Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
Факультет інформаційних технологій
Кафедра інформаційних технологій та комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня *бакалавра*

Студентки *Петриги Марини Вадимівни*

академічної групи *126 – 18 – 1*

спеціальності *126 «Інформаційні системи та технології»*

на тему: *Розробка веб-орієнтованої інформаційної системи тайм-менеджменту з використанням стеку технологій MERN*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		Рейтинговою	**Інституційною**	
кваліфікаційної роботи	*к.т.н., доц. Соколова Н.О.*			
розділів:	2			
Рецензент	*к.т.н., доц., доц. Реута О.В.*			
Нормоконтролер	*д.т.н., проф. Коротенко Г.М.*			

Дніпро
2022

ЗАТВЕРДЖЕНО:

завідувач кафедри

Інформаційних технологій та комп'ютерної інженерії

(повна назва)

_____ *д.т.н., проф. Гнатушенко В.В.*

(підпис) (прізвище, ініціали)

«_____» _____2022року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня *бакалавра*

студентці *Петризі М.В.* **академічної групи** *126-18-1*

спеціальності: *126 «Інформаційні системи та технології»*

на тему «*Розробка веб-орієнтованої інформаційної системи тайм-менеджменту з використанням стеку технологій MERN*»

затверджену наказом ректора НТУ «Дніпровська політехніка» від *18.05.2022 р. №268-с*

Розділ	Зміст	Терміни виконання
1. Розділ 1	1. *Дослідити потреби самоорганізації* 2. *Дослідити методи тайм менеджменту* 3. *Дослідити та проаналізувати існуючі рішень;*	02.05.2022 – 07.05.2022 08.05.2022 – 12.05.2022 13.05.2022 – 16.05.2022
2. Розділ 2	1. *Розробка прототипу, структури та моделі інформаційної системи* 2. *Розробка інформаційної системи*	17.05.2022 – 23.05.2022 24.05.2022 – 05.06.2022

Завдання видано _____ *доц. Соколова Н.О.*

(підпис) (прізвище, ініціали)

Дата видачі: *02.05.2022 р.*

Дата подання до екзаменаційної комісії:_____

Прийнято до виконання _____ _____

(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 58 сторінок, 39 рисунків, 1 додаток, 18 джерел.

Тема дипломної роботи: «Розробка веб-орієнтованої інформаційної системи тайм-менеджменту з використанням стеку технологій MERN»

Предмет дослідження: методи керування часом та якісні інструменти для керування.

Об'єкт дослідження: тайм менеджмент.

Об'єкт розроблення: веб-інформаційна система тайм-менеджменту.

Мета дипломної роботи: створення зручної інформаційної системи з методами тайм менеджменту та успішного запровадження цього поняття у різні сфери життя шляхом аналізу існуючих рішень та потреб.

Розроблене технічне рішення може бути впроваджено як система особистого тайм менеджменту чи як система тайм менеджменту працівника.

Ключові слова: ТАЙМ МЕНЕДЖМЕНТ, ЕФЕКТИВНІСТЬ, ІНФОРМАЦІЙНА СИСТЕМА, MERN, ІНТЕРФЕЙС, СЕРВЕР, БАЗА ДАНИХ

ABSTRACT

Explanatory note: 58 pages, 39 figures, 1 appendix, 18 sources.

Thesis topic: "WEB-based time management information system development using the MERN technology stack".

Subject of research: time management methods and quality management tools.

Object of research: time management.

Object of development: web time management information system.

The purpose of the thesis: to create a convenient information system with methods of time management and successful implementation of this concept in various spheres of life by analyzing existing solutions and needs.

The developed technical solution can be implemented as a personal time management system or as an employee time management system.

Keywords: TIME MANAGEMENT, EFFICIENCY, INFORMATION SYSTEM, MERN, INTERFACE, SERVER, DATABASE

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1. АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧІ 8
1.1 Опис предметної області 8
1.2 Аналіз сучасних тенденцій 16
1.3 Аналіз існуючих рішень 17
1.3.1 Habitica 17
1.3.2 Tweek 19
1.3.3 MyLifeOrganized 21
1.3.4 Clear 23
1.4 Постановка задачі 24
1.5 Висновки по першому розділу 25
РОЗДІЛ 2. ПРОЕКТНІ РІШЕННЯ 26
2.1 Огляд методів розробки 26
2.1.1 Огляд платформи для додатка 26
2.1.2 Огляд технологій для моделювання системи 28
2.1.3 Аналіз технологій розробки клієнтської сторони 29
2.1.4 Аналіз технологій розробки серверної сторони 32
2.2 Проектування і моделювання системи 32
2.2.1 Розробка моделей архітектури 32
2.2.2 Проектування додатку 36
2.2.3 Розробка прототипу екранів сайту 41
2.3 Програмна реалізація 44
2.3.1 Створення серверної частини 44
2.3.2 Реалізація бази даних 49
2.3.3 Реалізація клієнтської сторони 51
2.3.4 Структура створеного проекту 53
2.4 Результат розробки 55

2.5 Висновки по другого розділу..... 59
ВИСНОВКИ 60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 61
ДОДАТОК А 63

ВСТУП

Проблема швидкоплинності часу та його ефективного використання має глибоке коріння в історії людства. Багато поколінь задається питання що таке час, як його можна контролювати, а якщо його неможливо контролювати, як обходиться з величезним впливом на людське життя. Час – це найцінніший ресурс, з яким треба вчитися поводитися.

Представляючи людину як цільну систему, в якій один з механізмів цієї системи несправне, то несправна ціла система. Як здоровий дух, не може бути в нездоровому тілі. Так не роздільні поняття робочого тайм менеджменту та особистого. В цей саме момент, людина зіткається з думкою, що час контролювати дійсно неможливо, але їм можна успішно керувати і використовувати у своїх цілях. Дану систему контролю час так і назвали – тайм-менеджмент.

Головними рушіями системи тайм менеджменту є методи та техніки з управління. Техніки дозволяють різними шляхами компонувати план на різні періоди та оцінювати пріоритет. Методи в свою чергу вирішують питання виконання цього плану.

Як вже було зазначено вище, коли всі механізми працюють налагоджено та добре, система працює. Тому не менш важливим є інструмент запровадження. Дуже добре себе запропоновали інформаційні системи. Можливість динамічного планування, масштабності, система нагадувань, система нагороджень – все це відображає один із потужніших рішень для запровадження тайм менеджменту.

Предметом даної роботи є методи керування часом та якісні інструменти для керування.

Об’єктом даної роботи є тайм менеджмент.

Метою даної роботи є створення зручної інформаційної системи з методами тайм менеджменту та успішного запровадження цього поняття у різні сфери життя шляхом аналізу існуючих рішень та потреб.

Для досягнення мети роботи поставлені такі завдання:

а) огляд потреб самоорганізації у особистому розвитку та професійні діяльності;
б) дослідження методів тайм менеджменту;
в) огляд існуючих рішень;
г) розробка прототипу, структури та моделі інформаційної системи;
д) розробка інформаційної системи.

РОЗДІЛ 1. АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧІ

1.1 Опис предметної області

Уявити собі життя без поняття часу неможливо. Це єдине, чию дію не можна зупинити, але можливо під нього ефективно налаштуватися.

Ще за часів Стародавнього Риму, відомий мислитель Сенека, розмірковував у першому листі з «Моральні листи до Луцілія» як не витрачати час: «Час — це єдиний і найцінніший ресурс, яким ви володієте. Будьте суворі з часом, знайте, на що витрачаєте його, і цінуйте, якщо інші приділяють вам свій час.». Також, Сенека вів постійний облік часу в письмовому виді, так само як вів облік грошей [1].

Треба розуміти, що на життя людини та її благополуччя впливає не тільки добре організована професійна сфера, а й вміння самоорганізації. Особливо це помітно в час саморозвитку і великої кількості інструментів для нього. З можливостями, які не хочеться втрачати, приходить потреба реалізації свого бажання встигнути все і більше простору та відповідальності для вміння врегулювання своїх планів.

Наразі пандемія надала ще більше вільного часу. За даними навчальної платформи Prometheus, в період пандемії кількість активних слухачів онлайн курсів збільшилося в 4 рази, але середньостатистичний показник показує, що лише 13% людей закінчують обрані курси [2, 3]. На це впливає багато питань, які може вирішити ефективне управління своїми планами на будь-який строк та вчасний аналіз вже пройденого шляху. Будуть доречними слова бізнесмена та інвестора Роберта Кійосакі, що у кожної людини не більше часу, аніж у вас, але деякі встигають більше. Головний ключ – правильний

розподіл часу, оскільки це інвестиція в успішний розвиток робочих проектів та особистий, що є неподільними речами [4].

Міркування та розуміння поняття часу має невід'ємну частину реалізації з його управління, тобто – тайм менеджменту.

Тайм-менеджмент – сукупність інструментів, методів, процесів раціонального, свідомого контролю за кількістю часу, витраченого на певну діяльність, що націлена на зростання ефективності та продуктивності. Термін розширив своє значення не тільки як контроль робочої діяльність, але і особистої. Це вийшло з часом, так як методи тайм менеджменту підійшли до різноманітних рішень не пов'язаних з шляхом реалізацій ділових цілей, а балансування з цілями пов'язаними з родиною, захоплюваннями, саморозвитком [5].

Управління часом дозволяє вирішувати різноманітні задачі людської ефективності та працездатності:

е) Керування собою. Тайм менеджмент навчає приймати зовнішні обставини та корегувати їх вписуючи в своє життя. Облік, планування та перерозподіл ресурсів разом з розумінням людської психології допомагає вирішувати питання прокрастинації, мотивації, самодисципліни;

ж) Постановка цілей;

з) Зосередження на найважливішому на певному шляху до цілі;

и) Досягнення результатів. Завдяки технікам тайм менеджменту підвищується продуктивність праці, яка в свою чергу мотивує робити далі краще і досягати максимум;

к) Підвищується оперативність. Задача, яка з кожним роком стає більш затребуваною разом із прискоренням часу. Наявність великою кількості різних за масштабністю справ, частіше одночасно. Тайм

менеджмент допомагає аналізувати перелік питань обов'язкових або зайвих, оцінювати значність [6].

Тайм менеджмент містить в собі такі дії:

л) постановка цілі;
м) створення плану;
н) надання пріоритету;
о) організація;
п) декомпозиція за потребою;
р) делегація ресурсів;
с) моніторинг та аналіз пройденого шляху;
т) створення власної системи на основі аналізу.

Можна побачити, що інструмент підходить для створення різноманітних планів різними за строком, сутністю. Плани можуть бути поділені на проекти, детальні плані або звичайний список задач. У свою чергу процеси перетворюються у план з список діл, розклад, календар [7].

Для запровадження тайм менеджменту, рекомендується зробити аналіз використовування часу та планування кожний рівнозначний період на основі того, що зроблено до цього періоду та що планується на наступний.

Насправді, це репрезентує одну з технік наданню пріоритету завданням до управління часом «ABCD analysis», або «АБВГ аналіз», або аналіз за принципом Ейзенхауера (рис. 1.1). Техніка, яка базується на створенні категорій великих даних на групи, який використовує принципи важливості та терміновості [8].

Матриця Ейзенхауера		
	Терміново	Нетерміново
Важливо	Зробити негайно	Зробити за планом
Неважливо	Делегувати	Ігнорувати

Рисунок 1.1 – Матриця Ейзенхауера

Завдання класифікуються наступним чином:

а) Тип «A» – важливі та термінові, не делегувати нікому та зробити як можна скоріше;
б) Тип «B» – важливі та нетермінові, не делегувати нікому, але установити строк або делегувати підлеглим та залишити час для контролю;
в) Тип «C» – неважливі та термінові, необхідно делегувати підлеглим, якщо їм не потрібно мати спеціальні знання та навички;
г) Тип «D» – неважливі та нетермінові, радять від діл цієї групи відмовитися взагалі.

Існують і інші техніки, наприклад, техніка Брайана Трейсі, фахівця із саморозвитку, в якій використовується метод «з'їсти жабу на сніданок», де жаба це найважче завдання на певний день, тоді решта справ завершиться практично без зусиль [7].

Техніка Парето - продовження техніки Брайана Трейсі, яка є корисною, коли за увагу змагається декілька напрямків задач. У кінцевому кроку аналізу вигідності кожної задачі, обирається саме та, яка принесе загальну вигоду. Цю техніку варто обмежити виключенням важливих проблем на розсуд

виконавця, які на перший погляд і на конкретний момент часу є не великими, але мають претензію зростати з часом.

Ця техніка більш корисною у поєднанні з аналітичними інструментами, такими як аналіз видів і наслідків відмов, який аналізує та виявляє найбільш критичні кроки виробничих процесів з метою управління якості, та аналіз “дерева відмов”, яка за допомогою ребр графу і логічних операторів пов’язує відмови елементів із відмовою об’єкта.

Також цю техніку називають правилом “80/20”, де лише 20% всіх справ надає 80% бажаного результату.

Техніка ABC-аналізу, є схожою на техніку Парето. Завдання поділяються на три групи, відповідно до їх значимості, що базується на трьох закономірностях:

- Тип «A», або найважливіші справи, які дорівнюють 15% загальної кількості від всіх справ. Справи надають близько 65% бажаного результату;
- Тип «B», або важливі завдання, які дорівнюють 20% від загальної кількості всіх справ. В свою чергу, значимість цих справ становить 20% бажаного результату;
- Тип «C», або неважливі завдання, які дорівнюють 65% від всієї кількості справ. Їх внесок становить 15% до здолання мети.

Можна побачити, що останні техніки включають в себе стороннє вирішення важливості завдання. У випадку з принципом Ейзенхауера, на етапі надання класифікації для групи «неважливо, терміново» потрібно додатково визначати ступінь підготовленості підлеглого, до якого буде делеговано завдання. Методика визначення цього аргументи полягає на розсуд користувача, що робить кожний існуючий випадок використання цих технік унікальним і підлеглим обширному аналізу.

Існує поєднання ABC-аналізу з принципом Ейзенхауера (рис. 1.2). Він не вирішує погрішність визначення важливості, але зменшує обсяг даних для вибору і аналізу.

A	Важливо	Та	Терміново
B	Важливо	Або	Терміново
C	Ані	Важливо	Ані	Терміново

Рисунок 1.2 – Поєднання ABC з принципом Ейзенхауера

Має такі три групи типу:

- Тип «A», або важлива та термінова. У реальному випадку, таких завдань на день не більше одного чи двох. Таку задачу чи задачі потрібно вирішити самому з підвищеною увагою;
- Тип «B», завдання яке або важливе та нетермінове, або неважливе та термінове. У цьому типі, власник завдання повинен або зробити штучний дедлайн для виконання задачі особисто, або делегувати підлеглому та залишити час для перевірки;
- Тип «C», завдання, яке не є важливими для різноманітних неділових цілей та має невизначений термін здачі. Такі здачі обов’язково повинні бути делеговані підлеглому.

Можна нехтувати варіантом типу «D» (рис. 1.3), які є неважливі та нетермінові, частіше всього це задачі не необхідні для ділових цілей або робочих, але якщо вони приносять задоволення або відпочинок, тоді їх можна утримувати.

D	Неважливо	Нетерміново	Не необхідно

Рисунок 1.3 – Тип "D"

Звичайний робочий день або вирішення ділових задач має містити або повинні виконуватися першими тільки ті завдання, які мають тип «A» та «B», або «A», «B» та «C» [8].

Можна роздивитися також систему методів управління часом ALPEN, заснованої на наборі принципів, які розкриваються під кожною буквою назви, які є акронімами німецьких слів:

- Aufgaben або «A», посилається на ваш список справ. Має схожості з типом «A» з техніки ABC, тобто справи пріоритетні та мають бути виконаними як можна скоріше;
- Länge або «L», представляє час, який розрахований для кожного завдання;
- Pufferzeiten або «P», представляє кількість часу, яка використовується як додатковий час для кожного завдання або проекту, аби було забезпечення виконавців завдання або проекту можливістю у разі форс-мажору дотримуватися строків;
- Entscheidungen або «E», представляє надання пріоритетів, щоб визначити важливі та нетермінові завдання, яким або виставити для власного вирішення справи строк, або делегувати підлеглому, коли це можливо;
- Nachkontrolle або «N», представляє аналіз розкладу розрахований на будь-який період на основі попередніх кроків, щоб визначити і зрозуміти, які дії та команди можна поліпшити, аби зробити більше продуктивним та результативним наступний період.

Можна побачити, що представлена схема керування часом дозволяє виконавцю робити свої завдання, коли він найбільш продуктивний та використовувати решту часу для виконання інших завдань або зробити перерву [9].

Наявні базові методи тайм менеджменту, які добре підходять для запровадження управлінням часу, такі як:

- GTD, Getting Things Done, або ж методика зробити так, щоб завдання були виконані, створена Девідом Аленом. Базовою ідеєю методики є закінчення всіх невеликих завдань якомога скоріше і розбиття великого завдання на малі. Причиною для пошуку такого роду рішення було застереження інформаційного перенасичення, коли список завдань становиться з кожним днем більше, а вирішується не так швидко. Практикою такою методики є викладення завдань та ідей на папері або ні та організувати їх якомога скоріше, щоб простіше було їх контролювати;
- Pomodoro, яка описується як фундаментальна техніка вимірювання часу та з 30-хвилинною тривалістю, де 25 хвилин відведено на роботу, а інших 5 хвилин на відпочинок, перерив. Також є рекомендація з частотою раз на чотири перериви, мати відпочинок в 15-30 хвилин;
- Організація списку завдань. Рекомендується створювати загальний список завдань та щоденний список завдань, який створюється з завдань з загального списку на певний день. Альтернатива це створювати списки неважливих завдань. Зазвичай завдання набувають пріоритети наступним чином: нумеруються в порядку їх важливості і робляться у цьому порядку, розміщуються технікою ABC, подібний метод до ABC, де до типу «A» відносять завдання на день, тип «B» на тиждень, типу «C» на місяць та метод надання пріоритету, де до типу «A» відносять небажані завдання, коли вони

зроблені, інші виконувати легше, а тип «B» та «C» можуть наслідувати ту ж ідею, але замість того, щоб обов'язково виконувати саме небажане завдання, його ухилятися, це дасть мотивацію зробити всі інші завдання, щоб не виконувати його [6].

1.2 Аналіз сучасних тенденцій

Системи тайм менеджменту часто включають додатки, які використовуються для відстеження робочого процесу. Додатки дають користувачам уявлення про робочу силу, це дозволяє контролювати та підвищувати продуктивність та процес керування часом. Система автоматизує процеси та аналіз, що дозволяє позбавитися застарілих методів планування та аналізування.

У той самий момент, тайм менеджмент має багато людських факторів неспроможності правильного відтворення і наслідків цієї неспроможності – прокрастинація. Всім знайомий стан, коли в процесі роботи чи навчання, будь-яке неочікуване невелике повідомлення, не обов'язково важливе, привертає всю вашу увагу та концентрацію, та через деякий час ви знаходите себе, наприклад, за гортаням стрічки новин. Це вирішується самодисципліною, а вона в свою чергу системою нагородження чи статистикою витрати часу на марні, незаплановані дії.

Існує багато додатків, які певною мірою реалізують ті чи інші методи тайм менеджменту, наприклад, часто це включає реалізацію повного методу Pomodoro чи використання тільки так званого time-boxing. Більшість людей використовують саме 25-хвилинний time-boxing, адже техніка Pomodoro включає в 25 хвилин навіть перевірку поштової скриньки, що зазвичай користувачі не роблять [10].

У цій справі, вагомим фактором успішного оволодіння навичками та використання управління часом за допомогою додатків є зручність

використання платформи. На це впливає адаптивність сайту та добрий UI/UX, простота взаємодії з функціоналом, кроссплатформеність. Окрім цього, для психологічної сторони питання важливим є можливість аналізу користування у вигляді статистики та система нагородження.

1.3 Аналіз існуючих рішень

1.3.1 Habitica

Веб-додаток для підвищення продуктивності, самовдосконалення та запровадження звичок з накладеною можливістю геймфікувати процеси та виконувати завдання, залишаючись вмотивованим і водночас розважаючись. Гра викладена у вигляді RPG, в якій гравець збирає такі предмети, як золото та броню, щоб стати потужнішим. Винагороди у свою чергу досягаються завдяки дотриманню реальних цілей у вигляді звичок, щоденників і завдань.

Додаток має такі функції:

- керування списком справ;
- керування контрольним списком завдань;
- керування контрольним списком;
- формування звичок;
- функціональність відстеження канбан-дошки;
- аналіз здорової та нездорової звички, які впливають на систему нагородження.

Додаток має такі переваги:

- соціальна мережа;
- грати, залишаючись відповідальним відстежуючи свої цілі та завдання;
- нагороди.

Додаток має такі недоліки:

- складний зовнішній вигляд сайту (рис. 1.4) ;

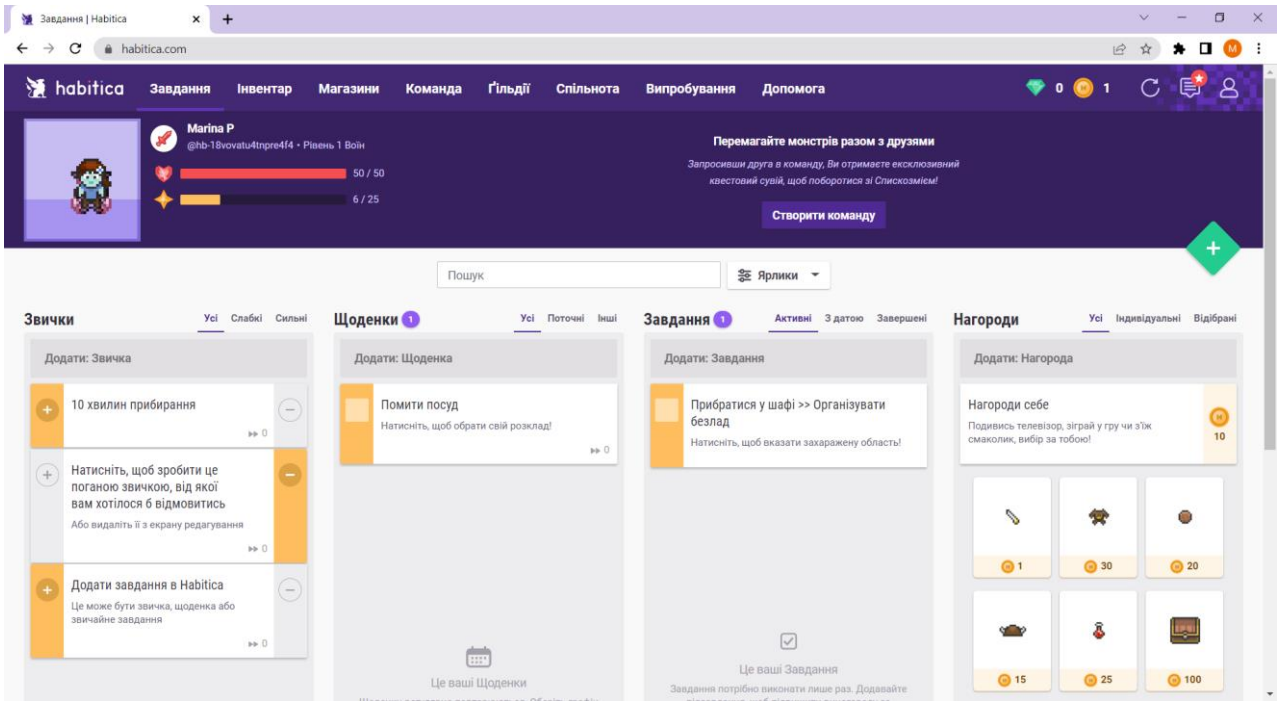

Рисунок 1.4 – Зовнішній вигляд сайту Habitica

- немає можливості створювати свої групи завдань;
- система нагородження має окремі нагороди у вигляді алмазів, які треба купувати за реальні гроші (рис. 1.5) ;

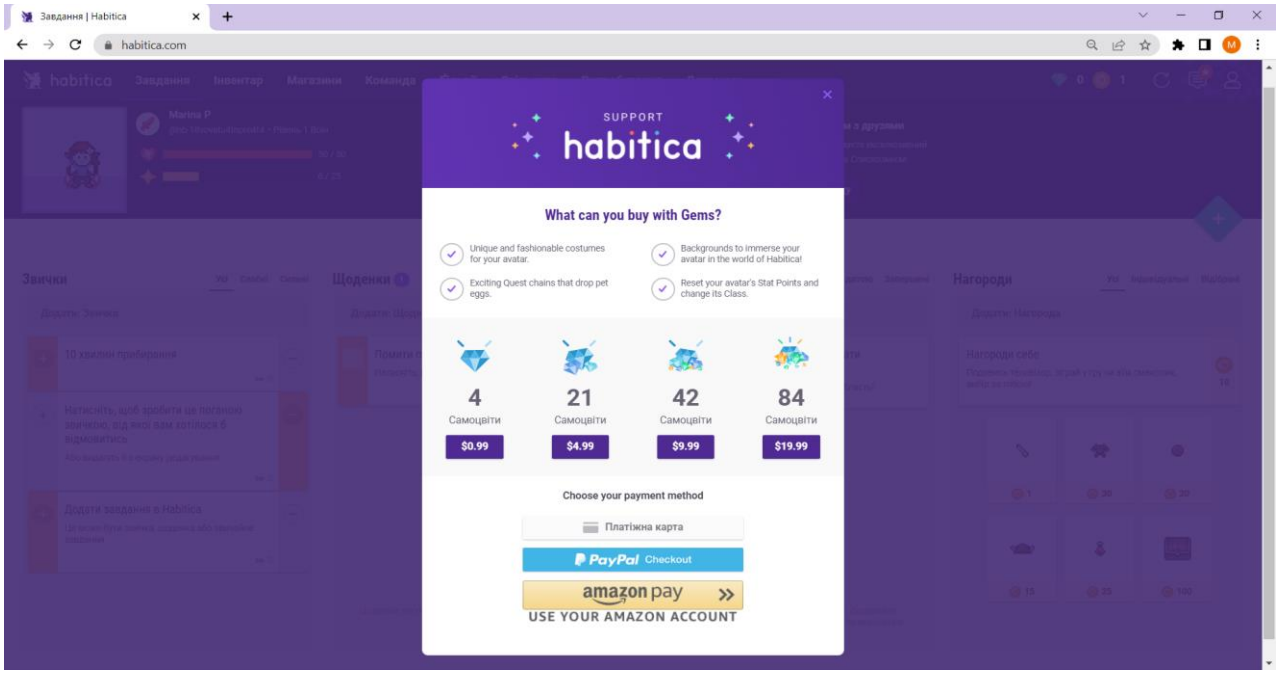

Рисунок 1.5 – Нагороди за гроші

- не підходить для вірного запровадження звички тайм менеджменту, так як може викликати ігрову залежність;
- система включає в себе соціальну мережу, але для того, щоб мати можливість грати і запроваджувати звички разом з друзями, треба активувати платний план.

Додаток існує на різних платформах, такі як мобільні додатки для iOS і Android та веб версію.

Додаток безкоштовний, тарифний план команди починається від 9 доларів на місяць [11].

1.3.2 Tweek

Програма для керування завданнями, яка підвищить вашу продуктивність. Цей інструмент побудований на основі перегляду тижневого календаря без погодинного планування (рис. 1.6).

Додаток має такі функції:

- щотижневий перегляд ваших завдань;
- контрольні списки та підзавдання;
- повторювані завдання;
- наклейки планувальника та кольорові теми;
- синхронізований календар Google.

Додаток має такі переваги:

- простий щотижневий планувальник паперу, який підвищить вашу продуктивність;
- автоматично переносе ваші незавершені завдання на наступний день;
- дозволяє запускати кілька календарів одночасно;
- ділитися своїми особистими та робочими календарями з іншими.

Додаток має такі недоліки:

- відсутня можливість створювати списки груп або папок.

Додаток існує на різних платформах, такі як мобільні додатки для iOS і Android та веб версію.

Додаток безкоштовний, але є преміальний варіант за 3 доларів на місяць [12].

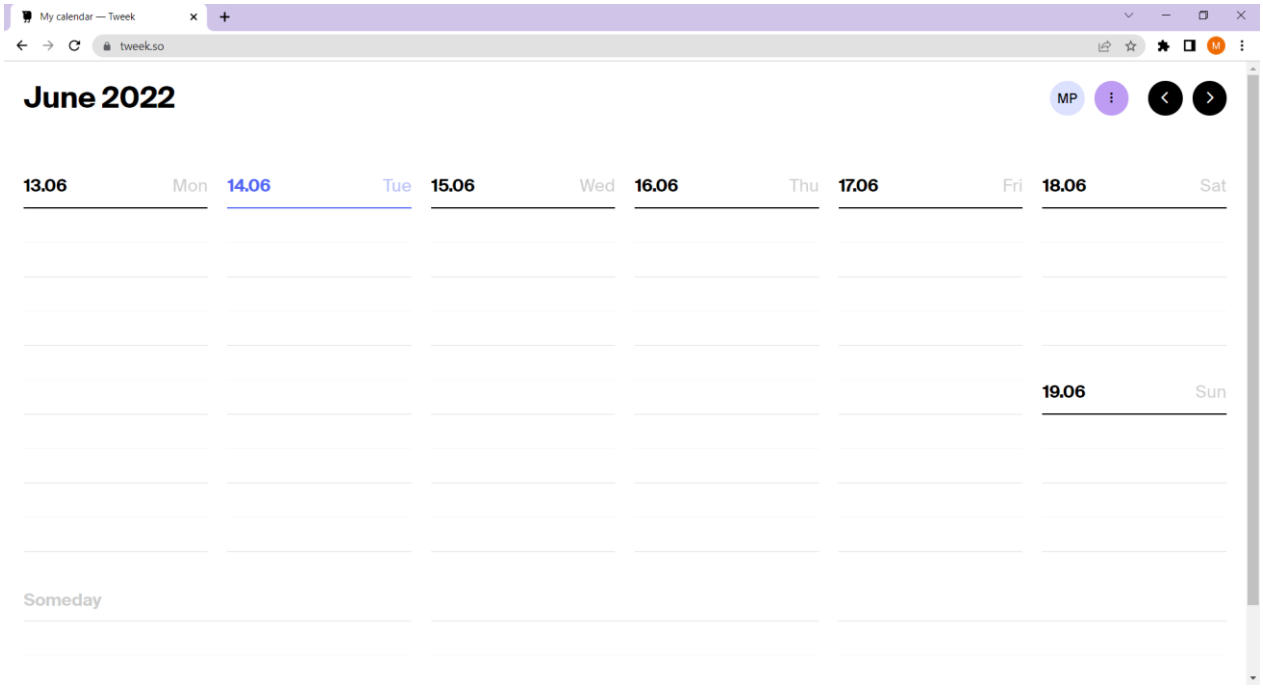

Рисунок 1.6 – Зовнішній вигляд сайту Tweek

1.3.3 MyLifeOrganized

Інструмент для керування завданнями, проектами та звичками (рис. 1.7). Створений для балансування між простим і складним. Після завантаження інформації інструмент створить простий список, який містить лише ті дії, які потребують вашої негайної уваги. Це найкраще підходить власникам бізнесу, керівникам проектів та IT-фахівцям.

Додаток має такі функції:

- список справ;
- відстеження часу;
- мобільний доступ;
- створення підзадач.

Додаток має такі переваги:

- велика кількістю функцій;

- поставляється з додатковими контактами, тегами, нагадуваннями та завданнями;
- налаштовані фільтри та перегляди відповідно до потреб;
- розбиття проекту на менші завдання.

Додаток має такі недоліки:

- ціна висока для звичайного користувача;
- за інші платформи, наприклад, Android потрібно платити окремо;
- окрема оплата синхронізації;
- не має функції календаря.

Додаток існує на різних платформах Android, iOS та Windows, але не має веб версії.

Додаток платний: професійний план коштує 59,95 доларів на місяць, стандартний план 49,95 доларів на місяць, стандартне оновлення до Pro 10 доларів [13].

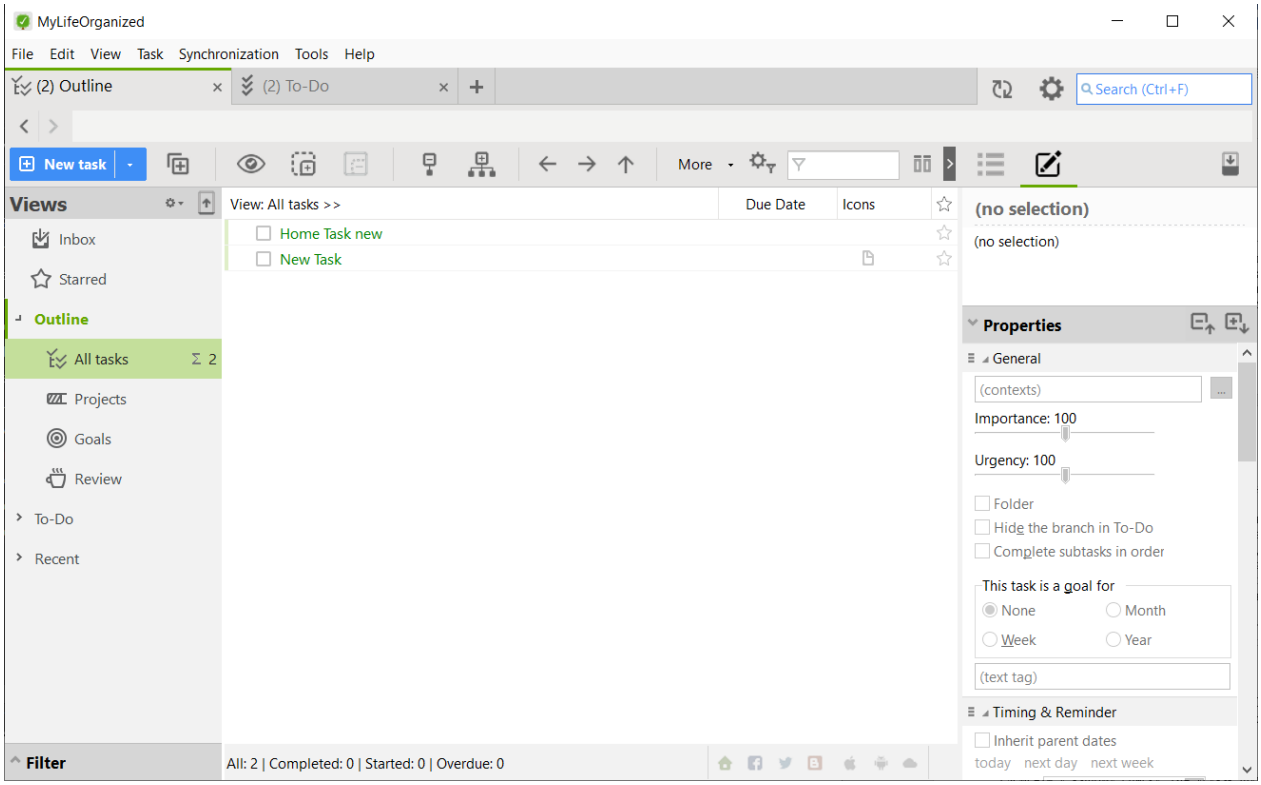

Рисунок 1.7 – Зовнішній вигляд програми My Life Organized

1.3.4 Clear

Це проста програма для списку справ, яка дозволяє вам викреслювати елементи у вашому списку (рис. 1.8). Він усуває шум і відволікає завдання, щоб ви почувалися більш продуктивними.

Додаток має такі переваги:

- простота у використанні.

Додаток має такі недоліки:

- доступний лише на платформі системи iOS.

Додаток безкоштовний, має преміум-план, який коштує від 4,99 доларів [14].

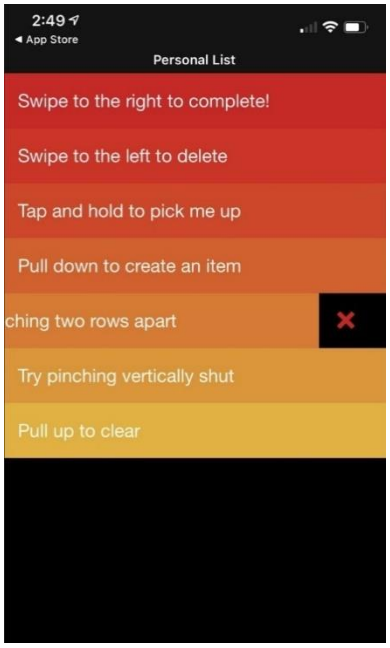

Рисунок 1.8 – Зовнішній вигляд програми Clear

1.4 Постановка задачі

Тема тайм менеджменту зроблена та досліджена в дуже широкому обсязі. Наскільки глибоко у неї занурюються, тим більше видніша потреба в вірних інструментах для вирішення питань. Наразі існують багато різних, всеосяжних рішень у вигляді інформаційних систем, які у більшості випадків можуть бути правильно і по повній використані тільки людьми, які мають розуміння тайм менеджменту і мають якісь навички.

Метою цієї роботи є вирішення таких задач:

- дослідження поняття тайм менеджменту;
- дослідження технік та методів тайм менеджменту;
- аналіз мінімального обсягу методів для дієвого запровадження і вивчення тайм менеджменту;
- створення зручної інформаційної системи, де реалізовані інструменти для використання методів тайм менеджменту.

1.5 Висновки по першому розділу

У першому розділі розглянуто питання тайм менеджменту. Основні техніки та методи управлінням часом. Проведено дослідження існуючих рішень по керуванню часу, їх доступність, зручність у використанні. Поставлено задачу у створенні зручної інформаційної системи, яка дозволить використати базові методи тайм менеджменту.

РОЗДІЛ 2. ПРОЕКТНІ РІШЕННЯ

Створення додатку бере початок з огляду проблеми і огляду методів для її вирішення. Від аналізування цільової аудиторії та поширеності вирішуваної проблеми починається процес розробки.

Наступними кроками є результат аналітичного етапу – запровадження функціональності додатку, розробка архітектури, яка включає в себе бізнес-логіку, масштабування додатку, тобто платформ розробки. Від попереднього кроку залежить технологічний спектр, кожна можлива платформа потребує власного методу проектування, розробки та певні інструменти, які для неї підходять.

В даному розділі розглядається методи розробки та інструменти та реалізація додатка, моделювання і проектування додатка та реалізація.

2.1 Огляд методів розробки

2.1.1 Огляд платформи для додатка

Сучасні тенденції розробки і користування проекту включає в себе обрання платформи та технологій. Даний проект має вимоги мати постійний доступ, легкий доступ до бази даних та легкість оновлення та експлуатації функцій додатку, доступність у грошовому плані для розробки та використання.

За даними статистики на травень 2022 року кількість користувачів мобільними додатками займає лідерську позицію з 59%, на противагу 38% користувачів десктопних додатків [15]. Доступність, мобільність – головні

помічники рушія сьогодення людства. Розгляд переваг і недоліків різних платформи.

Веб-додатки мають такі переваги порівняно з іншими програмними забезпеченням:

- легкість оновлення та додавання функцій;
- зі сторони користувача відсутня проблема технічної можливості пристрою входу для використання додатку;
- відсутність обмежень у просторі зберігання даних для користувачів;
- доступність додатку та даних з будь-яких платформ і місць розташування запитувача;
- відсутність потреби у дорогих матеріальних інвестиціях для розробки та підтримки продукту.

Недоліки вказаної платформи такі, як:

- відсутність можливості взаємодії повною мірою функціями смартфона, такі як дані місцезнаходження, фотокамера або відеокамера;
- швидкість завантаження даних у веб-додатку має залежність від якості інтернет поєднання.

Мобільні додатки на противагу мають такі переваги:

- немає обмежень, окрім обмежень дозволу користувача, до функцій та можливостей функцій пристрою;
- можливість використовування додатку в автономному режимі;
- зберігання даних користувача на пристрої прискорює обмін даними та доступ до них.

Недоліки мобільних додатків:

- доступ до мобільного додатку залежить на першому кроці його доступності на платформі маркету на основі операційної системи, якою користується клієнт;
- складність виявляється у технічних можливостях користувацького пристрою, тобто, з кожним оновленням мобільного додатку, може знадобитися більша потужність апарату;
- розташування мобільного додатку залежить від операційної системи, дві найпопулярніші з них – операційна система Android, яка базується в корінні на Linux та інших відкритих системах та iOS, операційна система створена Apple. В свою чергу кожна система потребує власної мови програмування, тобто для кожної системи потрібно розробляти дві різні програми, вартість розробки, підтримки та обслуговування збільшується;
- додаток аналізується кожним маркетом, не завжди є можливість, що додаток можна буде розмістити на обраному маркеті;
- шлях до мобільного додатку потребує заходження на маркет, пошук додатку та завантаження.

Перечисленні обмеження впливають на те, щоб додаток дістався своєму кінцевому користувачеві та на використання його.

Можна побачити, що для даного проекту вистачає функцій та можливостей веб платформи, тому обрано розробку веб додатку.

2.1.2 Огляд технологій для моделювання системи

Як було зазначено раніше, одним з етапів розробки додатку є його моделювання та проектування. Процес аналізування додатку потребує вірного і в різних масштабах представлення кожного рівня програми. Вирішена задача специфікації, проектування та візуалізації підсистем

представляє структуру програми, яка закриває усі потреби в функціонування. Якщо розглядати методи моделювання системи можна виділити дві технології BPMN та UML.

BPNM, або модель бізнес-процесів і нотація, це стандарт представлення бізнес-процесів у вигляді графічного показника у вигляді блок-схем. Головною задачею системи є підтримка управління процесам, як для бізнес-користувачів, так і для технічних.

Дана система має ряд слабких сторін такі як, двозначність і плутанина використання моделей у спільному доступі та відсутність підтримки правил ведення бізнесу та прийняття рішень [16].

Уніфікована мова моделювання, або UML, яка призначена для забезпечення стандартного способу візуалізації дизайну та архітектури програми. Спосіб візуалізації базується на представлені процесів та ролей, методів у вигляді діаграм.

Існує багато різних типів діаграм, які можна поділити на дві категорії – структурний тип та поведінковий тип. Структурний засіб представляє статичну частину системи, яка виділяє речі, які повинні бути присутніми у системі. Діаграма поведінки представляє динамічний аспект, який виділяє те, що має відбуватися в системі [17].

Виходячи з цього огляду двох систем моделювання, для даного проекту краще використовувати UML.

2.1.3 Аналіз технологій розробки клієнтської сторони

Один з важливих шагів створення проекту вибір технологій, які будуть мати вплив на стабільність, стійкість системи, та вірний простір UI/UX сферу. Існує багато рішень та технологій за допомогою яких можна

створювати клієнтську частину для веб-додатку. Клієнтська сторона включає в себе те, що користувач буде бачити. Це в свою чергу впливає на комунікацію системи з ним та взаємодію. Найпопулярнішим вирішенням є JavaScript.

JavaScript мова високого рівня, яка може компілюватися в «той самий момент» та відповідає стандарту ECMAScript. Часто кооперується та поширюється за допомогою сторонніх бібліотек та фреймворків. Головною перевагою цієї мови є те, що веб-браузери мають двигун для компіляції коду.

Це багатопарадигмальна мова, яка може застосуватися різними стилями програмування. Має інтерфейс прикладного програмування, зазвичай саме так працює на практиці веб-браузер або інша система виконання, яка надає інтерфейс JavaScript для введення-виведення, який дозволяє працювати з різними типами даних, регулярними виразами, об’єктною моделлю документів, тобто DOM, яка представляє фундаментальну роботу з мовою розмітку гіпертексту HTML.

Механізми JavaScript є основними компонентами деяких серверів і різноманітних програм, окрім роботи з веб-браузером. Однією з цих систем є Node.js. В свою чергу, це дозволяє використовувати JavaScript не тільки для розробки клієнтської сторони.

Існування багатьох рішень у вигляді бібліотек та фреймворків робить цю мову популярною. Задля запобігання створення «велосипедів», використання різноманітних рішень від реалізацій кастомного способу ведення даних до створення цілих сторінок певних типів, використання таких рішень є вірною практикою [18].

Найпопулярнішими рішеннями є React та Angular.

React це бібліотека з відкритим кодом, яку зазвичай використовують для побудови користувацьких інтерфейсів та компоненти інтерфейсів. Це компонентний та декларативний підхід дозволяє створювати легкі та

інтерактивні інтерфейси. Це дає можливість створювати швидкі та добре масштабовані рішення для багатьох платформ, діючи за принципом «вивчи один раз, пиши для всього».

React використовує синтаксичне розширення JSX, яке нагадує мову розмітки гіпертексту. Це показує те, що React має змогу реалізувати дійсний факт не роз’єднання логіки візуалізації з логікою інтерфейсу користувача та різних маніпуляцій з даними – обробка подій, зміна стану з часом і підготовка даних до відображення. Це дає змогу не прописувати логіки у різних файлах, а створювати програму за допомогою компонентів, які містять і розмітку, і функцію.

Angular це фреймворк для розробки, який створений на основі TypeScript, мові програмування, яка являє собою строгий синтаксичний наднабір JavaScript. Фреймворк базується на компонентному підході, має великий набір інструментів, що дозволяють будувати різні за розміром проекти.

Частіше Angular використовується для створення складних програм корпоративного рівня, такі як SPA, або односторінкові програми та прогресивні веб-програми. Фреймворк має велике спільноту та має об’єктно орієнтований підхід. У той самий момент, React використовується для створення, базуючись на компонентному підході, інтерфейси, де часто змінюються дані. React має зручний, гнучкий метод розробки, який підходить для невеликих команд.

Виходячи з освітлених фактів, для даного проекту підходить бібліотека React.

2.1.4 Аналіз технологій розробки серверної сторони

На основі обраних інструментів для розробки та мети проектного рішення, обрано серверну платформу Node.js. Спеціальна платформа, розроблена масштабованих мережевих проектів. Це асинхронне, тобто паралельно виконуюче запроси користувачів чи систем, кероване подіями середовище виконання JavaScript.

Node.js побудована на Google Chrome V8 движку, що компілює код у машинний код. Треба зазначити, що ця платформа не є фреймворком чи бібліотекою, це середовище виконання. Асинхронність Node.js означає, що під час операцій вводу-виводу, основний потік запитів не буде заблоковано. Він продовжуватиме виконувати запроси паралельно. Також є поняття зворотних викликів, або callbacks. Це функції, які запрограмовані виконуватися у майбутньому після певних дій. Перша, коли сервер отримує запит, друга, коли файл буде прочитано та контент буферізовано.

2.2 Проектування і моделювання системи

2.2.1 Розробка моделей архітектури

Оглянутий та вибраний стек технологій представляє собою fullstack-розробку, тобто розробку клієнтської сторони, серверної сторони та бази даних і зв’язок між ними. Це представляє собою так звану трирівневу модель архітектури (рис. 2.1).

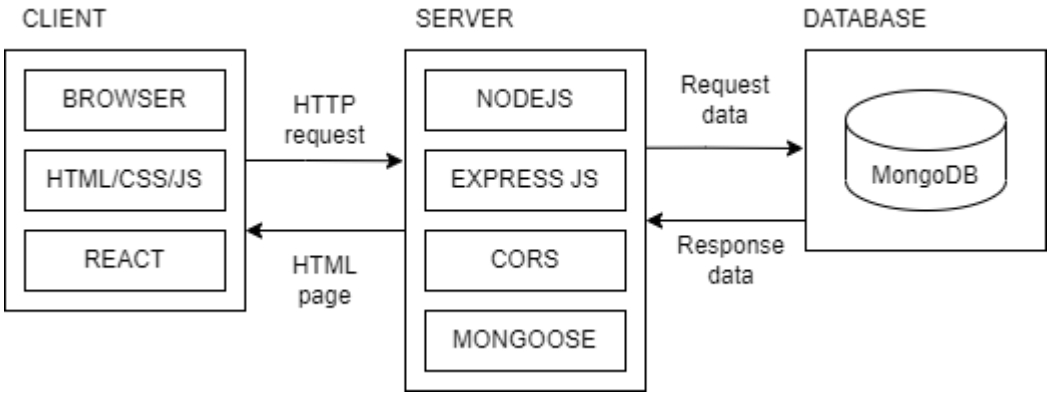

Рисунок 2.1 – Структура 3-tier архітектури

Трирівнева архітектура, яка поділяє програми на три рівні:

- веб-сервер, у якому інтерфейс — це вміст, який відображається браузером;
- середній динамічний сервер обробки вмісту та рівень генерації;
- база даних, яка містить не тільки набори даних, але й розроблену логіку керування.

Є однією з популярних рішень для проектування програмного забезпечення для традиційних програм клієнт-сервер (рис. 2.2).

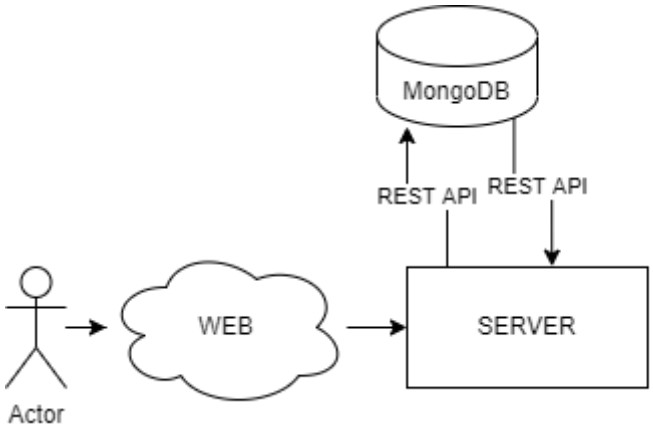

Рисунок 2.2 – Модель клієнт-сервер

У веб-розробці рівні мають різні назви, але виконують схожі функції.

Веб-сервер є рівнем презентації та забезпечує інтерфейс користувача. Зазвичай це веб-сторінка або веб-сайт, наприклад сайт електронної комерції, де користувач додає товари в кошик для покупок, додає інформацію про оплату або створює обліковий запис. Вміст може бути статичним або динамічним і розробляється за допомогою HTML, CSS і Javascript, React.

Сервер додатків відповідає середньому рівню, в якому міститься бізнес-логіка, яка використовується для обробки введених користувачів. Щоб продовжити приклад електронної комерції, це рівень, який запитує базу даних інвентарю, щоб повернути доступність продукту, або додає деталі до профілю клієнта. Цей рівень розробляється з використанням Node.js та фреймворку Express.js.

Сервер бази даних — це рівень даних або серверний рівень веб-додатка. Він працює на програмному забезпеченні для управління базами даних, в випадку цього проекту і обраного стеку, MongoDB.

Структура має ряд переваг, наприклад, надає велику свободу оновлювати або замінювати лише окремі частини програми, не впливаючи на продукт в цілому. Це дозволяє досить легко розширити та розширити програму, відокремивши інтерфейсну програму від баз даних, вибраних відповідно до індивідуальних потреб клієнта, завдяки цьому той самий час критичні компоненти програми можуть бути інкапсульовані та збережені, поки вся система продовжує органічно розвиватися.

Діаграми компонентів використовуються при моделюванні фізичних аспектів об’єктно-орієнтованих систем, які використовуються для візуалізації, визначення та документування систем на основі компонентів, а також для побудови виконуваних систем шляхом прямого та зворотного проектування. Діаграми компонентів, по суті, є діаграмами класів, які

зосереджуються на компонентах системи, які часто використовуються для моделювання статичної реалізації системи.

Діаграми розгортання використовуються для візуалізації топології фізичних компонентів системи, де розгорнуті програмні компоненти та для опису статичного вигляду розгортання системи, вони складаються з вузлів та їх взаємозв'язків. Сам термін розгортання описує мету діаграми. Діаграми розгортання використовуються для опису апаратних компонентів, де розгорнуті програмні компоненти. Діаграми компонентів і діаграми розгортання тісно пов'язані.

Діаграми компонентів використовуються для опису компонентів, а діаграми розгортання показують, як вони розгорнуті в апаратному забезпеченні.

Діаграма компонентів даного проекту зображає клієнтську сторону з програмою React.js, серверну стороні з Node.js та базою даних (рис. 2.3).

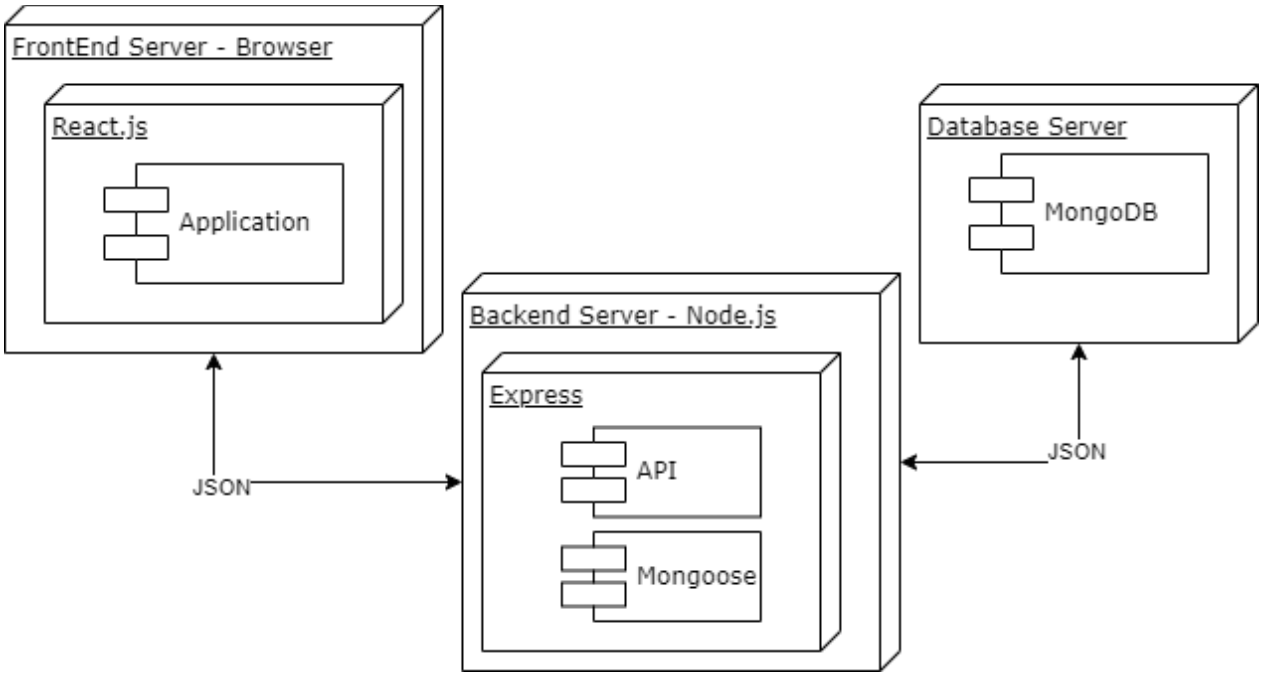

Рисунок 2.3 – Діаграма компонентів

UML в основному призначений для зосередження на програмних артефактах системи. Однак дві діаграми є спеціальними діаграмами, які використовуються для фокусування на програмних та апаратних компонентах.

Більшість діаграм UML використовуються для обробки логічних компонентів, але діаграми розгортання створені для зосередження на апаратній топології системи.

Дана діаграма розгортання представляє три компоненти апаратної топології системи: сервер на боці клієнта, веб сервер та база даних (рис. 2.4).

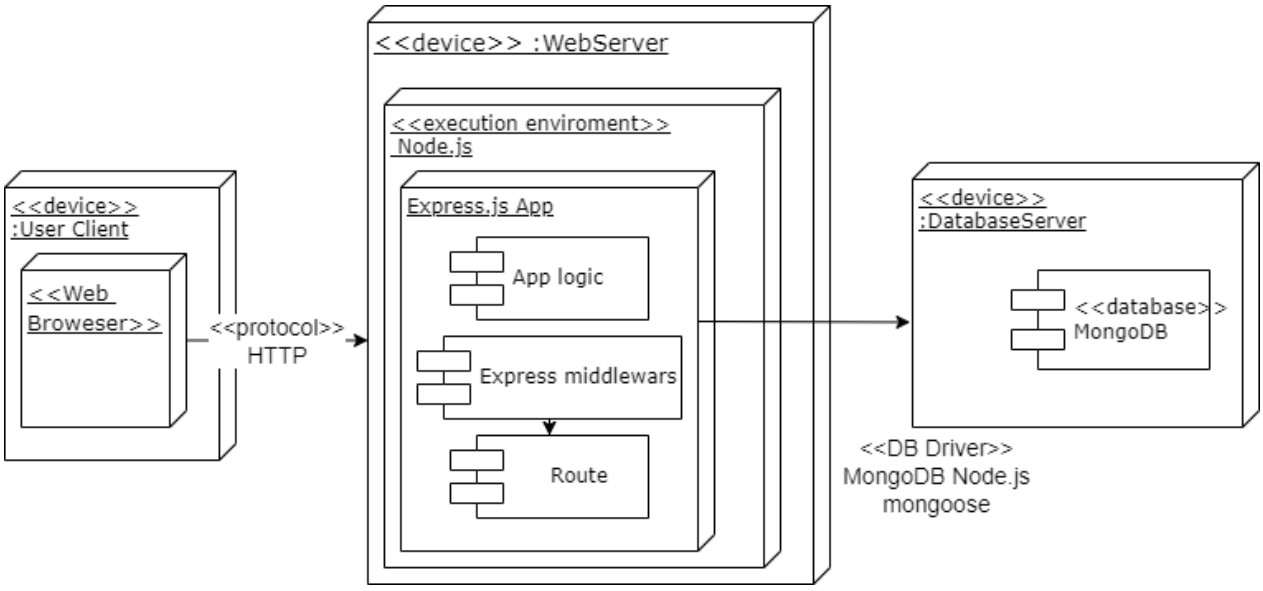

Рисунок 2.4 – Діаграма розгортання

React.js заснований на односторінковому додатку (SPA), і це дозволяє уникнути завантаження нової сторінки з кожною дією і значно спрощує роботу користувачів.

2.2.2 Проектування додатку

На основі постановки завдання, були сформовані такі функції до інформаційної системи:

- створення користувача;
- створення груп;
- створення завдання;
- перегляд завдання;
- завдання власних значень для таймеру Pomodoro.

Діаграма прецедентів, це моделювання взаємодії користувача з системою. Може представляти різні варіанти використання, а також різні за рівнем доступності до системи користувачами. Виконувані дії представлені або колами, або еліпсами. Актори часто зображені у вигляді фігурок. Діаграми прецедентів допомагає у візуалізації функціональних вимог системи, що вплине на вибір дизайну та пріоритети розробки.

Надані діаграми прецедентів для даного проекту з переліком всіх можливих дій (рис. 2.5, 2.6, 2.7, 2.8).

Рисунок 2.5 – Діаграма прецедентів

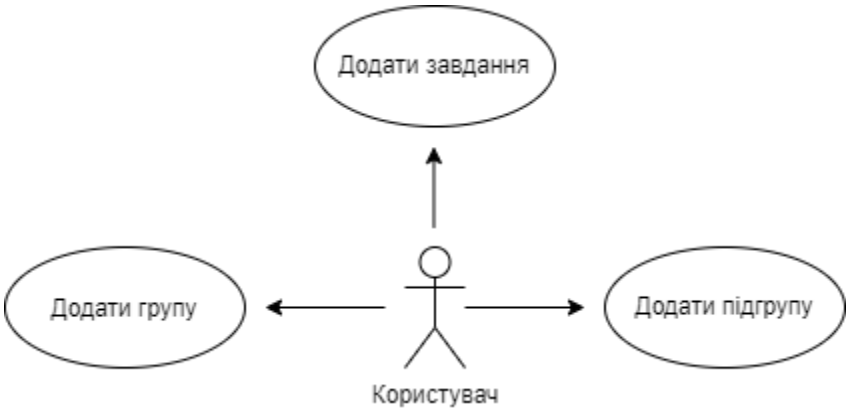

Рисунок 2.6 – Діаграма прецедентів

Рисунок 2.7 – Діаграма прецедентів

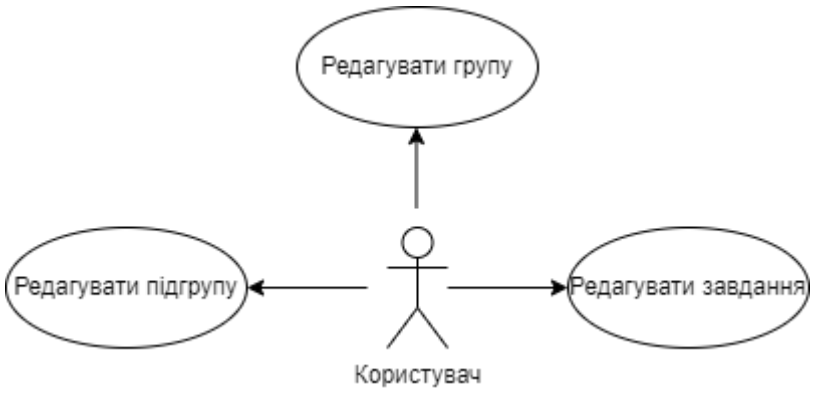

Рисунок 2.8 – Діаграма прецедентів

За статичний вигляд програми відповідає діаграма класів. Він представляє типи об’єктів у системі та зв’язки між ними. Класи складаються

з власних об'єктів і можуть успадковуватися від інших класів. Діаграми класів використовуються для візуалізації та опису системи, також дає змогу конвертувати діаграму у код для будь-якої мови.

Він показує атрибути, класи, функції та відносини, щоб дати загальне уявлення про програмну систему. Він містить імена класів, атрибути та функції в окремому відсіку, який допомагає у розробці програмного забезпечення. Оскільки це набір класів, інтерфейсів, асоціацій, співпраці та обмежень, його називають структурною діаграмою.

В UML відносини бувають трьох типів:

- залежність — це семантичний зв’язок між двома або більше класами, коли зміна в одному класі спричиняє зміни в іншому класі. Це формує більш слабкі відносини;
- узагальнення – це зв’язок між батьківським класом (суперкласом) і дочірнім класом (підкласом). У цьому дочірній клас успадковується від батьківського класу;
- асоціація - описує статичний або фізичний зв'язок між двома або більше об'єктами. Він показує, скільки об’єктів є у відносинах.

Кратність визначає певний діапазон допустимих екземплярів атрибутів. Якщо діапазон не вказано, кратність за замовчуванням розглядається як одиниця.

Агрегація — це підмножина асоціацій, яка представляє зв’язок. Це конкретніше, ніж асоціація. Він визначає відношення частина-ціле або частина. У таких відносинах дочірній клас може існувати незалежно від свого батьківського класу.

Композиція є підмножиною агрегації. Він зображує залежність між батьком і його дочірнім, що означає, що якщо одна частина буде видалена, то інша частина також відкидається. Він представляє відносини ціла частина.

Діаграма класів даного проекту має третій тип відносин з композицією, тобто якщо користувач видаляється з бази, то видаляються всі створені ним групи та завдання. Також, якщо користувач створюється, в нього можуть бути присутніми безліч груп, або ніскільки, але якщо створюється підгрупа у групи, або завдання у підгрупі, то існувати вже може від одного до безліч у батьківському класі (рис. 2.9).

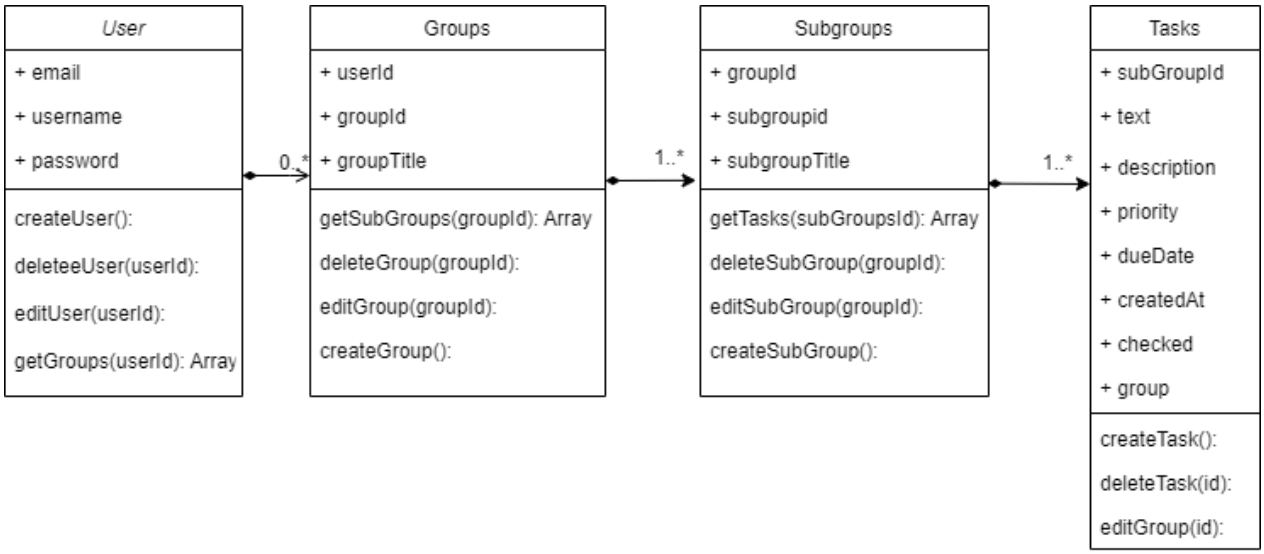

Рисунок 2.9 – Діаграма класів

Діаграма об'єктів залежить від діаграми класів, оскільки вона є похідною від діаграми класів. Вона представляє екземпляр діаграми класів. Об’єкти допомагають відобразити статичний вигляд об’єктно-орієнтованої системи в певний момент. Діаграма об’єктів відображає загальну поведінку компонентів.

І діаграма об'єкта, і діаграма класів до певної міри схожі, єдина відмінність полягає в тому, що діаграма класів надає уявлення про систему. Це допомагає у візуалізації певної функціональності системи, але діаграма класів моделює класи, їх методи, доступність та зв’язки між собою. Діаграма об’єктів моделює лише екземпляр.

Діаграма об’єктів даної системи (рис. 2.10).

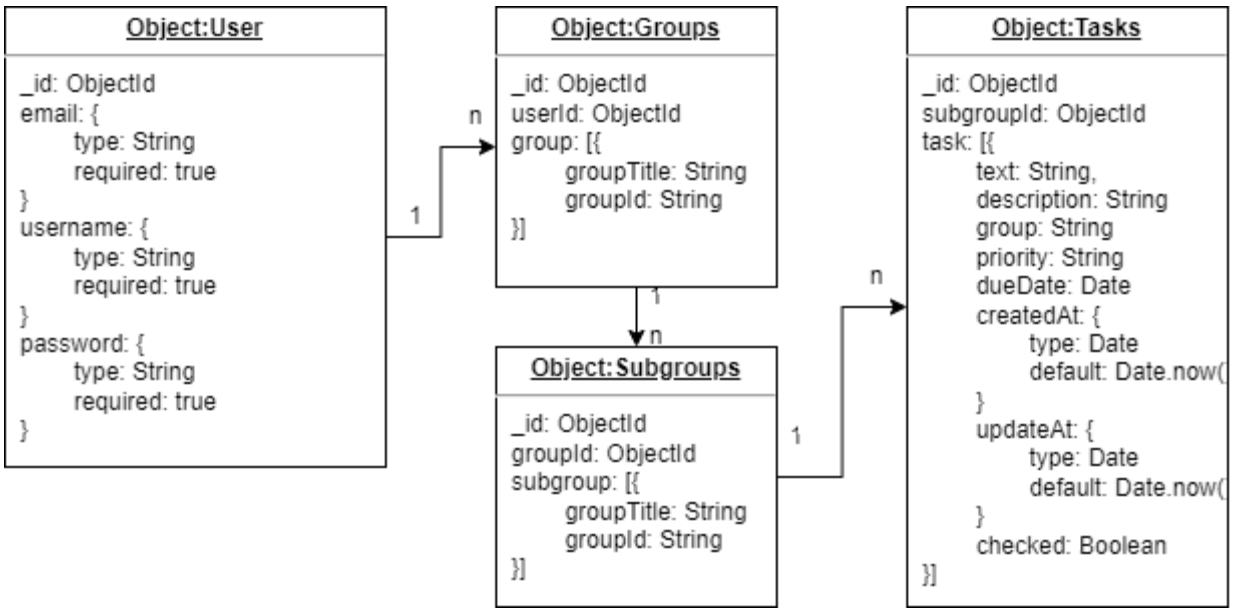

Рисунок 2.10 – Діаграма об’єктів

2.2.3 Розробка прототипу екранів сайту

Прототип — це цифрове моделювання або демонстрація продукту чи послуги, що дає змогу перевірити припущення та віртуально досліджувати продукт перед його створенням. Прототипи можуть приймати різні форми, від робочого зразка до моделі чи моделювання цифрового продукту чи послуги, на основі яких можна розробити мінімально життєздатний продукт та його варіації.

Прототип вхідної сторінки для входу або реєстрації користувача (рис. 2.11).

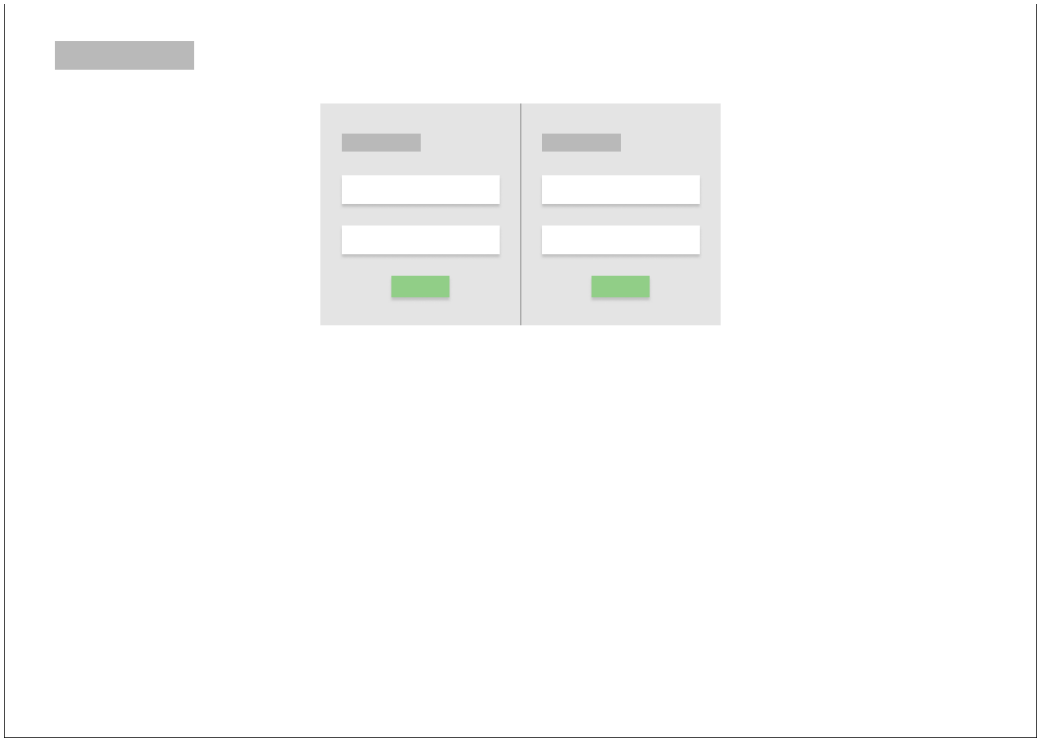

Рисунок 2.11 – Прототип вхідної сторінки

Прототип головної сторінки, де за замовчуванням відтворюється перелік створених груп (рис. 2.12). З початку роботи можна створити групу, створивши групу можна створювати підгрупу і у підгрупі створювати завдання. Групи, завдання, підгрупи можна редагувати та видаляти.

Є можливість натиснути на групу і відтворити перелік завдань та можливість обрати завдання і подивитися більш детальну інформацію. Також присутня інформація про користувача та можливість вийти з облікового запису.

Рисунок 2.12 – Прототип головної сторінки

Прототип модального вікна створення завдання з полям для веденням даних та кнопками прийняття та скасування дій (рис. 2.13).

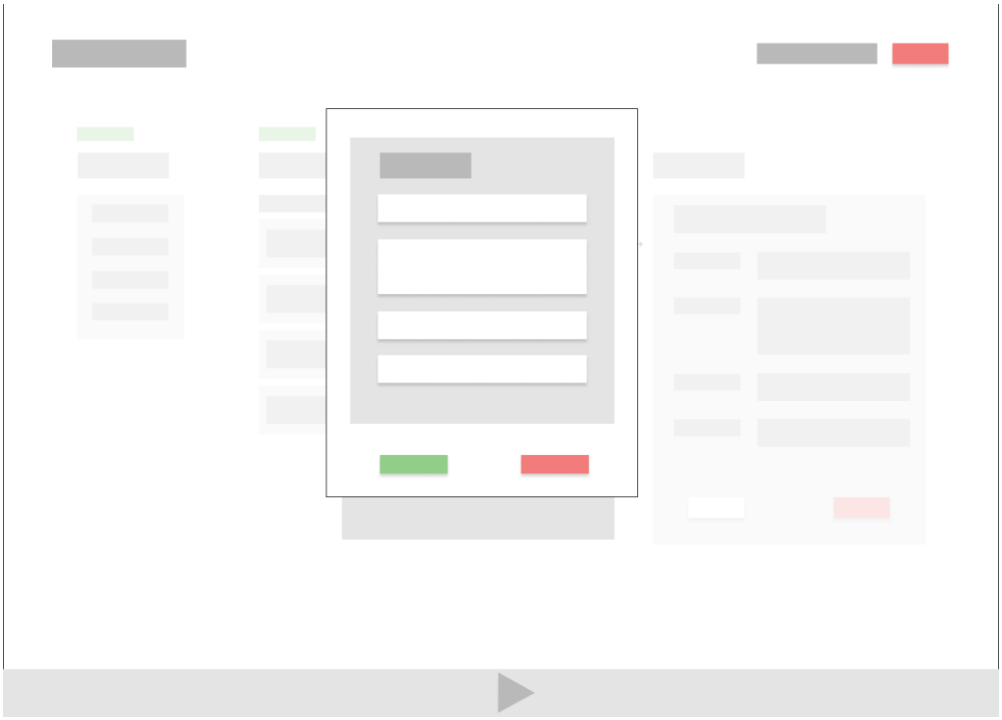

Рисунок 2.13 – Прототип модального вікна

2.3 Програмна реалізація

2.3.1 Створення серверної частини

Початок реалізації додатку починається зі створення серверної частини. Мануально встановлюється файл залежностей (dependencies file) з налаштуваннями по замовчуванню за допомогою команди:

npm init –yes

Цей документ — це все, що потрібно знати про те, що потрібно у вашому файлі package.json. Це є фактичний JSON, а не просто літерал об’єкта JavaScript. Тут можна побачити всі підключені бібліотеки, які використовуються у розробці серверної частини, автора, головний загрузочний файл та сценарії (scripts) за допомогою яких буде запускатися сервер.

На більшість поведінки, описаної в цьому документі, впливають налаштування конфігурації, описані в config.

Встановлення залежностей, а саме бібліотек, відбувається завдяки команді:

npm install [назва бібліотеки] [назва бібліотеки]

Для розробки серверу були використані наступні модулі (рис. 2.14):

- cors – це механізм на основі HTTP-заголовка, який дозволяє серверу вказувати будь-які джерела (домен, схему або порт), крім свого власного, з якого браузер повинен дозволяти завантажувати ресурси;
- dotenv – це модуль нульової залежності, який завантажує змінні середовища з файлу .env у process.env;
- express – це фреймворк Node.js, розроблений для швидкого створення API і полегшення роботи вузла js;
- mongoose – це інструмент моделювання та керування об’єктів MongoDB;
- nodemon – це інструмент, який допомагає розробляти програми на основі Node.js шляхом автоматичного перезапуску програми вузла, коли виявлено зміни файлів у каталозі;
- jsonwebtoken – модуль, який використовується як спосіб безпечної передачі інформації між сторонами як закодований об’єкт JSON, включаючи набір вимог;
- bcryptjs – є функцією хешування паролів;
- body-parser – аналіз тіла вхідних запитів у проміжному програмному забезпеченні перед обробниками, доступними у властивості req.body.

server > {} package.json > ...

```json
{
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.0.1",
    "express": "^4.18.1",
    "mongoose": "^6.3.6",
    "nodemon": "^2.0.16",
    "jsonwebtoken": "^8.5.1",
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.20.0"
  },
  "name": "server",
  "version": "1.0.0",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Рисунок 2.14 – Вихідний код package.json

Складові елементи серверного боку (рис. 2.15) для Завдань, де:

- config – файл з сервісами, послугами проекту;
- routes – це файл зі всіма шляхами, «маршрутами», із зазначеними функціями зворотного виклику (рис. 2.16);
- models – спроектовані таблиці бази даних (рис. 2.17);
- controllers – файл з відповідними функціями зворотного виклику для шляхів (рис. 2.18);

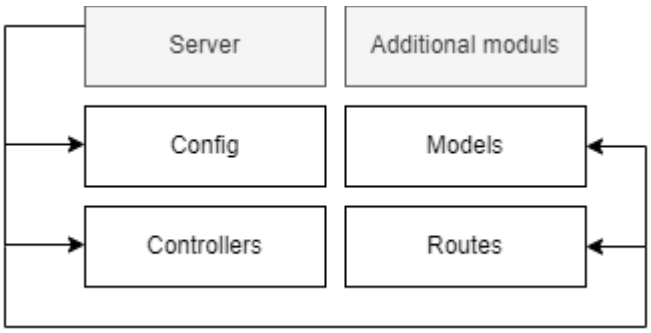

Рисунок 2.15 – Файлова структура папки server

server > routes > JS task.js > ...

```
1   const express = require("express");
2   const router = express.Router();
3   const { getTask, createTask, updateTask, deleteTask, }
4       = require("../controllers/task");
5
6   router.post("/tasks", createTask);
7   router.get("/tasks", getTask);
8   router.put("/:id", updateTask);
9   router.delete("/:id", deleteTask);
10
11  module.exports = router;
```

Рисунок 2.16 – Вихідний код routes.js

server > models > JS task.js > ...

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

const tasksSchema = new mongoose.Schema({
  groupId: String,
  task: [
    {
      id: String,
      text: String,
      description: String,
      group: String,
      priority: String,
      dueDate: Date,
      checked: Boolean,
    },
  ],
});

const Task = mongoose.model("Task", tasksSchema);

module.exports = Task;
```

Рисунок 2.17 – Вихідний код models.js

server > controllers > JS task.js > ...

```
const Task = require("../models/task");
const Group = require("../models/group");

exports.getTask = async (req, res) => {
    const tasksItems = req.body;
    const group = await Group.findOne(tasksItems.group).exec();

    if (!group) {
        res.status(500)
            .json({
                message: "Group doesn't exist!",
            });
        return;
    }

    const { task } = await Task.findOne({ groupId: group._id }).exec();
    res.json(task);
};
```

Рисунок 2.18 – Вихідний код controllers.js

2.3.2 Реалізація бази даних

На основі діаграми об’єктів можна спроектувати базу даних. Для початку для зручності в окремому файлі створюється підключення до бази даних. Використовується метод require() і підключається методом бібліотеки mongoose.connect(), указуючи як перший аргумент посилання на базу даних (рис. 2.19).

server > config > JS mongo.js > ...

```
1   const mongoose = require('mongoose');
2
3   module.exports = () => {
4       try {
5           mongoose.connect(process.env.MONGO_URI, {
6               useNewUrlParser: true,
7               useUnifiedTopology: true
8           });
9           console.log('Connected to database...')
10      } catch (error) {
11          console.error(`Cannot connect. ${error}`);
12      }
13  };
```

Рисунок 2.19 – Вихідний код mongo.js

У головному файлі проекту на боці серверу, оголошується змінна створеної реалізації підключення до бази даних, та викликається (рис. 2.20).

server > JS index.js > ...

```
1   const mongo = require('./config/mongo');
2   mongo();
```

Рисунок 2.20 – Частина вихідного коду index.js

Створюється модель за допомогою інтерфейсу Schema для таблиці з завданнями (рис. 2.21). Вказуються поля та типи даних. Схема компілюється в остаточній моделі методом mongoose.model(). Саме за назвою вказаною у цьому методу, можна буде шукати, створювати, оновлювати та видаляти об’єкти.

server > models > JS group.js > ...

```
1   const mongoose = require('mongoose');
2   const ObjectId = require("mongoose").Types.ObjectId;
3
4   const groupsSchema = new mongoose.Schema({
5     userId: mongoose.Schema.ObjectId,
6     group: [
7       {
8         groupId: String,
9         groupTitle: String,
10      },
11    ],
12  });
13
14  const Group = mongoose.model("Group", groupsSchema);
15
16  module.exports = Group;
```

Рисунок 2.21 – Вихідний код group.js

Приклад реалізації HTTP методу get(), для отримання даних з бази даних та використання створеної таблиці для груп. Оголошуються змінні, що були вказані першим аргументом при створенні бази даних, реалізація пошуку об’єктів запиту за допомогою функції findOne() бібліотеки mongoose, в параметрах вказуються будь-які дані, за якими буде відбуватися пошук (рис. 2.22).

server > controllers > JS group.js > ...

```
const Group = require("../models/group");
const User = require("../models/user");

exports.createGroup = async (req, res) => {
    const { authorization } = req.headers;
    const username = authorization;
    const groupsItems = req.body;

    const user = await User.findOne({ username }).exec();
    if (!user) {
        res.status(403)
            .json({
                message: "invalid access",
            });
        return;
    }

    const groups = await Group.findOne({ userId: user._id }).exec();
    if (!groups) {
        await Group.create({
            userId: user._id,
            group: groupsItems,
        });
    } else {
        groups.group = groupsItems;
        await groups.save();
    }

    res.json(groupsItems);
};
```

Рисунок 2.22 – Частина вихідного коду створення функції get-запиту

2.3.3 Реалізація клієнтської сторони

Створення починається з ініціалізації стандартного шаблону для створення React додатків, за допомогою команди:

npx create-react-app client

Це дозволяє використати створений автоматично файл залежностей та мінімально створений інтерфейс сайту. Використання бібліотеки react-bootrstrap допомагає у розробці з налаштованими стандартними

компонентами та стилями до них. Приклад застосування наведено на рисунку 2.23, де можна побачити, що пропоновані компоненти Form, Button, Form.Text потребують лише виклику та надання атрибутів для повного використання та надання стилю.

client > src > components > Task > JS Task.js > ...

```
import React from "react";
import { Button, Form } from "react-bootstrap";

export default function Task({ currentTask }) {
    return (
        <Form className="ms-4">
            <Form.Text><b>Title: </b>{currentTask.text}</Form.Text>
            <Form.Text><b>Description: </b>{currentTask.description}</Form.Text>
            <Form.Text><b>Group: </b>{currentTask.group}</Form.Text>
            <Form.Text><b>Due to: </b> {currentTask.dueDate}</Form.Text>
            <Form.Text><b>Priority: </b>{currentTask.priority}</Form.Text>
            <Button
                size="sm"
                className="me-2"
                variant="primary">
                Edit
            </Button>
            <Button
                size="sm"
                variant="outline-danger"
                id="btnGroupAddon2">
                Delete
            </Button>
        </Form >
    )
}
```

Рисунок 2.23 Вихідний код Task.js

Складові частини структури клієнтської сторони зображено на рисунку 2.24.

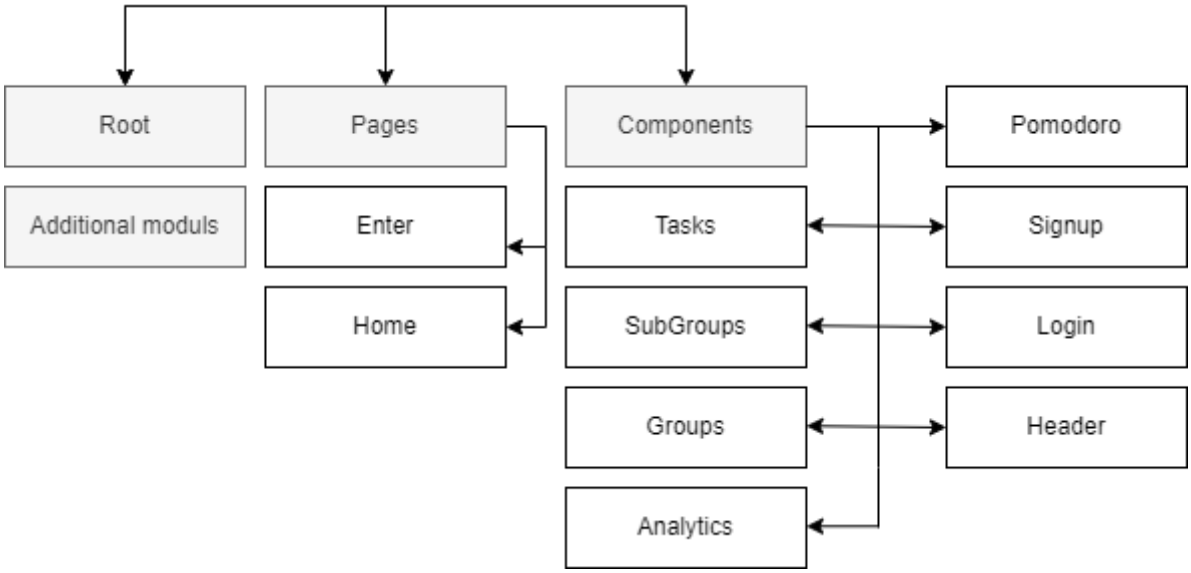

Рисунок 2.24 – Структури клієнтської папки

2.3.4 Структура створеного проекту

На кінці налаштування маршрутів, створення бази даних та реалізацій функцій зворотного виклику, виходить такий вигляд головного стартового файлу (рис. 2.25).

server > JS index.js > ...

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');
const user = require("./routes/user");
const group = require("./routes/group");
const subgroup = require("./routes/subgroup.js");
const task = require("./routes/task");
const port = process.env.PORT || 8000;
const app = express();
const mongo = require('./config/mongo');

mongo();
app.use(cors());
app.use(express.json());
app.use(bodyParser.json());
app.use("", user);
app.use("", group);
app.use("", subgroup);
app.use("", task);

app.listen(port, () => {
    console.log(`Listening to the http://localhost:${port}/ port...`);
});
```

Рисунок 2.25 – Вихідний код головного файлу сервера

Пакетна діаграма, різновид структурної діаграми, показує розташування та організацію елементів моделі в середньому та великому проекті. Діаграма пакетів може показувати як структуру, так і залежності між підсистемами або модулями, показуючи різні уявлення про систему, наприклад, як багаторівневе (він же багаторівневе) додаток - багаторівнева модель програми.

Даний проект має наступну пакетну діаграму зображену на рисунку 2.26.

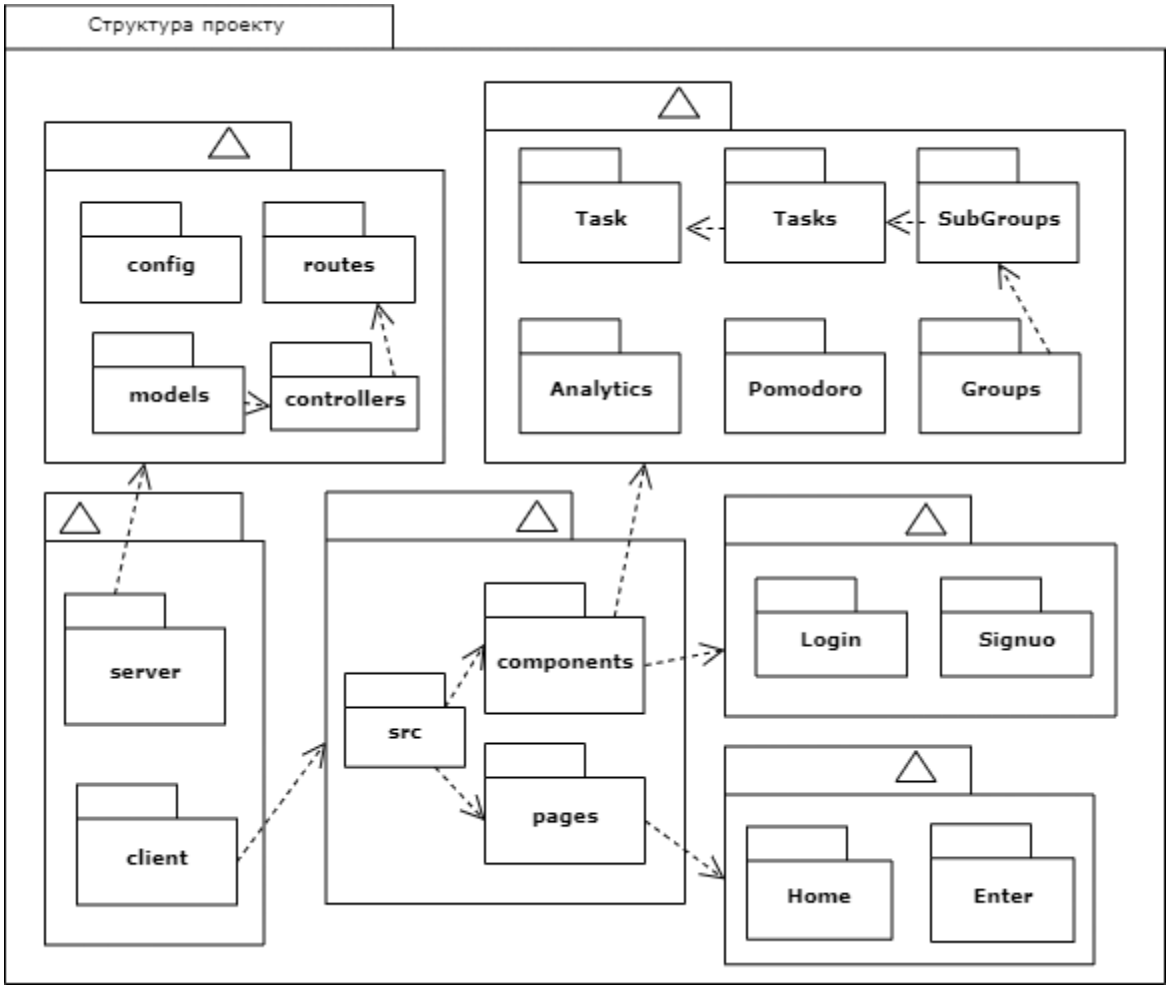

Рисунок 2.26 – Діаграма пакетів

2.4 Результат розробки

В даній роботі кожний бік програми був запушений на таких портах:

- 3000 – клієнтська частина;
- 5000 чи 8000 – серверна.

На прикладі створеної таблиці Group, у базі даних MongoDB записи мають такий вигляд (рис. 2.27).

Рисунок 2.27 – Вигляд таблиці груп

Інтерфейс додатку (рис. 2.28, 2.29, 2.30, 2.31). Завдяки використанню бібліотеки React та запровадження односторінкового методу створення, користувач не витрачає час на загрузку сторінок, завдяки компонентному підході бібліотеки. Додаток має мінімалістичний дизайн та гучний UI/UX, відсутні складні моменти, має добрий акцент для компонентів, з якими працює користувач. Це підвищує користувацький досвід з роботою сайту і зручність його використання.

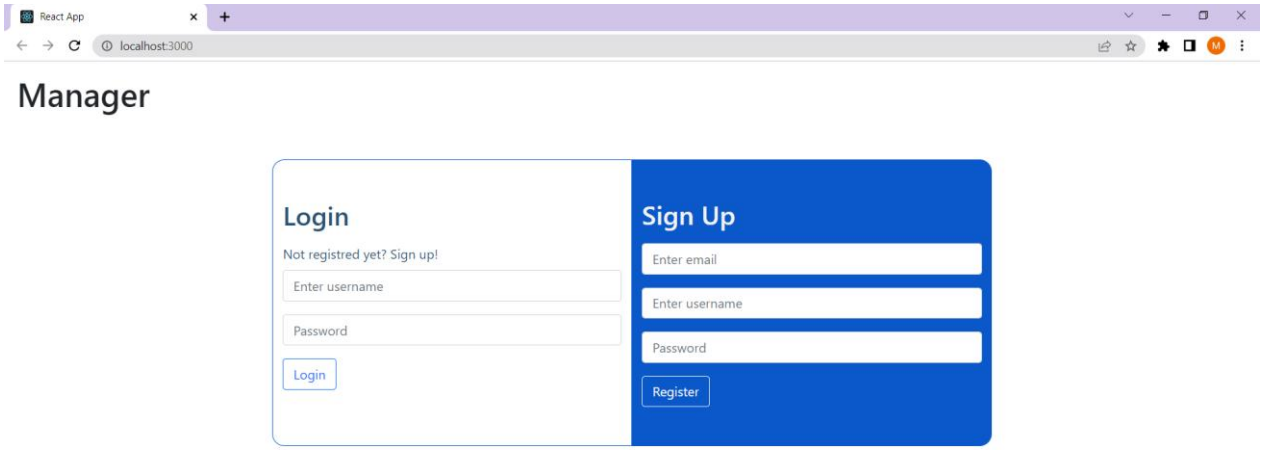

Рисунок 2.28 – Вигляд вхідної сторінки

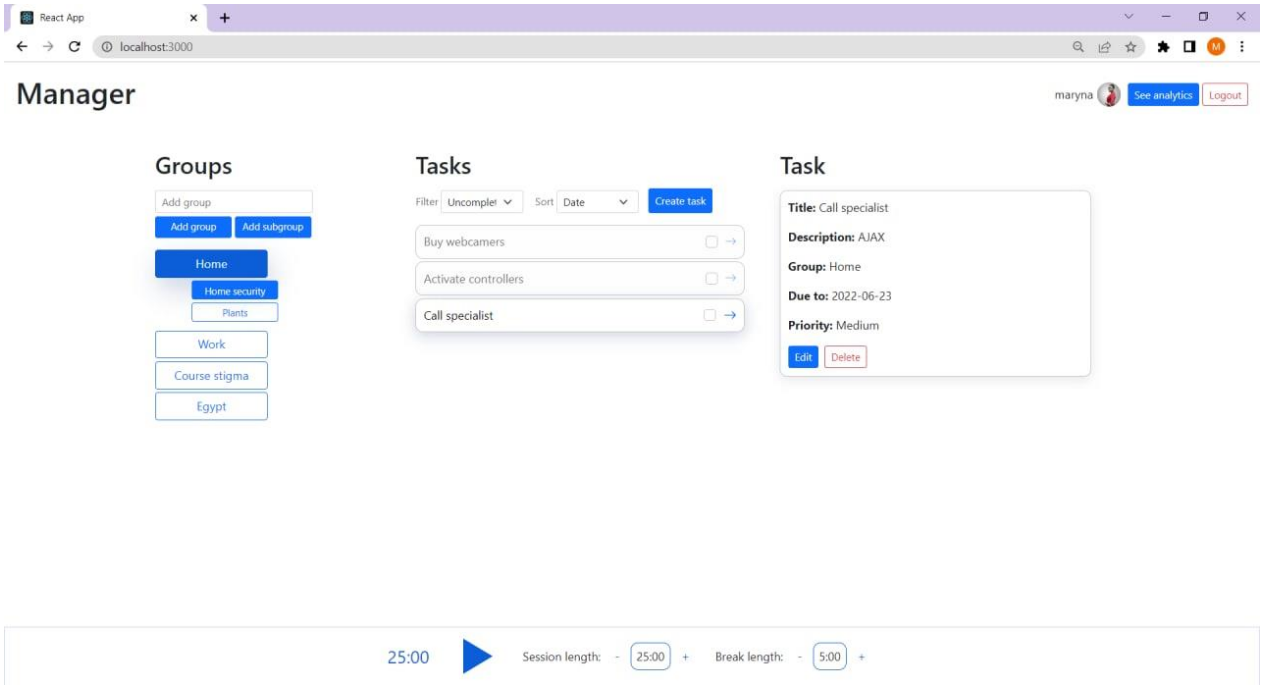

Рисунок 2.29 – Вигляд головної сторінки

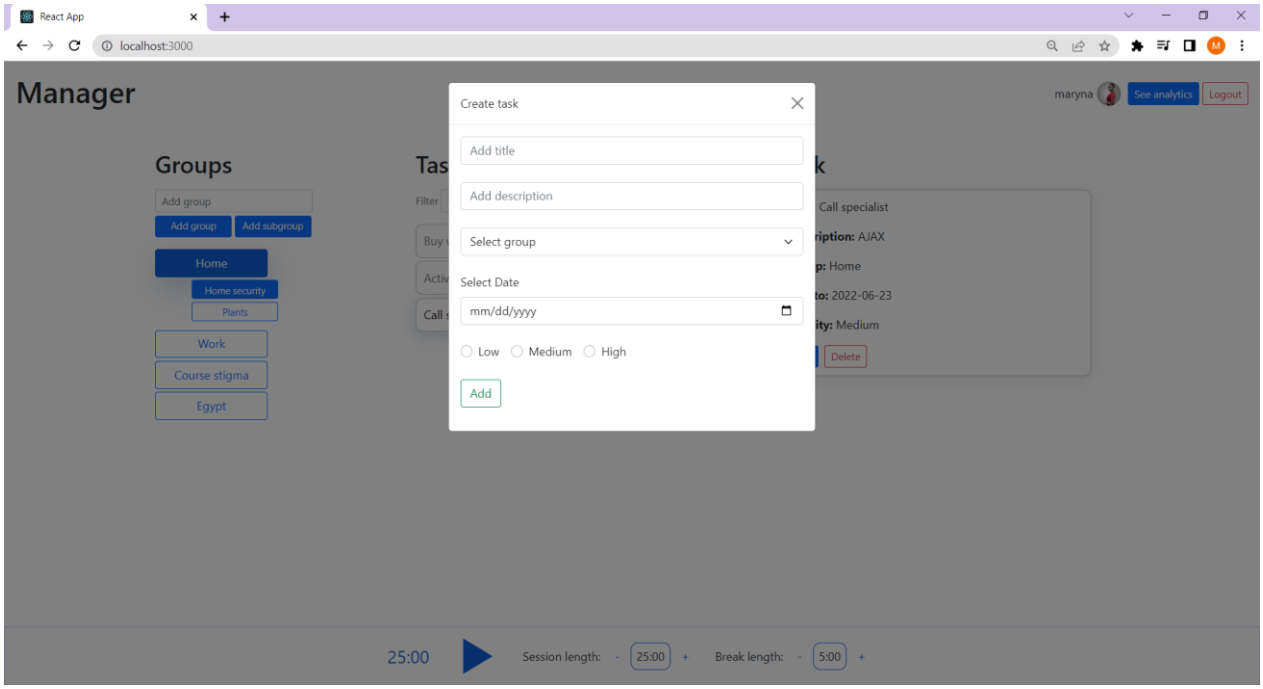

Рисунок 2.30 – Вигляд створення завдання

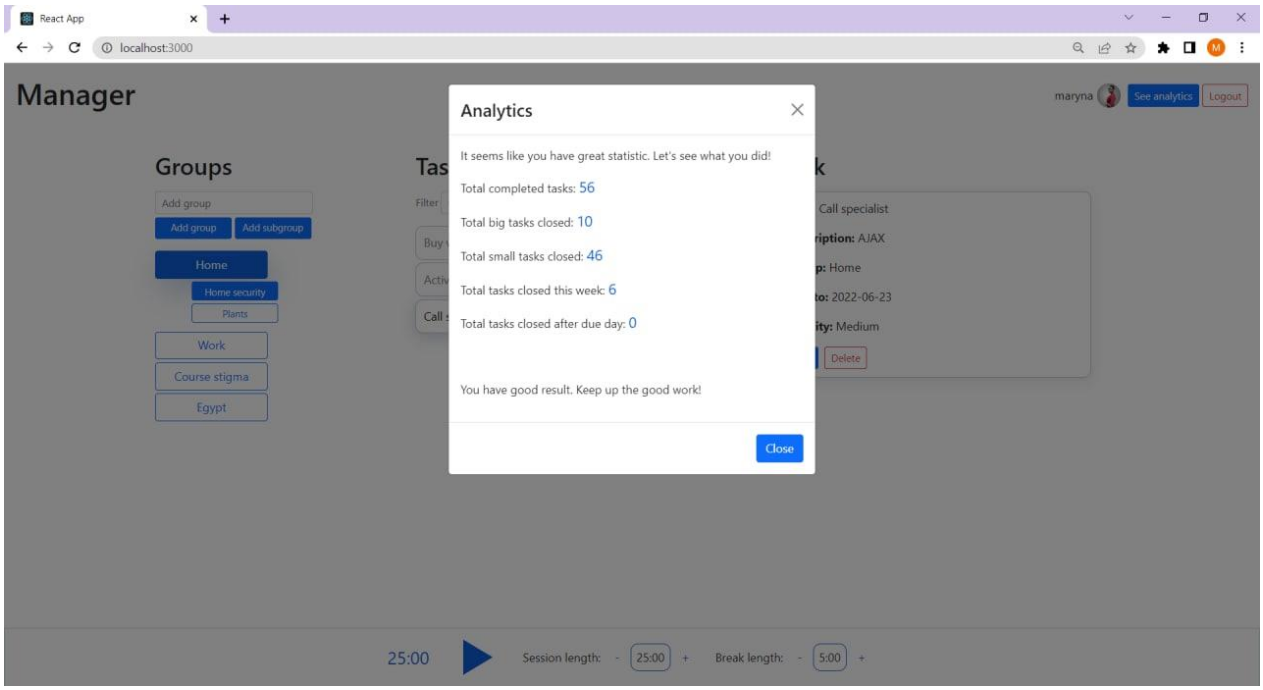

Рисунок 2.31 – Вигляд модального вікна аналізу

2.5 Висновки по другого розділу

У другому розділі обґрунтовано платформу та інструменти для розробки інформаційної системи, де React використовується для розробки інтерфейсу веб-додатку, Node.js та фреймворк Express.js для сервера додатку, де будуть реалізовані створення та запит API для комікування з документо-орієнтованою системою керування базою даних MongoDB за допомогою модулю mongoose. Проведено моделювання інформаційної системи за допомогою уніфікованою мовою моделювання UML та її діаграми прецедентів, розгортання, класів, пакетів, компонентів, які спрощують створення логіки додатку та відображають структуру додатку. Також проведено проектування системи за допомогою створення прототипу в векторному онлайн сервісі Figma. Створено веб-додаток з базовими методами тайм менеджменту.

ВИСНОВКИ

Робота присвячена питанню тайм менеджменту та розробці веб-додатка. В першому розділі детально розглянуто стан питання тайм менеджменту, техніка розподілу і ставлення пріоритету завданням, базовані методи контролю часу та створення списків задач. Розглянуто стан сучасних аналогів, який показав, що існує велика кількість подібних робіт, але завжди виникає питання щодо платформи, ціни, надлишку чи відсутність деяких функцій. Сформовано постановку завдання дослідження тайм менеджменту та створення зручної інформаційної системи тайм менеджменту для запровадження та надання практичних звичок з керування часу базуючись на основних методах тайм менеджменту.

Другий розділ присвячено повній розробці інформаційної системи. Зроблено огляд та підстава для розробки проекту на веб платформі. Моделювання і проектування інформаційної системи за допомогою діаграм UML. Огляд та підстава розробки додатка 3-рівненої архітектури на основі стеку технологій MERN, де React.js використано для розробки інтерфейсу додатка, Node.js та фреймворк Express.js для розробки серверної сторони і MongoDB для створення та контролювання базою даних.

Як результат роботи отримано зручний розроблений веб-додаток з базовими методами тайм менеджменту такими, як таймер Pomodoro, можливість розбивати завдання на групи та підгрупи та пріоритет задач, з мінімалістичним зовнішнім виглядом та вірним UI/UX задля звеличення впровадження тайм менеджменту, щоб нічого не відволікало, але інтуїтивно зрозуміло.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Thoughts on Moral Letters to Lucilius: Letter 1. [Електронний ресурс] – Режим доступу до ресурсу: https://0xedward.io/thoughts-on-moral-letters-to-lucilius-letter-1-on-saving-time/
2. Іван Примаченко про онлайн-навчання на карантині. [Електронний ресурс] – Режим доступу до ресурсу: https://www.radiosvoboda.org/a/30582636.html
3. Social eLearning: A Strategy for Increasing Online Course Completion Rates. [Електронний ресурс] – Режим доступу до ресурсу: https://www.novoed.com/resources/blog/elearning-social-completion-rates/#:~:text=Completion%20rates%20in%20online%20courses,low%2C%20averaging%20around%2013%25.
4. Маліновська О.Я. Конспект лекцій з навчальної дисципліни «Особистий, корпоративний та публічний тайм-менеджмент» / Маліновська О.Я. – Львів: ЛНУ, 2018. – 112 с.
5. Stella Cottrell. The Study Skills Handbook / Stella Cottrell. – Palgrave Macmillan, 2013. – 123 с.
6. Time management. [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Time_management
7. Тайм-менеджмент (Управление временем). [Електронний ресурс] – Режим доступу до ресурсу: https://training.pwc.ru/blog/time-management/
8. Менеджмент: Навчальний посібник / Н.С. Краснокутська, О.М. Нащекіна, О.В. Замула та ін. – Харків : «Друкарня Мадрид», 2019. – 231 с.
9. ABC Method Time Management: What It Is & How to Implement.. [Електронний ресурс] – Режим доступу до ресурсу: https://biz30.timedoctor.com/abc-method-time-management/#2-alpen-method

10. Pomodoro Technique Considered Harmful [Електронний ресурс] – Режим доступу до ресурсу: https://arialdomartini.wordpress.com/2014/05/19/pomodoro-technique-considered-harmful-dont-worry-you-are-not-using-it/
11. Habitica [Електронний ресурс] – Режим доступу до ресурсу: https://habitica.com/
12. Tweek [Електронний ресурс] – Режим доступу до ресурсу: https://tweek.so/
13. MyLifeOrganized [Електронний ресурс] – Режим доступу до ресурсу: https://www.mylifeorganized.net/
14. Clear [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/us/app/clear-todos/id493136154
15. Desktop vs Mobile vs Tablet Market Share Worldwide [Електронний ресурс] – Режим доступу до ресурсу: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet
16. Business Process Model and Notation [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation
17. Unified Modeling Language [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Unified_Modeling_Language
18. Документація JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript

ДОДАТОК А

Відомість матеріалів кваліфікаційної роботи

№ з/п	Позначення	Найменування	Кільк. аркушів	Примітки
1				
2		Документація		
3				
4	**ІТКІ.КР.22.05.ДА.ПЗ**	Пояснювальна записка		Формат А4
5				
6		Презентація		
7				
8		Диск CD-R з презентацією	1	Диск CD-R
9				

Зм.	Ар-куш	№ докум.	Підпис	Дата	**ІТКІ.КР.22.05.ДА.ПЗ.**

Розроб.	Петрига М.В.			**Матеріали кваліфікаційної роботи**	Літ.		Аркуш	Аркушів
Керівник	Соколова Н.О.				Н		1	1
Рецензент	Реута О.В.				НТУ «ДП», 12; 126-18-1			
Н.контр.	Коротенко Г.М.							
Зав. каф.	Гнатушенко В.В.							