          date: "",
          time: ""
        },
        priority: "C",
        creationTime: "",

      };
      state.swichNumber--;
    },
    sortArray(state, action){
      state.sortStatus = action.payload;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(fetchTodoItems.pending, (state, action) => {
        state.status = 'loading';
      })
      .addCase(fetchTodoItems.fulfilled, (state, action) => {
        state.status = 'succeeded';
        state.todoArray = action.payload;
      })
      .addCase(fetchTodoItems.rejected, (state, action) => {
        state.status = 'failed';
      })
      .addCase(addTodoItem.fulfilled, (state, action) => {
        state.status = 'succeeded';
        // Оновлюємо стан залежно від того, як ви оновлюєте масив на клієнті
        state.todoArray = action.payload;
      });
  },
});
export const {addItem, changeChekbox, deleteItem, editItem, swichNumb, sortArray} = TodoStorySlice.actions;
```

export default TodoStorySlice.reducer;

ДОДАТОК Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)

Розробка веб-додатку 'TaskMaster' за допомогою React, Node.js та MongoDB

Автор: Євгеній Двуреченський
Керівник: Олена Шикула, д.ф.-м.н., професор
Київ 2024

Актуальність роботи

У сучасному світі веб-додатки стають дуже популярними завдяки своїй доступності з будь-якого пристрою, що має підключення до Інтернету. Все більше користувачів обирають веб-додатки для управління своїми завданнями та плануванням.

Існує велика потреба в ефективних інструментах управління завданнями. Користувачі шукають зручні та функціональні рішення, але існуючі додатки часто не відповідають їхнім вимогам.

Особливо важливо створити інструмент, який задовольнятиме потреби різних користувачів: студентів, професіоналів, фрілансерів та домогосподарок. "TaskMaster" пропонує можливість адаптації додатку до індивідуальних потреб кожного.

Розробка "TaskMaster" базується на сучасних технологіях: React, Node.js та MongoDB. Це забезпечує високу продуктивність і надійність додатку. Інтеграція з MongoDB Atlas надає можливість масштабування та забезпечує безпеку даних.

Таким чином, розробка "TaskMaster" є *актуальною* та важливою, оскільки вона відповідає сучасним потребам користувачів та технологічним тенденціям, забезпечуючи ефективне управління завданнями і вдосконалення тайм-менеджменту.

Вступ

- Мета роботи: Розробка функціонального та надійного веб-додатку для управління завданнями, який би відповідав сучасним вимогам користувачів та ринку.
- Об’єкт дослідження: Процес управління завданнями за допомогою веб-додатків, що використовують сучасні технології фронтенд та бекенд розробки.
- Предмет дослідження: Програмні засоби та технології для реалізації ефективної системи управління завданнями.

1 Аналіз проблемної області

При розгляді існуючих рішень для управління завданнями, можна виділити декілька ключових проблем, з якими стикаються користувачі:

1.Зручність використання:

1. Багато додатків мають складний інтерфейс, що може ускладнювати роботу для нових користувачів.
2. Недостатньо інтуїтивно зрозумілі елементи управління, що знижує ефективність використання додатку.

2.Обмежені функції у безкоштовних версіях:

1. Багато додатків обмежують доступ до основних функцій у безкоштовних версіях, змушуючи користувачів купувати преміум-версії для повноцінного використання.
2. Це може стати бар'єром для тих, хто шукає функціональний інструмент без додаткових витрат.

1 Аналіз проблемної області

3.Відсутність функцій сортування і пріоритезації:

1. Не всі додатки пропонують зручні інструменти для сортування завдань за пріоритетами, що ускладнює управління великими списками завдань.
2. Відсутність можливості швидко і легко встановлювати пріоритети може призводити до неефективного планування і виконання завдань.

Ці проблеми вказують на необхідність розробки нового інструменту, який би був зручним у використанні, пропонував всі необхідні функції без додаткових витрат і мав потужні інструменти для сортування та пріоритезації завдань. "TaskMaster" спрямований на вирішення цих проблем, забезпечуючи користувачів зручним та ефективним інструментом для управління завданнями.

1 Огляд ринку та аналогічних рішень

Існує багато інструментів для управління завданнями, але кожен з них має свої переваги та недоліки. Розглянемо деякі з найбільш популярних:

1.Todoist:

1. Популярний додаток для управління завданнями.
2. Пропонує широкий функціонал, включаючи створення і редагування завдань, встановлення термінів виконання, пріоритезацію завдань, додавання описів та вкладень, а також підтримку колективної роботи.

2.Microsoft To Do:

1. Інтегрований з іншими продуктами Microsoft.
2. Забезпечує зручність для користувачів екосистеми Microsoft, пропонуючи функції створення і редагування завдань, встановлення термінів, пріоритезацію та підтримку колективної роботи.

1 Огляд ринку та аналогічних рішень

Функції	Todoist	Microsoft To Do	Google Keep
Створення і редагування завдань	Так	Так	Так
Встановлення термінів виконання	Так	Так	Так
Пріоритезація завдань	Так	Так	Ні
Додавання описів та вкладень	Так	Так	Так
Підтримка колективної роботи	Так	Так	Ні

1 Цільова аудиторія

Основні групи цільової аудиторії:

1.Індивідуальні користувачі

1. Студенти: Організація навчання, планування завдань і проектів.
2. Фрілансери: Управління замовленнями, контроль термінів, організація робочого процесу.

2.Професіонали та фрілансери

1. Професіонали: Управління робочими завданнями, проектами, зустрічами.
2. Фрілансери: Управління проектами, контроль термінів, відстеження оплат.

3.Освітні установи та студенти

1. Учні: Планування навчання, організація проектів, відстеження термінів.
2. Викладачі: Організація навчальних планів, розкладів, відстеження завдань.

4.Особиста продуктивність

1. Особисті цілі: Планування і досягнення цілей, тренування, читання.
2. Експериментатори: Тестування методів тайм-менеджменту, пошук ефективних способів організації часу.

2 Огляд та вибір технологій для розробки

Для розробки веб-додатку "TaskMaster" були обрані сучасні технології, які забезпечують високу продуктивність, надійність та масштабованість. Розглянемо основні технології, використані у проекті:

1.React:

1. Чому обрано: React є однією з найпопулярніших бібліотек для створення користувацьких інтерфейсів. Вона дозволяє розробляти динамічні та інтерактивні веб-додатки за допомогою компонентного підходу.
2. Переваги: Висока продуктивність, легкість у створенні компонентів, велика спільнота розробників та широкий вибір готових рішень.

2 Огляд та вибір технологій для розробки

2Node.js:

1. Чому обрано: Node.js є потужним середовищем виконання JavaScript на сервері. Воно дозволяє створювати високопродуктивні серверні додатки з використанням одного і того ж мовного середовища як на клієнтській, так і на серверній стороні.
2. Переваги: Висока швидкість обробки запитів, неблокуюча архітектура, активна підтримка спільноти та велика кількість доступних модулів.

2 Огляд та вибір технологій для розробки

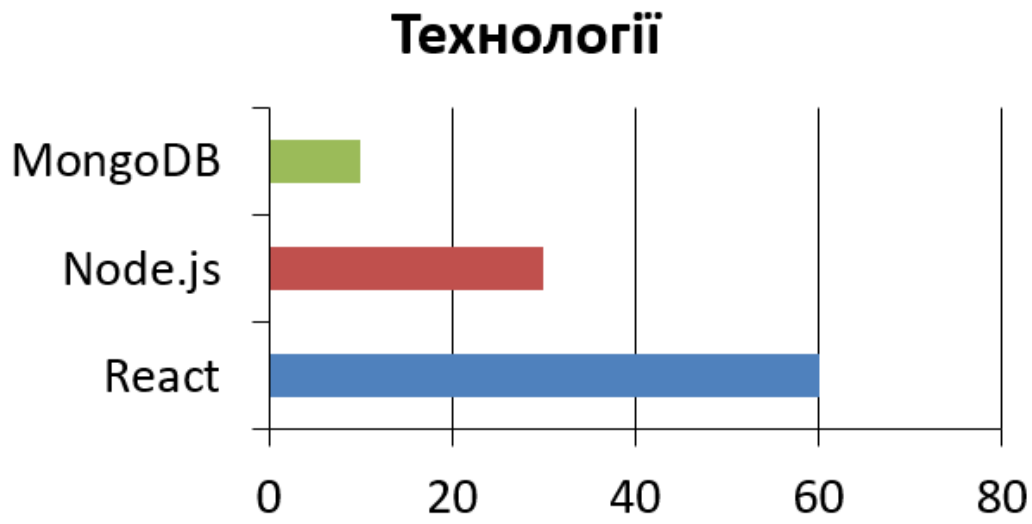
3.MongoDB:

1. Чому обрано: MongoDB є документно-орієнтованою базою даних NoSQL, яка забезпечує гнучкість та масштабованість. Вона ідеально підходить для зберігання структурованих та неструктурованих даних.
2. Переваги: Простота у використанні, гнучка модель даних, висока продуктивність при великих навантаженнях, інтеграція з хмарними сервісами, такими як MongoDB Atlas.

Ці технології були обрані через їх відповідність вимогам проекту "TaskMaster". Вони забезпечують розробку надійного та масштабованого додатку, що може ефективно вирішувати завдання користувачів та відповідати сучасним тенденціям в області веб-розробки.

2 Використання технологій

На діаграмі представлені три ключові технології, що використовувалися під час розробки веб-додатку "TaskMaster« та вказане відсоткове співвідношення їх в проекті:



2 Інші технології

1.TailwindCSS:
    1. Опис: Утилітарний CSS-фреймворк.
    2. Переваги: Швидкість розробки, гнучкість у налаштуванні, мінімізація CSS-коду.
2.React Router DOM:
    1. Опис: Бібліотека для маршрутизації в React.
    2. Переваги: Простота у налаштуванні маршрутів, підтримка динамічних маршрутів, управління навігацією.

2 Інші технології

3.Express:
    1. Опис: Мінімалістичний веб-фреймворк для Node.js.
    2. Переваги: Легкість налаштування серверних маршрутів, підтримка middleware, гнучкість у створенні серверних рішень.
4.Redux та Redux Toolkit:
    1. Опис: Бібліотека для управління станом додатків.
    2. Переваги: Централізоване управління станом, спрощення розробки, полегшення тестування.

2 Процес розробки

Розробка веб-додатку "TaskMaster" включала кілька основних етапів, кожен з яких забезпечував важливі аспекти проекту:

1.Створення та налаштування проекту:

1. Підготовка середовища розробки: Встановлення необхідного програмного забезпечення та інструментів, налаштування середовища для розробки.
2. Ініціалізація проекту: Створення основної структури проекту, налаштування вихідного коду та конфігураційних файлів.
3. Підключення необхідних технологій: Встановлення та налаштування основних бібліотек та фреймворків, таких як React, Node.js, MongoDB, TailwindCSS, Express, та Redux.

2 Процес розробки

2.Розробка клієнтської частини:

1. Створення інтерфейсу користувача: Розробка компонентів інтерфейсу, таких як Header, TodoInput, TodoList, та TodoPage, використовуючи React.
2. Маршрутизація: Налаштування навігації між різними сторінками додатку за допомогою React Router DOM.
3. Стайлінг: Використання TailwindCSS для створення стильного та сучасного інтерфейсу.

2 Процес розробки

3.Розробка серверної частини:

1. Створення серверу: Використання Node.js та Express для налаштування серверу, обробки API запитів та виконання бізнес-логіки.
2. Інтеграція з базою даних: Підключення до MongoDB для зберігання та обробки даних про завдання.
3. Безпека та автентифікація: Налаштування механізмів безпеки, таких як автентифікація користувачів та захист даних.

3 Досягнуті результати

1.Функціональний веб-додаток для управління завданнями:

1. Користувачі можуть створювати, редагувати, видаляти та пріоритезувати завдання.
2. Інтуїтивно зрозумілий та зручний інтерфейс.
3. Підтримка колективної роботи для командних проектів.

2.Висока продуктивність та зручний інтерфейс:

1. Використання React, Node.js та MongoDB для швидкої та надійної роботи.
2. Адаптивний дизайн для коректної роботи на комп'ютерах, планшетах і смартфонах.

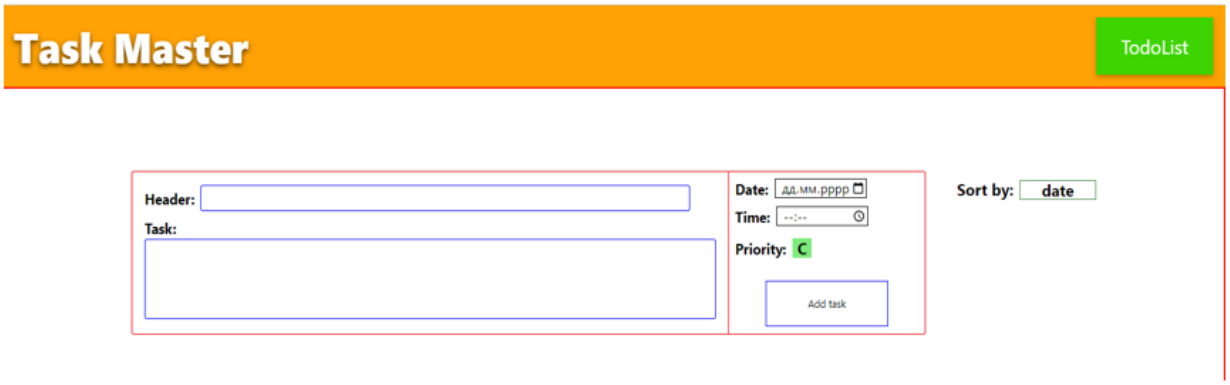
3.Централізоване управління станом додатку за допомогою Redux:

1. Централізоване управління станом додатку для передбачуваності та зручності.
2. Спрощення налагодження та тестування завдяки централізованому управлінню станом.

4.Інтеграція з хмарними сервісами:

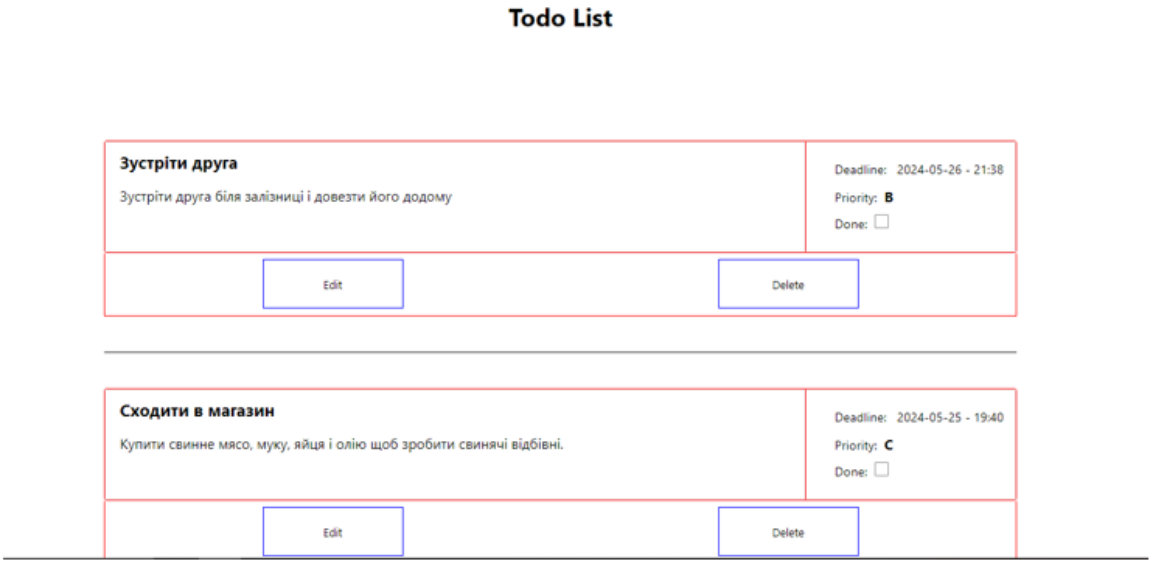
1. Використання MongoDB Atlas для надійного зберігання та масштабованості даних.
2. Високий рівень безпеки даних завдяки хмарним сервісам.

3 Вигляд додатку



Вигляд форми для створення завдання

3 Вигляд додатку



Вигляд списку задач

Висновки та подальший розвиток

1.Висновки:

1. Успішна реалізація: Розроблений функціональний і зручний веб-додаток для управління завданнями.
2. Задоволення потреб користувачів: Додаток відповідає потребам студентів, професіоналів, фрілансерів та інших груп користувачів.
3. Висока продуктивність: Використання сучасних технологій забезпечує швидку та надійну роботу додатку.

Висновки та подальший розвиток

1.Подальший розвиток:

1. Додавання нових функцій: Розширення можливостей додатку, включаючи нові інструменти для управління завданнями.
2. Покращення інтерфейсу: Подальше удосконалення користувацького інтерфейсу для забезпечення ще більшої зручності.
3. Розширення інтеграцій: Інтеграція з іншими популярними сервісами та інструментами для покращення взаємодії користувачів.
4. Оптимізація продуктивності: Підвищення швидкості роботи додатку та зниження навантаження на серверні ресурси.