**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ**

# КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розроблення веб-додатку для організації робочого плану студента за допомогою React.js»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
освітньо-професійної програми Штучний інтелект

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

_____ Євген ПІОНТКІВСЬКИЙ

Виконав: здобувач вищої освіти гр.ШІД-41
Євген ПІОНТКІВСЬКИЙ

Керівник: Максим ФЕСЕНКО
к.т.н., доцент

Рецензент:
_____

**Київ 2024**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

**Навчально-науковий інститут інформаційних технологій**

Кафедра Штучного інтелекту
Ступінь вищої освіти Бакалавр
Спеціальність 122 Комп'ютерні науки
Освітньо-професійна програма Штучний інтелект

**ЗАТВЕРДЖУЮ**
Завідувач кафедрою Штучного інтелекту

_____ Ольга ЗІНЧЕНКО
«_____» _____ 2024 р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Піонтківському Євгену Романовичу

1. Тема кваліфікаційної роботи: Розроблення веб-додатку для організації робочого плану студента за допомогою React.js

керівник кваліфікаційної роботи Максим ФЕСЕНКО к.т.н., доцент,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» 02.2024р. № 36

2. Строк подання кваліфікаційної роботи «31» травня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, статистичні дані, вимоги документації веб-технологій.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Постановка мети, об'єкту та задач дослідження
Аналіз актуальної проблеми
Опис веб-технологій
5. Перелік графічного матеріалу: *презентація*
1. Аналіз веб-технологій
2. Аналіз поточної ситуації в області проекту
3. Архітектура веб-додатку

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

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз сучасних веб-додатків/додатків конкурентів, та сучасних технологій розробки | 27.02-05.03.24 | Виконано |
| 2 | Дослідження основних технологій веб-розробки | 05.03-15.03.24 | Виконано |
| 3 | Аналіз методології, принципів та методів веб-розробки | 16.03-25.03.24 | Виконано |
| 4 | Розробка веб-застосунку | 25.03-30.04.24 | Виконано |
| 5 | Оформлення роботи: вступ, висновки, реферат | 01.05-04.05.24 | Виконано |
| 6 | Написання кваліфікаційної роботи | 04.05-20.05.24 | Виконано |
| 7 | Завершення та підготовка кваліфікаційної роботи | 21.05-27.05.24 | Виконано |

Здобувач вищої освіти _____ Євген ПІОНТКІВСЬКИЙ

Керівник
кваліфікаційної роботи _____ Максим ФЕСЕНКО

# РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 49 стор., 1 табл., 31 рис., 15 джерел.

*Мета роботи* – підвищення ефективності роботи студентів та організації часу.

*Об'єкт дослідження* – процес розробки веб-додатку.

*Предмет дослідження* – система веб-додатку.

*Короткий зміст роботи*: У роботі проведено аналіз сучасних веб-технологій, їх різновидів, додаткових бібліотек тощо. Також було проведено дослідження в сфері проблематики проекту – вплив стресу на студентів та відсутність продуктивності. Було ретельно досліджено чинники, що впливають на студентів та методи підвищення продуктивності студентів.

За результатами проведеного аналізу та досліджень було розроблено прототип веб-застосунку з урахуванням UI/UX та розроблено веб-додаток за поданим макетом, використовуючи фреймворк React.js, HTML, CSS, JavaScript та бібліотеку Redux.

По завершенню роботи отримано додаток для організації робочого плану студента, що допоможе студентам раціонально розподілити свій час на будь-які задачі.

КЛЮЧОВІ СЛОВА: FRONTEND, ВЕБ-ТЕХНОЛОГІЇ, СТУДЕНТИ, ОРГАНІЗАЦІЯ ПЛАНІВ, ВЕБ-ЗАСТОСУНОК, СТРЕС, ПРОДУКТИВНІСТЬ.

**ЗМІСТ**

# ВСТУП

В сучасному світі розробка веб-застосунків є одним з актуальних напрямків IT-технологіях. З появою веб-технологій світ зробив великий прорив. Не аби яке значення має зараз веб-додатки в усіх сферах бізнесу та бізнес-процесів. З допомогою frontend-розробки створюються рішення, які покращують комунікацію між користувачами та підприємством, спрощує інформування споживачів, розширює клієнтську базу бізнесу та підвищує його дохід.

Наразі існує багато рішень у веб-технологіях, які з легкістю можуть створити якісний веб-додаток для підприємства, landing-page (картку-візитівку) чи веб-сайт для надання послуг.

Зараз для студентів як ніколи актуальним рішенням є застосунки для організації робочого плану та часу. Картина сучасного студента виглядає наступний чином:

– добре навчається;
– виконує всі роботи по навчанню;
– знайшов себе та знає чого хоче;
– самостійно опановує спеціалізацію, в якій хоче розвиватись;
– працює;
– веде активний спосіб життя;
– продуктивний.

Через культ продуктивності студенти намагаються відповідати певним нормам успішної людини, через що знаходяться в стані стресу, вигорання або навіть депресії. І це не дивно, адже у молодих людей набагато більші вимоги та очікування до себе та життя.

Більшості з студентів не вистачає організації часу та планів, які потрібно виконувати для досягнення певних цілей. Тож, вкрай корисним рішенням в цьому випадку буде веб-додаток для організації планів, задач та часу.

В рамках кваліфікаційної роботи було прийнято розробити веб-додаток, який можна інтегрувати, як в систему moodle, так і реалізувати самойстійним веб-

застосунком. Цей додаток містить в собі інструменти, які доможуть зосередитись студентам на актуальних задачах та планах, розробити реальний план на день відносно індивідуальних можливостей кожного та створювати нагадування аби нічого не забути та не пропустити дедлайн.

За проведеним аналізом, встановлено, що таке сучасне веб-рішення підвищить продуктивність студентів, знизить рівень стресу та допоможе дисциплінувати кожну сферу власного життя.

Мета проекту полягає створенні актуального веб-застосунку за допомогою сучасних технологій, який покликано покращити емоційний стан студентів та налаштувати робочий план.

В рамках кваліфікаційної роботи було поставлено наступні задачі:

1. Проаналізувати ринок схожих інструментів.
2. Провести аналіз актуальних веб-технологій, що стануть найкращим рішенням в реалізації веб-застосунку.
3. Встановити картину актуальних потреб та емоційного стану студентів
4. На основі проведеного аналізу розробити прототип веб-додатку
5. За створеним прототипом розробити веб-застосунок з урахування всіх потреб та актуальних рішень UI/UX принципів
6. Провести тестування аналізу
7. Задокументувати проведену роботу.

# 1 АНАЛІЗ АКТУАЛЬНИХ ВЕБ-ТЕХНОЛОГІЙ ТА АНАЛОГІВ-ДОДАТКІВ

В сучасному світі веб-додатки для організації планів та часу студентів стають все більш актуальними. Вони допомагають студентам ефективно керувати своїм часом, завданнями та навчальним процесом. Цей розділ буде присвячений аналізу актуальних веб-технологій та огляду аналогів конкурентів, що допоможе визначити найкращі практики та інновації для розробки нашого проекту за допомогою React.js, Redux, HTML, JavaScript та CSS.

## 1.1 Аналіз аналогів веб-додатків та додатків

Наразі існує багато додатків та веб-додатків за різним призначенням та функціоналом. Вони допомагають користувачам автоматизувати більшість процесів пов’язаних з продуктивним розподілом часу на всі справи. Також, такі додатки систематизують досягнення та певні задачі, які користувач виконує щодня. [12]

Зазвичай такі додатки ще включають в себе певний функціонал з нагадуваннями, відстеженням прогресу, додатковими вправами для підвищення продуктивності тощо.

### 1.1.1 Any.do

Один з популярних додатків, розроблений для організації особистого та професійного життя. Даний додаток включає в себе наступні функції:

- створення списків задач та нагадувань;
- планування задач за допомогою календаря (особливою перевагою є інтеграція з іншими додатками);
- фільтрація та сортування задач;
- підтримка командних налаштувань.

Як зазначається в джерелах, інтерфейс доволі простий та інтуїтивно зрозумілий, що є в більшості випадків перевагою, адже такий підхід не буде обтяжувати користувачів та перенавантажувати їх сприйняття. Але після проведеного аналізу все ж таки хочеться спростувати ці твердження. Бо після

проведеного тестування додатку, були зроблені висновки, що інтерфейс є доволі складним для сприйняття та занадто перенавантажує сприйняття (рис. 1.1).

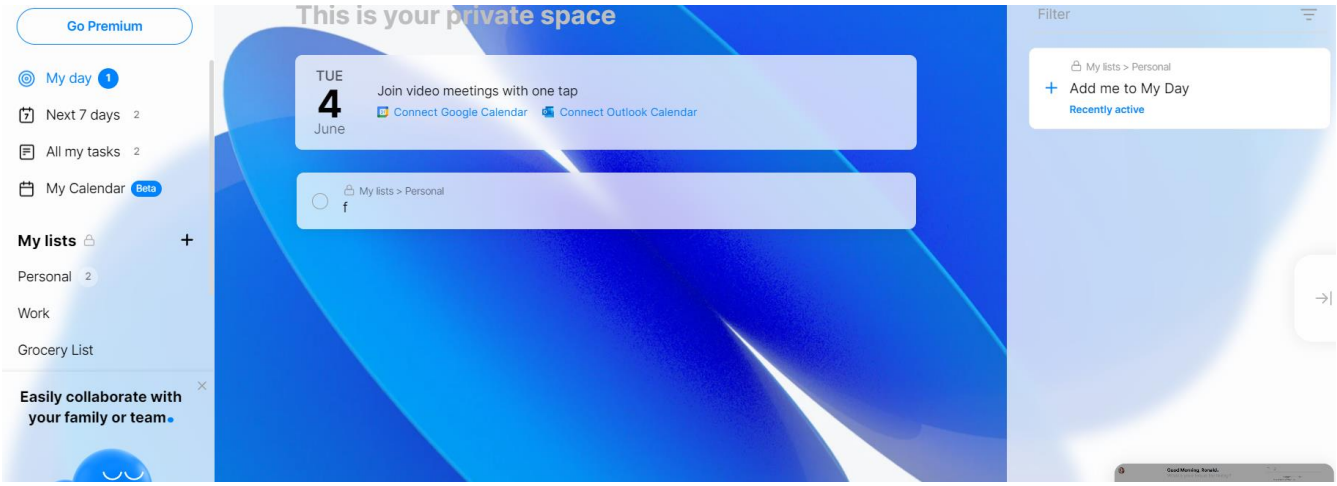



Рис. 1.1 – Інтерфейс головної сторінки Any.do

І хоча на перший погляд все виглядає доволі простим та знадобиться деякий час аби звикнути до такого інтерфейсу. На рис. 1.2 представлено сторінку з усіма задачами користувача. І поки в списку немає задач, то виглядає все більш менш зрозуміло, але варто додати декілька задач і простота сприйняття змінюється.

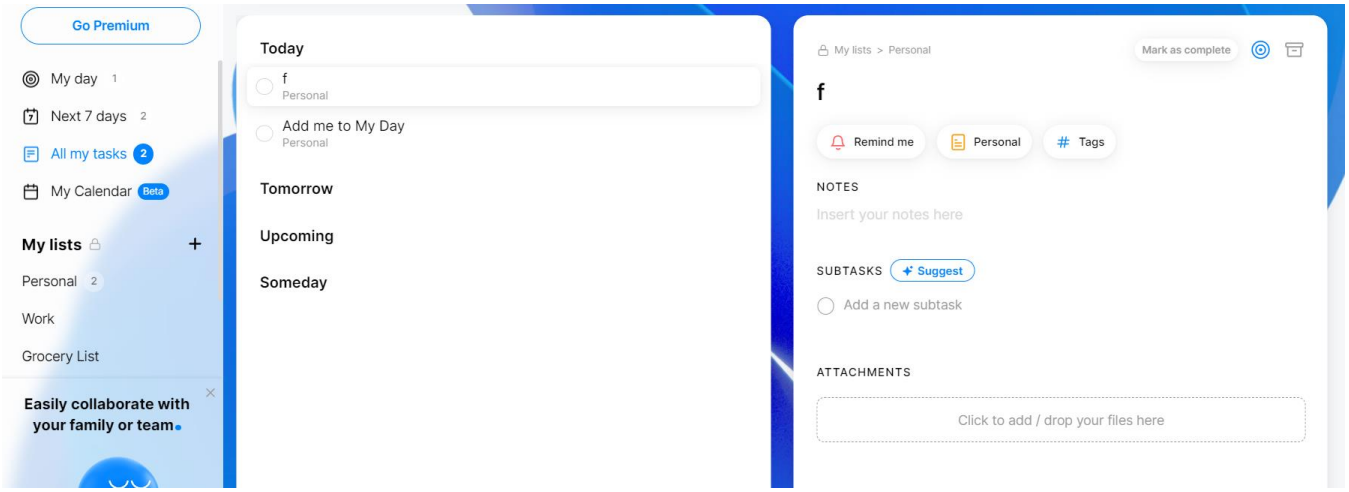



Рис. 1.2 – Сторінка з задачами Any.do

Щодо недоліків, то в джерелах відзначається про обмеження на велику кількість корисних функцій у безкоштовній версії.

### 1.1.2 Todoist

Цей веб-застосунок був створений аж у 2007 році. Притягнув він до себе увагу простим інтерфейсом (рис. 1.3), але корисним функціоналом. Додаток доступний на різних платформах та у різних варіаціях: Android, iOS, Windows, macOS.

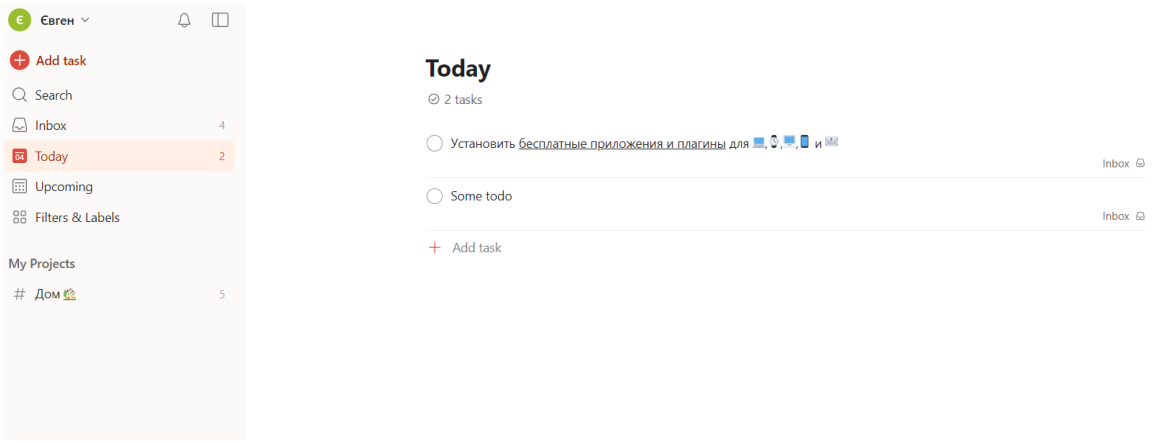



Рис. 1.3 – Інтерфейс веб-застосунку Todoist

Основні функції додатка включають в себе:

– створення завдань з обширним описом та вибором різних критеріїв (рис. 1.4);
– можливість створення великих проектів та поділяти їх на підпроекти;
– налаштування як для особистої роботи, так і для командної;
– налаштування нагадувань;
– можливість організації плану на одній дошці.

Додатки для організації плану повинні містити в собі велику кількість функціоналу, але залишатись лаконічними водночас. Адже користувачі хочуть спростити собі організацію, тож чим менше в додатку будуть відволікаючих факторів, тим ефективнішим він буде.

З приводу недоліків Todoist має дещо застарілий функціонал (в деяких випадках) та обмеження в безкоштовній версії. А більшість нестандартних корисних інструментів в додатку надаються лише після придбання преміум-версії.

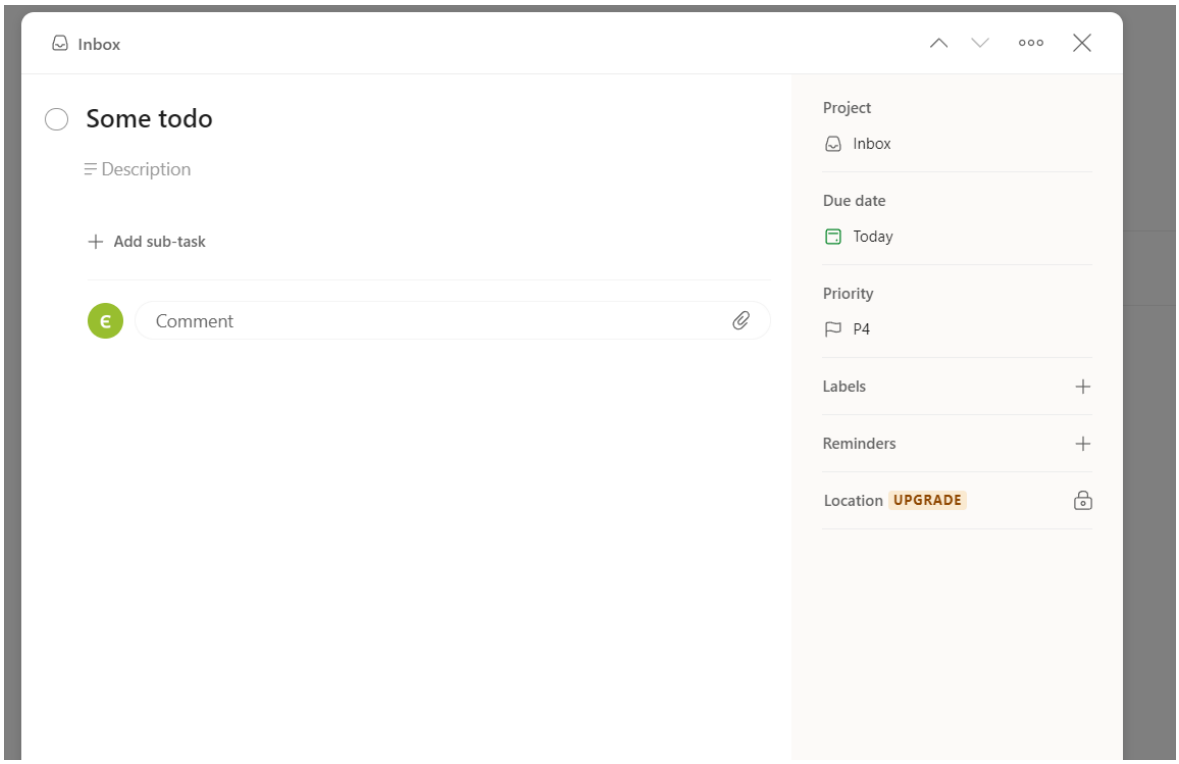



Рис. 1.4 – Вікно для створення задач в Todoist

## 1.2 Аналіз HTML

HTML (HyperText Markup Language) – це стандартна мова розмітки, яка використовується для створення веб-сторінок та веб-додатків. Вона є основою і разом з CSS та JavaScript складає основу сучасної веб-розробки. HTML дозволяє створювати елементи на сторінці і по завершенню вони будуть відображати контент на веб-сторінці.

Історія створення HTML починається з 1989 року, коли Тім Бернерс-Лі, працюючи у Європейській організації ядерних досліджень (CERN), запропонував проект гіпертекстової інформаційної системи на основі гіпертексту для полегшення обміну інформацією між науковцями. вченими. У 1990 році він розробив перший веб-браузер і перший веб-сервер, а також створив початкову оригінальну специфікацію HTML. Перша офіційна версія HTML була випущена в 1993 році. Мова продовжує розвиватися, набуваючи покращення функцій.

Принцип дії HTML полягає в використанні допомогою тегів для структурування та форматування вмісту веб-сторінок. Теги HTML зазвичай містять

текст або інші елементи у відкриваючих та закриваючих тегах, наприклад <p> для абзацу або <h1> для заголовків першого рівня. Браузер інтерпретує ці теги та відображає вміст згідно з їхніх властивостей.

Документ HTML починається з декларації <!DOCTYPE html>, яка вказує браузеру, що це документ HTML5. Наступним йде кореневий елемент <html>, який містить два основні розділи: <head> та <body>. Розділ <head> містить метадані про документ, такі як заголовок, посилання на CSS-стилі, мета-теги та скрипти. Розділ <body> містить фактичний вміст веб-сторінки, включаючи тексти, зображення, таблиці, форми та інші елементи.

HTML5, офіційно випущений у 2014 році, додав багато нових можливостей та семантичних елементів, таких як <header>, <footer>, <article>, <section> та <nav>, які покращують структурування документа та роблять його більш зрозумілим як для розробників, так і для пошукових систем.

Ключовою особливістю HTML є його гнучкість та сумісність. HTML-документи можуть бути переглянуті на будь-якому пристрої з веб-браузером, що забезпечує кросплатформенність та доступність. Більше того, HTML є відкритим стандартом, який підтримується консорціумом World Wide Web (W3C) та Web Hypertext Application Technology Working Group (WHATWG), що гарантує його постійний розвиток і вдосконалення.

Документ HTML починається з оголошення <!DOCTYPE html>, яке повідомляє браузеру, що це документ HTML5. Далі йде кореневий елемент <html>, який містить дві основні частини: <head> і <body>. Розділ <head> містить метадані про документ, такі як заголовок, посилання на стилі CSS, мета-теги та сценарії. Розділ <body> містить фактичний вміст веб-сторінки, включаючи текст, зображення, таблиці, форми та інші елементи.

HTML5 був офіційно випущений у 2014 році, додавши багато нових функцій і семантичних елементів, таких як <header>, <footer>, <article>, <section> і <nav>, покращивши структуру документа та спростивши його для розробників розуміти і пошукові системи. HTML5 також підтримує вбудовані мультимедійні елементи,

такі як <audio> і <video>, що дозволяє відтворювати аудіо та відео без використання додаткових модулів.

HTML тісно інтегрований із CSS (каскадними таблицями стилів) і JavaScript, що дозволяє створювати сучасні, динамічні та стильні веб-сторінки. CSS використовується для дизайну та зовнішнього вигляду веб-сторінок, дозволяючи стилізувати такі елементи HTML, як колір, шрифти, відступи, позиціонування та анімацію. JavaScript, у свою чергу, додає можливість створювати інтерактивний та динамічний вміст, такий як форми з перевіркою даних, анімованими меню, обробкою подій тощо.

Важливим аспектом використання HTML є забезпечення доступу до веб-сторінок для всіх користувачів, у тому числі для людей з обмеженими можливостями. Для цього розробники використовують семантичні теги, альтернативний текст зображення, атрибути символів та інші методи, які допомагають програмам зчитування з екрана та іншим допоміжним технологіям правильно інтерпретувати вміст.

Таким чином, HTML є основою сучасної веб-технології, забезпечуючи структуру та основу для створення веб-сторінок і програм. Гнучкість, сумісність і постійний розвиток HTML роблять його незамінним інструментом для розробників для створення багатих, доступних і привабливих веб-ресурсів.

### 1.3 Аналіз CSS

CSS (Cascading Style Sheets) – це мова стилізації, що використовується для опису стилів та форматування веб-сторінок, що написані мовою розмітки HTML або XML. CSS дозволяє розділити структуру та наповнення веб-документу від його оформлення, що робить їх більш організованими, гнучкими, легшими для розширення та обслуговування. З моменту свого створення CSS дуже розширився і став не додатковим, а одним із основних компонентів веб-розробки.

CSS вперше був запропонований в середині 90-х років минулого століття Хаконом Віум Лі, щоб зробити код веб-сторінок менш складним та більш структурованим, яким він був на той час.

У 1996 році W3C (World Wide Web Consortium) прийняли перший офіційний стандарт CSS1. Він заклав основу для подальшого розвитку CSS, включаючи базові властивості для управління шрифтами, кольорами, відступами та вирівнюванням тексту.[8]

У 1998 році був прийнятий новий стандарт CSS2, який додав більше можливостей, наприклад, підтримку позиціонування, z-індекси, медіа-типи і стилі для різних пристроїв. У 2011 році було створено стандарт CSS2.1, який був допрацьованою версією CSS2. У ньому було виправлено численні недоліки, а також було уточнено специфікації.

Найбільший прорив відбувся з появою CSS3, який був розроблений на основі модульної структури. Це дозволило розробникам зосередитися на окремих модулях, що оптимізувало та пришвидшило їх роботу. Цей стандарт приніс багато нових можливостей і властивостей, що значно розширили можливості веб-дизайну. Були додані нові властивості для анімацій, трансформацій, переходів, а також покращено можливості для роботи з тінями, округленням кутів і градієнтами. Модулі Flexbox та Grid Layout дали розробникам потужні інструменти для створення складних макетів, які могли адаптуватися до різних розмірів екранів та пристроїв.[4]

Основними концепціями CSS є селектори, властивості та значення. Селектори дають можливість вибирати HTML-елементи для застосування стилів. Властивості визначають конкретні елементи стилізації (наприклад, колір, шрифт, розмір тощо). Значення задають конкретні параметри для властивостей. Однією з ключових особливостей CSS є каскадність. Вона дозволяє визначати порядок застосування стилів у випадку конфліктів між різними правилами. Пріоритет стилю визначається видом селектора, а також правилом «останнього оголошення».

CSS також дозволяє використовувати медіа-запити, які дають можливість створювати адаптивні веб-застосунки, що змінюють свій вигляд залежно від типу або розміру пристрою. Це стало дуже важливим в епоху мобільного інтернету, коли веб-додатки повинні виглядати однаково добре на різних пристроях – від екранів маленьких смартфонів до великих моніторів.

У наш час практики веб-розробки часто включають в себе використання препроцесорів CSS, таких як SASS (Syntactically Awesome Stylesheets) і LESS (Leaner Style Sheets). Вони дозволяють використовувати вкладені правила, змінні, міксини та функції, які роблять CSS-код більш організованим і легшим для підтримки. Препроцесори компілюють розширений CSS-код у звичайний CSS, який може зрозуміти браузер.

Також важливим аспектом є використання інструментів для побудови та оптимізації, таких як CSS Nano та PostCSS. Вони дають можливість автоматично додавати вендорні префікси для крос-браузерної сумісності, мінімізувати CSS-код для зменшення розміру файлів, а також виконувати інші оптимізації, які покращують продуктивність веб-застосунків.

Отже, CSS є невід'ємною або навіть однією з основних частин сучасної веб-розробки. Він забезпечує розробників потужними інструментами для створення візуально привабливих і функціональних веб-сторінок. Завдяки постійному розвитку і впровадженню нового функціоналу, CSS залишається актуальним і важливим компонентом для розробки інтерфейсів користувача, які відповідають сучасним вимогам і стандартам.

### 1.4 Аналіз JavaScript

JavaScript – це динамічна високорівнева мова програмування, яка стала одним із ключових інструментів при розробці сучасних веб-додатків.

Вона була створена Бренданом Айком у 1995 році для Netscape Navigator. JavaScript швидко здобула популярність і стала стандартом для програмування клієнтської сторони веб-застосунків. Основною метою цієї мови є забезпечення інтерактивності веб-сторінок, що дозволяє динамічно змінювати наповнення, реагувати на дії користувача, а також обробляти дані без необхідності завжди перезавантажувати сторінку.

JavaScript є мовою зі слабкою типізацією, вона підтримує парадигми об'єкто-орієнтованого програмування, функціонального та імперативного програмування. Це робить її універсальною та гнучкою, а також дозволяє вирішувати задачі

широкого спектру. Мова має вбудовану підтримку роботи з об’єктами, функціями вищого порядку та замиканням, що, в свою чергу, дозволяє розробникам створювати складні та багатофункціональні програми.

Дуже важливою характеристикою JavaScript є інтеграція з веб-браузерами. Усі основні браузери мають вбудовані інтерпретатори JavaScript, що дозволяє виконувати код безпосередньо у контексті веб-сторінки. Це робить JavaScript невід’ємною частиною веб-дизайну та розробки, а також дозволяє створювати інтерактивні та динамічні сервіси інтерфейси користувача. Ця мова дає можливість маніпулювати елементами HTML через DOM (Document Object Model), змінювати стилі CSS, а також обробляти події взаємодії користувача з веб-сторінкою.[3]

JavaScript підтримує асинхронне програмування через використання колбеків, промісів і асинхронних функцій. Це є надзвичайно важливим для веб-додатків, адже це дозволяє взаємодіяти з сервером або виконувати довготривалі операції без блокування користувацького інтерфейсу. Асинхронні операції дають можливість завантажувати дані, працювати з таймерами, обробляти запити до серверів і виконувати інші завдання, не блокуючи основний потік виконання коду.

З часом JavaScript розширив сферу свого використання далеко за межі браузера. З появою Node.js JavaScript став популярним і для серверної частини веб-застосунків. Node.js дозволяє запускати код JavaScript на стороні сервера, що відкриває багато можливостей для створення повноцінних серверних додатків, обробки запитів до баз даних, управління файлами і багато іншого. Node.js використовує подієву модель вводу та виводу, що робить його особливо підходящим для розробки високонавантажених мережевих додатків.[7]

JavaScript має велику кількість бібліотек і фреймворків, які значно спрощують розробку. Серед найпопулярніших можна виділити:

- React.js, створений Facebook, який дозволяє створювати складні інтерфейси користувача на основі компонентного підходу;
- Angular, розроблений Google, використовується для побудови масштабованих та підтримуваних веб-додатків;

- Vue.js – фреймворк, що поєднує переваги Angular і React, а також пропонує простоту використання та високу продуктивність.

JavaScript також підтримує модульність через стандарти ES6 (ECMAScript 2015) і новіші версії, що дозволяє розробникам розбивати код на модулі і повторно використовувати їх у різних частинах програми. Це сприяє кращій організації коду та спрощує його підтримку та тестування. Інструменти для побудови та збірки, такі як Webpack, Babel, та Parcel, також стали стандартами в JavaScript-розробці, дозволяючи розробникам компілювати сучасний JavaScript-код у форми, сумісні з усіма браузерами, і оптимізувати продуктивність додатків.

Сучасні веб-додатки часто вимагають високого рівня інтерактивності та швидкості, і JavaScript є ідеальним інструментом для досягнення цих цілей. Використання сучасних можливостей мови, таких як синтаксичний цукор для класів, деструктурування, стрілкові функції та інші нововведення, дозволяє писати чистий та ефективний код. Інтеграція з API браузера, такими як Fetch для асинхронних HTTP-запитів, WebSockets для реального часу, LocalStorage та IndexedDB для зберігання даних на стороні клієнта, розширює можливості розробників і дозволяє створювати багатофункціональні та швидкі веб-додатки.

Отже, JavaScript – це ключова мова для веб-розробки, яка постійно розвивається і адаптується до нових вимог та технологій. Завдяки свої потужності, гнучкості, а також широкій підтримці серед розробників, ця мова залишається на першій лінії інновація у створення веб-застосунків і забезпечує багаті можливості для інтерактивного та динамічного користувацького інтерфейсу.

# 2 АНАЛІЗ ТЕХНОЛОГІЙ ТА СУПУТНІХ ФАКТОРІВ

## 2.1 Аналіз чинників стресу, що впливають на життя студентів

### 2.1.1 Тенденція продуктивності

Однією з популярних проблем двадцять першого століття став хронічний стрес через перенавантаження та вигорання. Особливо ця тенденція стала помітною останні декілька років через розповсюдження культу продуктивності в соціальних мережах та Інтернеті в цілому. Особливо під вплив таких «трендів» потрапляють молоді хлопці та дівчата віком від 17 до 27 років. Через транслювання гарної картинки блогерів успіху та постійного тиску масової «продуктивності» молоді люди (особливо студенти) відчувають провину за те, що вони недостатньо продуктивні через що часто відчувають себе перенавантаженими та знаходяться в постійному стресі.

### 2.1.2 Статистика рівня стресу у студентів

За проведеним опитуванням студентів 4-ого курсу кафедри Штучний інтелект у 2024 році по отриманим результатам на рис. 2.1 близько 30% студентів відчувають постійне перенавантаження та стрес через навчальний процес. З них 16% відчувають додатковий стрес через роботу, ще 28% відсотків відчувають вигорання та перенавантаження тільки через роботу, близько 15% відсотків студентів, що не мають постійного стресу через роботу та навчання відчувають стурбованість тільки під час сесій, ще 27% студентів, що працюють та відчувають себе комфортно щодо навчання та роботи, мають рідкі прояви тривожності. В додачу, аж 36% та 12% з усіх опитаних відчувають постійний стрес через війну в країні та сімейні обставини відповідно.

За даними опитування Gradus Research 2022 року аж 70% з опитаних українців знаходяться в стані постійного стресу, особливо через фактор війни в Україні. Тобто студенти наразі піддаються стресу не тільки через фактор навчання чи роботу. [11]

### 2.1.3 Аналіз ринку IT в Україні

Через складну економічну ситуацію в країні більшість студентів стурбовані майбутнім місцем роботи, адже кількість робочих місць знизилась після лютого 2022 року, більшість IT-компаній вийшло з українського ринку, а ті що лишились скоротили свої витрати на людський ресурс. Вихід IT-компаній з українського ринку пов'язаний з тим, що менеджерам іноземних компаній не дають дозволу на відрядження через небезпечну ситуацію. В свою чергу представники наших IT-компаній (переважна більшість чоловіки) не мають змоги виїзжати закордон. Це підтверджує й відсоток забронюваних в IT-сфері – 1%.

Свій «внесок» в таку ситуацію зробила й нестабільна енергетична ситуації в країні. Більшість закордонних компаній не ризикують користуватись послугами українських IT-компаній через ризик не завершити проект своєчасно.

Також, більшість компаній не мають бажання наймати недосвідчених фахівців та займатись їх навчанням.


Рис. 2.2 Статистика відгуків та пропозицій за даними джині

На рис. 2.2 видно складну ситуацію в IT. По перше, кількість пропозицій від рекрутерів з початку 2022 року до кінця 2023 року знизилась з майже 300 тисяч до 65 тисяч. А от кількість відгуків зі 100 тисяч зросла майже до 300 тисяч, а то і більше. Якщо раніше на 300 тисяч вакансій було лише 100 тисяч відгуків, то зараз

ситуація змінилась дзеркально навпаки. З цієї статистики випливає й те, що на початку 2022 року була жорстка нестача працівників, що надавало перевагу молодим спеціалістам, адже відсутність достатньої кількості кадрів змушувала роботодавців наймати на роботу охочих та займатись їх навчанням, чого не скажеш про сьогодення.

Отож, ринок праці дійшов цієї точки, що бажаючих працювати недосвідчених фахівців багато, вакантних місць мало, ланка середньої заробітної плати знижується, а вимоги тільки зростають. А саме парадоксальне те, що спеціалістів рівня middle та senior катастрофічно не вистачає. Здається, що корінь проблеми в самому початку цього ланцюжка – відсутність бажання та відповідальності роботодавця брати на роботу молодих спеціалістів.

Проблеми з роботою являються ще однією причиною тривожності студентів. В них не залишається вибору, крім як відповідати вимогам роботодавців, а для цього потрібно наполегливо працювати. Тобто, крім навчання студентам потрібно додатково навчатись, достатньо відпочивати та вирішувати проблеми, з якими вони регулярно стикаються. І в цьому полягає проблема молодих людей двадцять першого століття – відсутність часу.

### 2.1.4 Наслідки стресу

Як відомо, постійний вплив стресу виснажує не тільки емоційне здоров’я, а й фізичне. В довгостроковій перспективі хронічний стрес призводить до постійної перевтоми та відсутності сил, що в свою чергу спричиняє вигорання та зниження мотивації.

## 2.2 Підвищення продуктивності студентів

### 2.2.1 Загальна інформація

Більшість людей не вміють правильно розпоряджатись своїм часом. Тайм-менеджмент є однією з головних складових успіху. Від правильного розпорядження часу залежить і досягання цілей, і дисципліна, і розподілення навантаження. Багато людей нехтують правилами тайм-менеджменту та беруть на себе більше справ, аніж можуть виконати. Чи не враховують правильне співвідношення між роботою та

відпочинком після чого виснуються та вигорають. Або ж просто забувають про деякі важливі завдання, що призводить до негативних наслідків.

### 2.2.2 Аналіз використання правил тайм-менеджменту

За даними Atto Time 86% студентів стикаються з проблемами правильної організації свого часу, а 95% студентів взагалі відкладають всі справи на останній момент, при цьому 87% студентів визнають що могли отримати кращі оцінки, якщо б правильно розпланували навантаження.

Зазвичай більшості людей складно самотужки дисциплінувати себе та організувати свій час правильно. Тому, саме для вирішення такої проблеми, є безліч додатків, що допомагають організувати своє робоче середовище, раціонально розподілити час на справи та правильно все спланувати.[1]

### 2.2.3 Аналіз системи moodle

Система Moodle – це платформа, що використовуються для створення веб-курсів чи веб-сайтів для дистанційного навчання (рис. 2.3). З її допомогою всі університети змогли організувати дистанційне навчання після початку covid-19 у 2019 році. Ця платформа дозволяє ефективно управляти всіма ресурсами, що на ній розміщуються та полегшити дистанційне навчання в університетах.

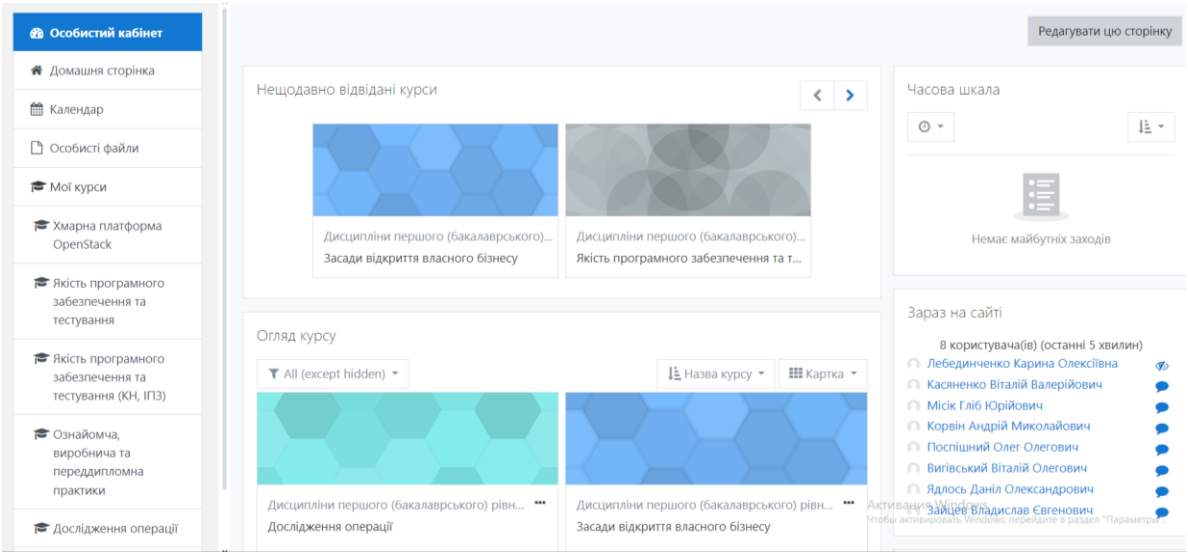



Рис. 2.3 Інтерфейс системи moodle

Закладам вищої освіти не потрібно розробляти власну веб-платформу для організації онлайн-навчання. Тож ця система є ефективним та підходящим винаходом.

Варто зазначити, що цю систему в університетах використовують й по сей день, незважаючи на очний формат навчання. Ця платформа дає можливість викладачам повністю викласти весь матеріал дисципліни, організувавши весь курс за темами, модулями чи підрозділами. Таким підхід покращує ефективність навчання та заощаджує час як студентам, так і викладачам.

**Система оцінювання**. Дозволяє студентам та викладачам запроваджувати прозору систему оцінювання за допомогою збереження успішності кожного студента та можливості створення оцінювання для кожної дисципліни та кожного окремого завдання, а автоматизація всіх ключових процесів заощаджує час як студенту, так і викладачу.

**Організація дисциплін.** За допомогою цієї платформи окремі викладачі курсів чи структури закладів вищої освіти можуть ефективно налагодити виклад матеріалів для студентів. Такий підхід впорядковує всі матеріали, що несе собою ефективну роботу. Перевагою цієї системи є те, що викладачу треба лише один раз впорядкувати та завантажити всі матеріали, що налагоджує один раз роботу на декілька років.

**Комунікація.** Платформа надає різні можливості для комунікації:

– особисті чати;
– групові чати;
– форуми;
– опитування та ін.

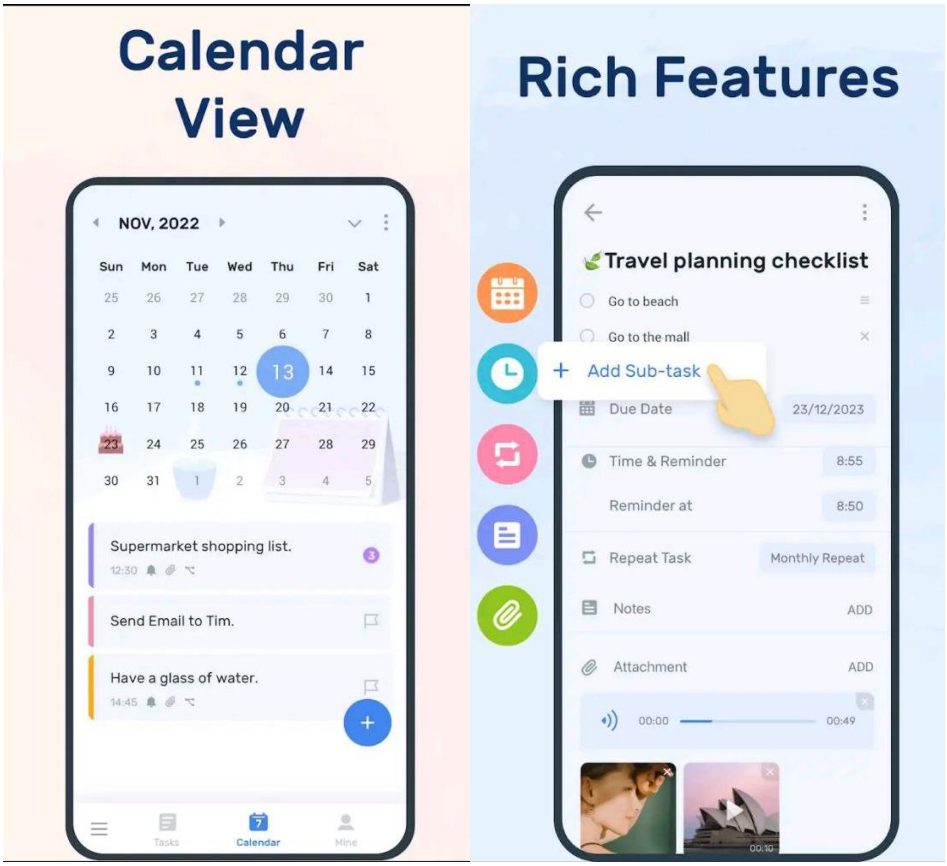
Але як показує статистика, то викладачі та студенти рідко використовують технології комунікації платформи. Зазвичай для спілкування обирають месенджери, бо це зручно та більш ефективно.

### 2.2.4 Проблематика організації роботи

В більшості випадків буває незручно встановлювати купу додатків на різні гаджети. В свою чергу така не зосередженість поміж додатками та планерами може призвести до ще однієї додаткової трати часу на організацію та збору всієї інформації до купи. Тому зручніше все тримати в одному додатку, або хоча б щоб більшість необхідного функціоналу було зосереджено в одному додатку.

На рис. 2.4 наведено лише невелику кількість додатків, які можуть допомогти з організацією та впорядкуванням будь-чого. Додатки можуть допомогати з:

– організацією задач на день, тиждень чи місяць;
– відслідковуванням прогресу;
– дотриманням дисципліни;
– створенням чек-листів;
– відслідковуванням продуктивності;
– створенням нагадувань тощо.



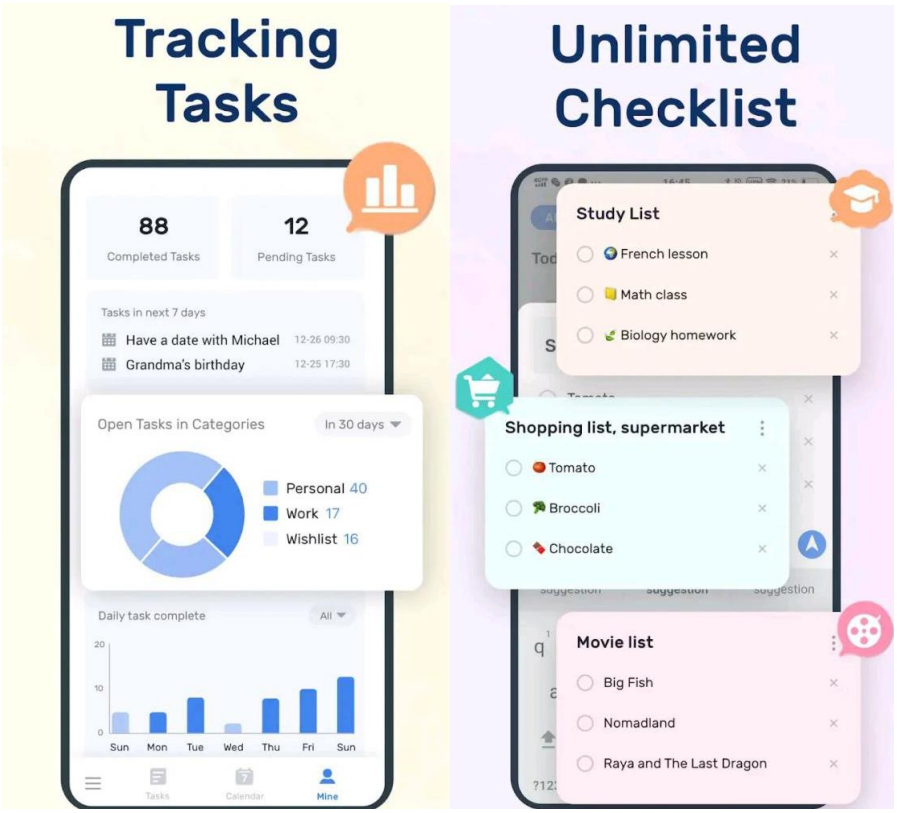



Рис. 2.4 Приклади додатків для планування чи організації задач

### 2.2.4 Вирішення проблеми розфокусування

Для студентів (особливо перших курсів) головним пріоритетом є навчання. Тому досить вдалим рішенням стане інтеграція додатку організації робочого плану студента в систему e-moodle для більшої зосередженості студентів. Завдяки такому рішенню можна навіть тісно пов’язати обидві системи.

Наприклад на такі важливі заняття як екзамени, заліки, кураторські години або лабораторні можна поставити автоматичні нагадування. Оцінки за предметами можна реалізувати разом з дисциплінами, адже таким чином студент може побачити свою успішність протягом всього семестру, що допоможе йому звернути уваги на недоліки та недосконало вивчені теми. Також, такий підхід допоможе спрогнозувати загальну оцінку за семестр, що дозволяє своєчасно виправити неприйнятні оцінки. А от інтеграція з списком задач допоможе створювати короткострокові та довгострокові плани без потреби переглядати купу додатків та нотатків.

Таке рішення допоможе сконцентруватись студентам на навчанні, підвищить їх ефективність та продуктивність, а також упорядкую повну картину успішності, що допоможе дисциплінувати графік навчання, роботи, саморозвитку та відпочинку, не даючи перенавантажитись студентам.

## 2.3 Технології, що використовувались в ході кваліфікаційної роботи

### 2.3.1 Загальна інформація

Створення веб-застосунку для організації робочого плану студентів є досить складним процесом та включає в себе вибудову складної архітектури додатку. Якісно створена архітектура несе собою стабільність та швидкість роботи додатка. Майбутній додаток має обробляти велику кількість інформації, тож потрібно чітко продумати кожну функцію та кожен рядок коду задля уникнення високої завантаженості та некоректної роботи веб-додатку.

В сучасному світі розробки веб-додатків використовують так звані фреймворки. Використання таких технологій значно покращує процеси розробки на всіх її рівнях.

**Процес розробки.** Перше, що потрібно зазначити – наявність шаблонів та можливість їх створення значно полегшує та пришвидшує роботи, зберігаючи основну структуру та якість всіх компонент. Реактивність та динамічність позбавляють написання складної логіки для реалізації навіть самого мінімального функціоналу.

**Структура проекту.** В кожному окремому фреймворку зазначається чітка структура проекту та протоколи, що допомагають створювати структуровано-впорядковані проекту. І це питання не лише одного веб-додатку. Такий підхід дозволяє дотримуватись більш менш єдиного підходу архітектури проекту по всьому світу, що зменшує кордони між розробниками та покращує адаптацію розробки веб-додатків.

Окрім вище перелічених переваг фреймворки мають чудову вбудовану організацію безпеки, можливість розширення функціоналу чи окремих модулів, а

також всесвітню підтримку та велику купу документації, завдяки своїй популярності.

### 2.3.2 Використання додатку Figma для створення макету

Для початку роботи по створенню веб-додатку є ряд процесів без яких не слід переходити до безпосереднього створення проекту. Одним з таких етапів є створення прототипу (макету) веб-сайту. Створення макету перш за все допомагає візуалізувати та створити приблизну архітектуру інтерфейсу додатку. Такий підхід значно економить час та ресурси, адже створювати веб-додаток за існуючим макетом значно легше, аніж розробляти все в ході роботи.

Проаналізувавши всі існуючі інструменти для даної задачі, було встановлено, що підходящим рішенням в цьому питанні є такий інструмент як Figma. Додаток Figma є одним з найпопулярніших додатків для створення прототипів інтерфейсу.

По-перше, він безкоштовний та користувачам не потрібно обтяжувати себе встановленням програмного забезпечення або додаткових плагінів для того, щоб працювати з цим інструментом, адже програмою можна користуватись безпосередньо в браузері. По-друге, над прототипом проекту можуть одночасно працювати декілька спеціалістів в режимі реального часу, що зменшує межі між командою та значно покращує якість роботи. По-третє, Figma має досить простий та зрозумілий інтерфейс, що не завдає фахівцям додаткових проблем під час роботи. І на останок, це програмне забезпечення має широку підтримку та розповсюдженість, що дозволяю встановлювати додаткові плагіни інструментів та розширювати можливості при розробці інтерфейсу. В додаток до сього, Figma має історію збережень, що часто буває необхідним при повернені до певного статусу розробки.

В Таблиця 2.1 розміщено результати аналізу популярних додатків для створення макетів. Деякі додатки мають складний інтерфейс або складність використання. Тому при розробці проекту перевага була на стороні платформи Figma як за власними вподобаннями, так і за суровою оцінкою аналізу.

Таблиця 2.1

Порівняння різних ПО для створення макетів

| **Назва** | **Figma** | **Adobe XD** | **Avocado** | **Canva** |
|---|---|---|---|---|
| Доступність | Так | Ні | Ні | Так |
| Потреба завантаження | Ні | Так | Так | Ні |
| Розширення можливостей (встановлення плагінів) | Так | Так | Так | Так |
| Простий шерінг (демонстрація) | Так | Ні | Ні | Так |
| Зручність | Так | Ні | Так | Так |

### 2.3.3 Використання HTML для створення структури веб-сайту

Використання HTML у веб-додатку передбачає створення чіткої структури веб-сторінки та всього документу з урахуванням майбутнього дизайну та семантичних правил для подальшого коректного відображення в усіх браузерах, а також якісного відображення в пошукових системах.

Але застосування фреймворку React.js передбачає використання такого формату як JSX. JSX – це таке розширення JavaScript, яке дозволяє інтегрувати мови розмітки безпосередньо в код, та не відривати його від логіки додатку. Такий підхід дозволяє наявно бачити майбутній UI та передбачати можливі помилки, що часто виникають під час взаємодії логіки з HTML. Не виключною перевагою стала й можливість додавати змінні або деякий функціонал безпосередньо в структуру веб-сторінок.[9]

```
toDoDoneList = toDoDoneListStore.map((item, index) => (
  <li key={item.id}>
    <input
      type="checkbox"
      id={"li_id_" + item.id}
      className="todo-chekbox"
      onChange={(e) => handlerUndoneTask(e)}
      defaultChecked
    ></input>
    <div className="li-text">{item.name}</div>
    <div className="li-date">{item.date}</div>
    <div className="li-priority">{item.priority}</div>
    <div className="li-note">{item.note}</div>
    <div className="todo-more-btn"></div>
  </li>
));

return (
  <>
    <div className="todo-table">
      <div className="tbl-title">
        <div className="tbl-name">Назва задачі</div>
        <div className="tbl-date">Дедлайн</div>
        <div className="tbl-priority">Пріорітетність</div>
        <div className="tbl-note">Примітка</div>
      </div>
      <ul>{toDoDoneList}</ul>
    </div>
  </>
);
```

Рис. 2.5 Приклад JSX формату в коді

На рис. 2.5 можна побачити приклад JSX розширення. Тут наявно показано, що за допомогою використання таких виразів фреймворк не тільки сам розуміє та визначає як рендерити елементи, але й ще це значно полегшує роботу з DOM-деревом. На чистому JavaScript доводиться прописувати багатофункціональної логіки для створення нових елементів чи зміни дерева елементів, але завдяки JSX при створенні веб-додатку з’являється можливість подібних махінацій не роботи, що значно економить час на створення та обчислювальні потужності девайсів.

### 2.3.4 Використання CSS для стилізації веб-додатку

Важливим кроком у створенні веб-додатків є стилізація елементів веб-сторінки. До цього питання потрібно поставитись з не аби якою уважністю, адже якісно функціонуючий додаток з боку логіки ще не є гарним додатком з точки зору користувача.

Оскільки CSS не має чітких правил використання, то в світі серед розробників затвердились свої методи та принципи якісного використання стилей з точки зору чистоти коду.

По-перше, потрібно дотримуватись кросбраузерності. Кросбраузерність – це властивість веб-сторінок коректно відображатись та функціонувати в будь-яких існуючих браузерах (або хоча б в найпопулярніших). В частині випадків доводиться застосовувати додаткові властивості стилей для коректного відображення в певних браузерах.[5]

По-друге, при стилізації веб-сторінок неодмінно потрібно пам’ятати про адаптивність. Адаптивність дозволяє підлаштовувати відображення веб-додатків під різні типи пристроїв такі як:

– великі, середні та малі монітори;
– ноутбуки;
– телевізори;
– планшети;
– смартфони різної роздільної здатності;
– та навіть smart-годинники.

За допомогою мета-тегів прописуються правила під окрему роздільну здатність, що буде вказувати браузеру як контенту на сторінках себе поводити при різних розмірах екранів.

По-третє, треба продумати який підхід краще використовувати в тому чи іншому випадку – чи то краще використати Grid Layout, чи Flex Layout. Це два основних направлення в сучасній адаптивній стилізації контенту, що широку застосовується по всьому світу.

В додачу, необхідно чітко продумати всю структуру стилів. Здавалось, що можна тут продумати? Але стилізація є досить затратним процесом в обчислювальній пам’яті девайсів, тож якщо писати громіздкий код, не думаючи про оптимізацію, то в завершенні можна отримати веб-додаток, який буде повільно працювати навіть при стабільному швидкому інтернет-з’єднанні. Також, потрібно притримуватись правила DRY (Don`t repeat yourself), адже занадто багато однакового коду несе й зайве навантаження, й не найкращим шляхом впливає на чистоту та читабельність коду.

За допомогою емуляторів можна подивитись як веб-додаток буде відображатись на різних типах пристроїв та при різній орієнтації дизайну. Емулятори необхідні для більш точного аналізу адаптивності веб-додатку. І хоча в браузері існують Інструменти розробника (DevTools), але вони мають ряд недоліків перед емуляторами та не завжди дають чітку картину для аналізу. На рис. 2.6 На перший погляд веб-сайт виглядає не так вже й погано, але якщо уважно придивитись, то стає помітно, що деякі деталі не зовсім коректно відображаються як от рядок пошуку та логотип чи основний контент, який не поміщається в зону видимості.

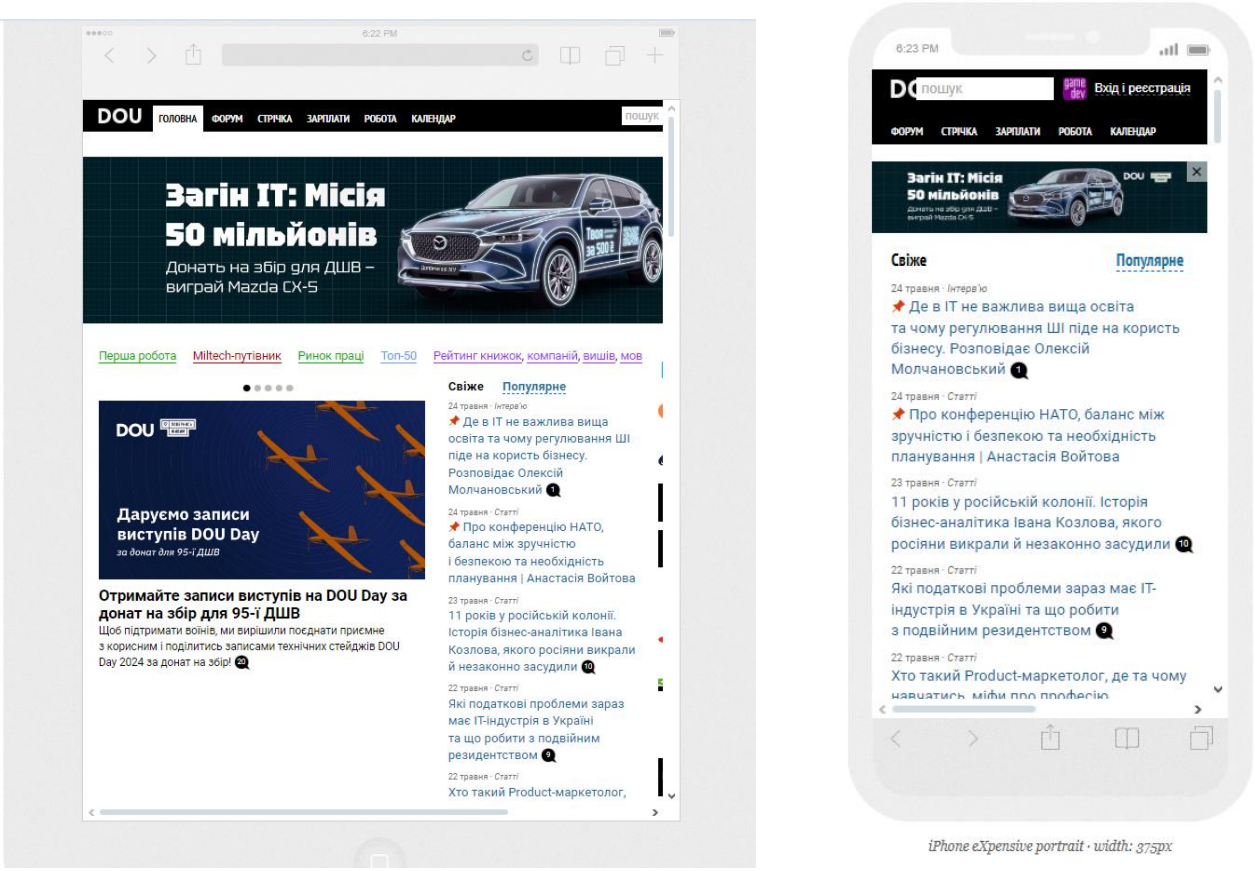



Рис. 2.6 Тестування адаптивності сайту в емуляторі

### 2.3.5 Використання JavaScript для розробки функціоналу

Використання мови програмування є невід'ємною частиною створення веб-додатка. Саме за допомогою JavaScript створюється вся логіка та функціонал додатку. Будь-яка функція кнопки, реалізація випадаючого списку, перемикання між сторінками, сортування чи відображення контенту та ін. – все це реалізується за допомогою JavaScript.

Також потрібно зазначити, що мова програмування є безпосереднім провідником між бібліотеками, фреймворками, серверною частиною та додатком.

Ще однією особливістю JavaScript є таке поняття як js-in-css. Дуже часто розробникам доводиться використовувати функціональну мову програмування для стилізації контенту, адже стилізація передбачає собою анімації, динамічну зміну контенту та інтерактивність. І, в багатьох випадках, можливості CSS не містять в своєму функціоналі можливостей для створення подібного контенту. Тож не рідко доводиться звертатись до JavaScript задля реалізації цікавого подання контенту.

### 2.3.6 Використання бібліотеки Redux Toolkit

Бібліотека Redux є невід'ємною частиною під час розробки веб-застосунка на React.js. Її функціонал передбачає створення сховища з власним функціоналом для кожної змінної, що дозволяє маніпулювати цими самими змінними з будь-якого куточку функціоналу.

Функціонал цієї бібліотеки необхідний для взаємодії компонент між собою. React.js, звісно, надає таку можливість, але головна проблема закладається в тому, що обмін даними можна здійснювати строго по структурній ієрархії компонентів, а такий підхід є доречним лише тоді, коли треба передати дані з батьківської компоненту в дочірню. Якщо уявити собі більш менш функціональний додаток, то можна зрозуміти, що компонент в ньому може бути безліч, а вкладень одних компонент в інші – ще більше. До того ж, часті випадки, коли з однієї компоненти треба передати дані в інші, не вкладену. І от тоді перед розробниками постає досить складна проблема реалізації обміну даними між не пов'язаними структурно компонентами.[2]

Тому, задля екологічного обміну даними між компонентами при розробці веб-додатків на React.js розробники використовують додатку бібліотеку Redux для керування станом компонентів та їх даними.

# 3 РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ

## 3.1 Архітектура проекту

Проаналізувавши запропоновані варіанти архітектури проектів було обрано наступний підхід: в кореневій папці src (головна папка всього проекту за замовчуванням) створюється папка components для створення кожної окремої великої компоненти. В папці components створюється папка під кожну окрему компоненту, (до прикладу) створюється папка header, в якій створюється папка images, файли header_styles.css та header.js. Таким чином структура кожної компоненти виглядає як показано на рис. 3.1.

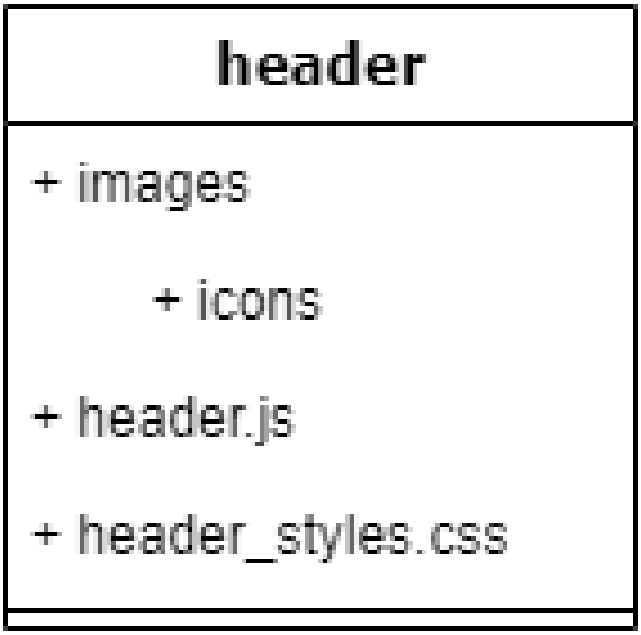



Рис. 3.1 Структура компоненти React.js

Такий підхід дозволяє забезпечити єдину якісну структуру всього проекту й не скидати в купу всі зображення, стилі чи компоненти, а лаконічно виокремити кожну компоненту з власними складаючими елементами.

Таким чином структура папка components виглядає як на рис. 3.2. Варто зазначати, що в папці images, окрім папки icons може міститись папка content саме для зображень, що будуть в контентній частині веб-сторінок, але в рамках проекту необхідності створювати додаткові папки немає.

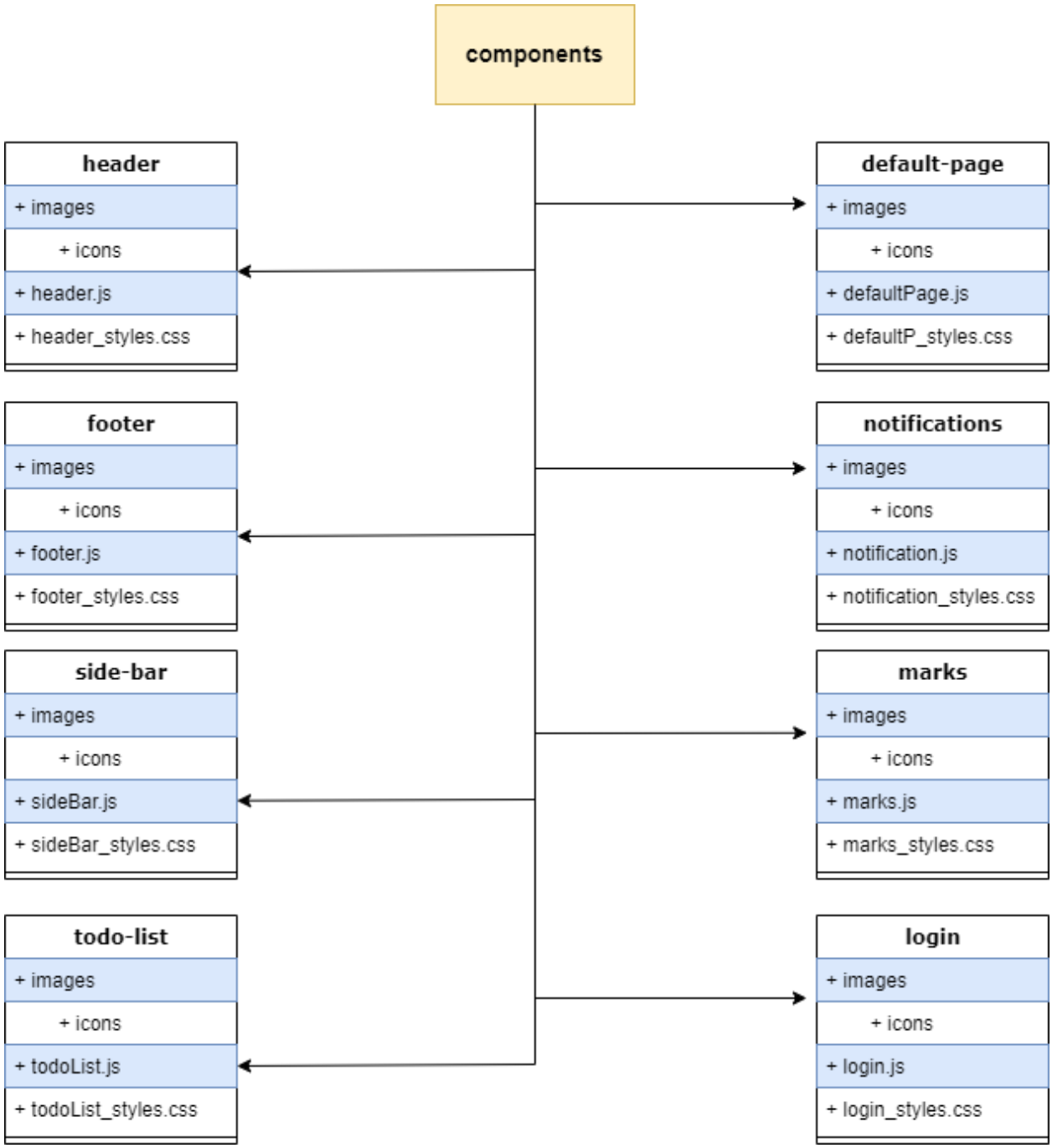



Рис. 3.2 Структура папки components

В даному випадку кожна окрема папка – це веб-сторінка, що містить в собі свою функціональність, логіку, стилі та певні дані, що забезпечує незалежність цієї самої компоненти.

Також в кожної окремої компоненти можуть бути свої внутрішні компоненти. Їх також можна виносити окремими файлами, але оскільки таких підкомпонент не так багато, було прийнято рішеня виносити їх окремо в файлі головної компоненти.

Ось як, наприклад на рис. 3.3 та рис. 3.4 компоненти TodoDoneTable та ModalNewTask винесені окремими компонентами з власною чіткою логікою та UI.

```
export function TodoDoneTable() {
  const toDoDoneListStore = useSelector(selectToDoDoneList);
  const dispatch = useDispatch();
  let toDoDoneList;

  function handlerUndoneTask(e) {
    const currentLi = e.target.id;
    toDoDoneListStore.forEach((item) => {
      if (item.id == currentLi.match(/\d+/)[0]) {
        dispatch(addTaskToList(item));
        dispatch(removeTaskFromDoneList(item.id));
      }
    });
  }

  toDoDoneList = toDoDoneListStore.map((item, index) => (
    <li key={item.id}>
      <input
        type="checkbox"
        id={"li_id_" + item.id}
        className="todo-chekbox"
        onChange={(e) => handlerUndoneTask(e)}
        defaultChecked
      ></input>
      <div className="li-text">{item.name}</div>
      <div className="li-date">{item.date}</div>
      <div className="li-priority">{item.priority}</div>
      <div className="li-note">{item.note}</div>
      <div className="todo-more-btn"></div>
    </li>
  ));

  return (
    <>
      <div className="todo-table">
```

Рис. 3.3 Компонента TodoDoneTable

```
export function ModalNewTask() {
  const dispatch = useDispatch();
  const modalToDoStore = useSelector(selectModalToDo);
  const taskCounterStore = useSelector(selectTaskCounter);

  function handlerAddTask() {
    const currentDate = new Date();
    let date;
    let todayDate = currentDate.toISOString().slice(0, 10);
    currentDate.setDate(currentDate.getDate() + 1);
    let tomorrowDate = currentDate.toISOString().slice(0, 10);
    currentDate.setDate(currentDate.getDate() - 2);
    let yestardayDate = currentDate.toISOString().slice(0, 10);

    const inpNameValue = document.getElementById("md_name").value;
    const selPriorityValue = document.getElementById("md_priority").value;
    const taTextValue = document.getElementById("md_text").value;
    const inpNoteValue = document.getElementById("md_note").value;
    const inpDateValue = document.getElementById("md_date").value;

    switch (inpDateValue) {
      case todayDate:
        date = "Сьогодні";
        break;
      case tomorrowDate:
        date = "Завтра";
        break;
      case yestardayDate:
        date = "Вчора";
        break;
      default:
        let day = inpDateValue.slice(8, 10);
        let month = inpDateValue.slice(5, 7);
        let year = inpDateValue.slice(0, 4);
        if(year === currentDate.toISOString().slice(0, 4))
          date = day + "-" + month;
        else
          date = day + "-" + month + "-" + year;
```

Рис. 3.4 Компонента ModalNewTask

Також до кожної компоненти прикріплені свої модульні стилі. Такий підхід дозволяє створювати стилі для кожної окремої компоненти не стикаючись з проблемою глобалізації стилів, коли через однакові назви стилі різних компонент накладаються один на одний, а постійно виокремлювати кожний ланцюжок ідентифікації чи кожному окрему елементу надавати власну назву – не завжди зручно та доречно. Таким чином код стилів виходить чистим, логічно продуманим та мінімалістичним, що добре впливає на його читабельність.

## 3.2 Структура компонент

### 3.2.1 Структура компоненти ToDoList

Компонента ToDoList – це досить складна функціональна складова проекту. Вона виступає окремою сторінкою, в якій є footer та header, а також дві таблиці з завданнями (або тасками), що потрібно виконати та виконаними задачами. На рис. 3.5 можна побачити список завдань, які потрібно виконати.

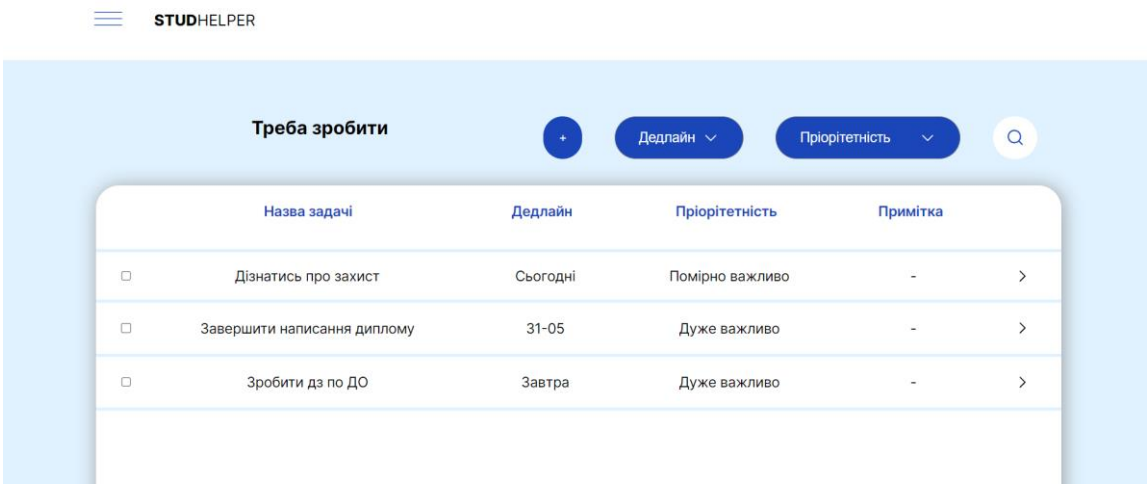



Рис. 3.5 Сторінка списку задач

Перша список задач має сортування за дедлайном та пріоритетністю завдань. Для списку виконаних задач таку можливість не надавали з точки зору неактуальності такого функціоналу.

Якщо відмітити задачу як виконану, то вона автоматично перекидається до другої таблиці в список виконаних завдань. На рис. 3.6 можна побачити задачу з попереднього списку. Також в цьому списку реалізована можливість повернення виконаного завдання в до списку тих, що потрібно виконати. Потрібно прибрати галочку й воно автоматично повернеться.

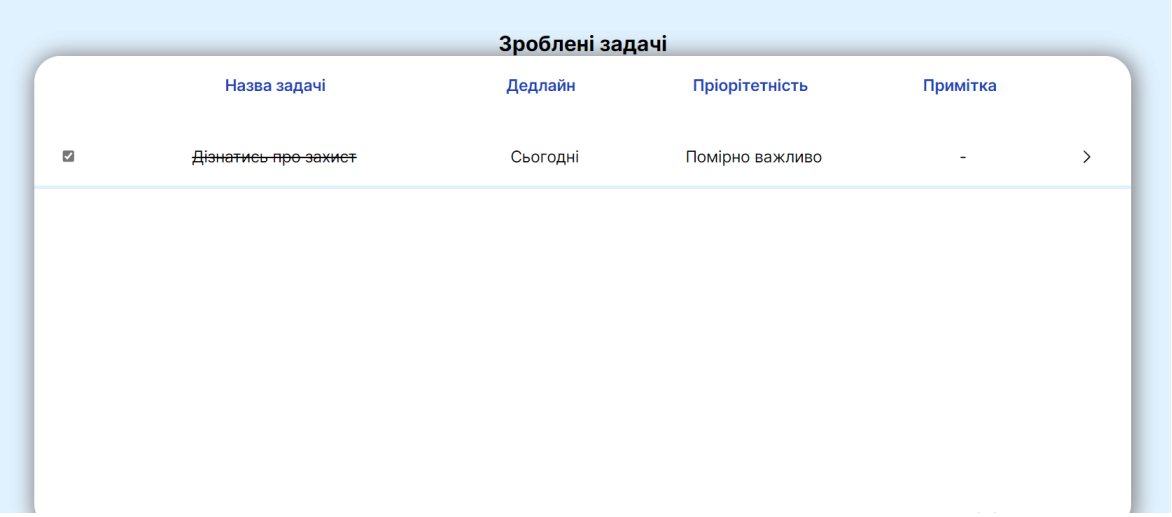



Рис. 3.6 Список виконаних задач

Можливість сортування реалізовувалась наступним чином:

– створюється редюсер за допомогою бібліотеки Redux Toolkit;

– створюються змінні value, showList, deadlines, priorityValue, deadlineValue;

– створюються відповідні функції, які оброблюють дані: addTaskToList, removeTaskFromList, changeShowList;

– до файлу компоненти під’єднуються потрібні зміні та функції, а також вбудовані функції бібліотеки;

– навішуються обробники (функції) подій на кожен з select-ів.

Відповідний код, який все це реалізує можна побачити на рис. 3.7 та 3.8. Функція addTaskToList додає нову задачу при її створені користувачем та вилучає необхідну інформацію по відповідним змінним. Функція removeTaskFromList видаляє задачу, коли користувач відмічає її як виконану. Функція changeShowList забезпечує відповідне сортування, щодо вибору користувачем відповідних фільтрів (дедлайн чи пріорітетність) та оновлює автоматично список задач за обраними критеріями.

```
import { createSlice } from "@reduxjs/toolkit";

export const toDoListSlice = createSlice({
  name: "toDoList",
  initialState: {
    value: [],
    showList: [],
    deadlines: [],
    priorityValue: true,
    deadlineValue: true,
  },
  reducers: {
    addTaskToList: (state, action) => {
      state.value = [action.payload].concat(...state.value);
      state.showList = [...state.value];
      state.deadlines.push(action.payload.date);
    },
    removeTaskFromList: (state, action) => {
      let newList = state.value.slice();
      newList.forEach((item, index) => {
        if (item.id == action.payload) newList.splice(index, 1);
      });
      state.value = newList;
      state.showList = newList;
    },
    changeShowList: (state, action) => {
      // let newList;
      if (action.payload.select === "sel_priority")
        state.priorityValue = action.payload.value;
      else
        state.deadlineValue = action.payload.value;
        console.log('Сьогодні' == true)
        state.showList = state.value.filter(
          (item) =>
            item.priority == state.priorityValue &&
            item.date == state.deadlineValue
        );
```

Рис. 3.7 Код файлу toDoListReducer.js

На рис. 3.8 вбудована функція dispatch для подальшої комунікації між редюсером та додатком. До кожного select (випадаючого списку з критеріями) під'єднується свій обробник, який викликає відповідний код для обробки змін в редюсері. Функція handlerChangeFilters відповідає за передачу певних даних у функції, що працює зі сховищем.

Також на рис. 3.8 є функція-редюсер changeModalToDoStatus. Вона потрібна для обробки статусу модального вікна через яке користувач створює нову задачу.

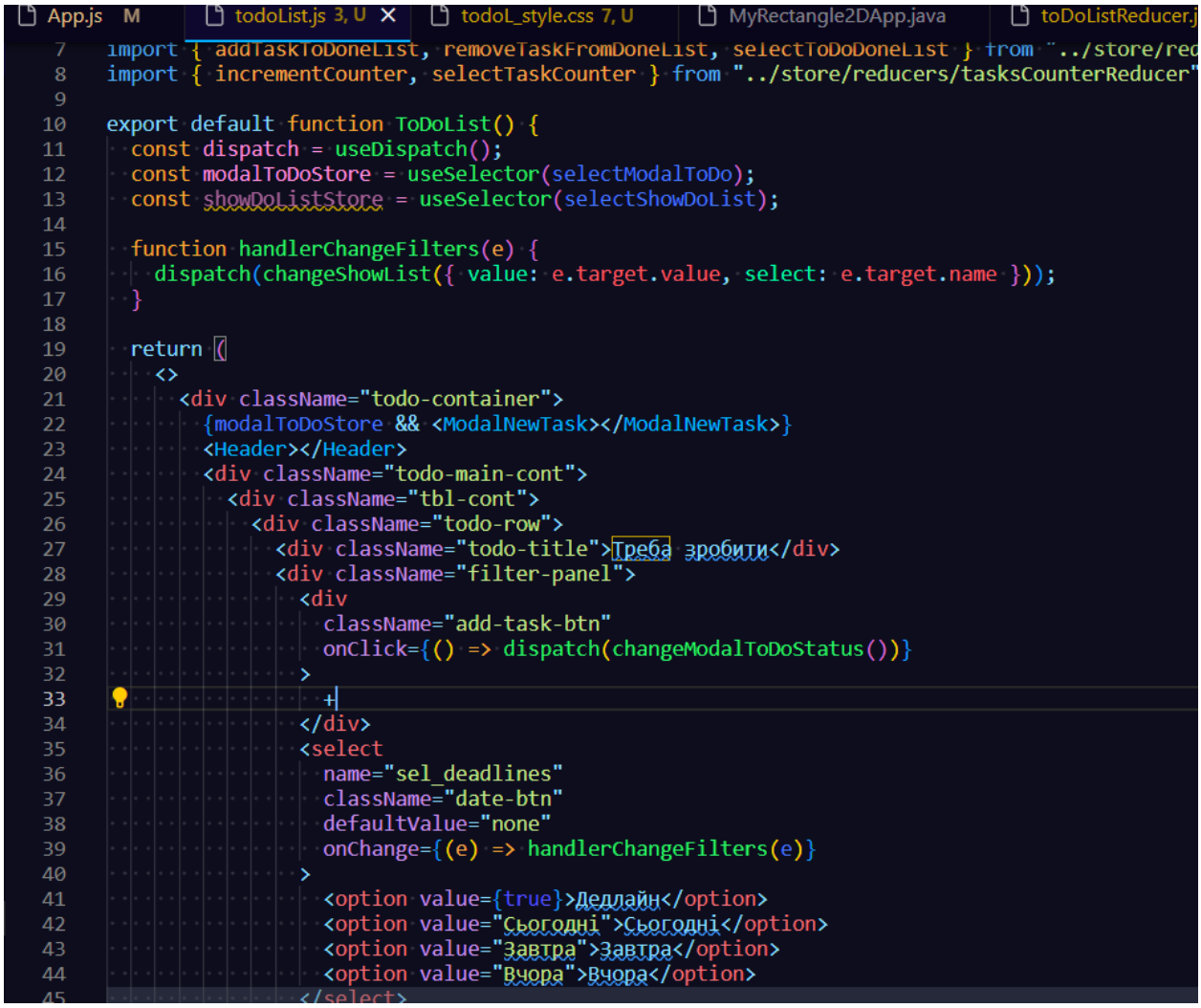



Рис. 3.8 Код файлу toDoList.js

На рис. 3.9 можна побачити код <TodoTable></TodoTable> та <TodoDoneTable></ TodoDoneTable >. Це внутрішні компоненти сторінки, які відповідають за логіку та відображення двох списків задач. Хоча вони схожі між собою, але мають дещо різну логіку та різні дані, тому їх поділено на дві окремі компоненти.

За допомогою формату JSX можна значно скоротити код та зробити його більш лаконічний. Кожна з долучених компонент також має свій UI в відповідному форматі, а React.js за допомогою вбудованих механізмів самостійно реалізовую завантаження та вставку в DOM-дерево без зайвої уваги.

```
                <option value="Дуже важливо">Дуже важливо</option>
                <option value="Важливо">Важливо</option>
                <option value="Помірно важливо">Помірно важливо</option>
                <option value="Не важливо">Не важливо</option>
              </select>
              <div className="search-btn"></div>
            </div>
          </div>
          <TodoTable></TodoTable>
        </div>

        <div className="tbl-cont">
          <div className="todo-title">Зроблені задачі</div>
          <div className="todo-table table-done">
            <TodoDoneTable></TodoDoneTable>
          </div>
```

Рис. 3.9 Приклад коду toDoList.js

Ось, власне, на рис. 3.10 можна побачити як виглядає компонента TodoTable зі своєю логікою та HTML-елементами. Функція handlerDoneTask спрацьовує при натискані користувачем на input, щоб повідомити веб-застосунок про виконання задачі. Після цього функція знаходить відповідну задачу в масиві сховища та переносить її зі списку задач до списку виконаних задач.

React.js надає прекрасну можливість формувати списки з елементами за допомогою перебираючого методу масивів map. Цей метод є вбудованим методом для сортування масивів в JavaScript. Розробники фреймворку вирішили його задіяти в своєму функціоналі, надавши розробникам веб-додатків можливість формувати деяку частину UI. На рис. 3.10 є зміна toDoList, яка передбачається для завантаження елементів спику li в ul. Змінній toDoList присвоюється стрілкова функція з методом map, що сортує масив сховища зі списком задач, створених користувачем. Функція перебирає кожен елемент масиву та відповідно вставляє дані до елементу li. Після проходу методу по всьому масиву список всіх задач відображається в таблиці веб-сторінки, за допомогою вставки змінною toDoList в UI компоненти. Змінна хоч і зберігає в собі певну логіку, але по закінченню цієї логіки вона повертає результат роботи функції. Таку реалізацію коду не вдалось би створити на чистому JavaScript.

```
export function TodoTable() {
  let toDoList;

  function handlerDoneTask(e) {
    const currentLi = e.target.id;
    toDoListStore.forEach(item => {
      if (item.id == currentLi.match(/\d+/)[0]) {
        dispatch(addTaskToDoneList(item));
        dispatch(removeTaskFromList(item.id));
      }
    });
  }
    toDoList = showDoListStore.map((item, index) => (
      <li key={item.id}>
        <input
          type="checkbox"
          id={"li_id_" + item.id}
          className="todo-chekbox"
          onChange={(e) => handlerDoneTask(e)}
        ></input>
        <div className="li-text">{item.name}</div>
        <div className="li-date">{item.date}</div>
        <div className="li-priority">{item.priority}</div>
        <div className="li-note">{item.note}</div>
        <div className="todo-more-btn"></div>
      </li>
    ));

  return (
    <>
      <div className="todo-table">
        <div className="tbl-title">
          <div className="tbl-name">Назва задачі</div>
          <div className="tbl-date">Дедлайн</div>
          <div className="tbl-priority">Пріорітетність</div>
          <div className="tbl-note">Примітка</div>
        </div>
        <ul>
          {toDoList}
        </ul>
      </div>
```

Рис. 3.10 Код компоненти TodoTable

Завершаючим функціоналом сторінки зі списком задач є модальне вікно, код якого демонструється на рис. 3.11 та 3.12. Головна функція handlerAddTask цієї компоненти спрацьовує після заповнення відповідних полей користувачем щодо завдання та натискання на кнопку «Додати задачу». Функція “витягує” всі дані з полей, встановлює відповідну дату та додає у сховище всю відповідну інформацію щодо задачі. Логіка з датою в цій функції має особливу ролі, адже в залежності від дати змінній «дедлайн» призначається певний зміст за параметрами, який виконується за допомогою функції JavaScrip “switch-case”.

```
export function ModalNewTask() {
  const dispatch = useDispatch();
  const modalToDoStore = useSelector(selectModalToDo);
  const taskCounterStore = useSelector(selectTaskCounter);

  function handlerAddTask() {
    const currentDate = new Date();
    let date;
    let todayDate = currentDate.toISOString().slice(0, 10);
    currentDate.setDate(currentDate.getDate() + 1);
    let tomorrowDate = currentDate.toISOString().slice(0, 10);
    currentDate.setDate(currentDate.getDate() - 2);
    let yestardayDate = currentDate.toISOString().slice(0, 10);

    const inpNameValue = document.getElementById("md_name").value;
    const selPriorityValue = document.getElementById("md_priority").value;
    const taTextValue = document.getElementById("md_text").value;
    const inpNoteValue = document.getElementById("md_note").value;
    const inpDateValue = document.getElementById("md_date").value;

    switch (inpDateValue) {
      case todayDate:
        date = "Сьогодні";
        break;
      case tomorrowDate:
        date = "Завтра";
        break;
      case yestardayDate:
        date = "Вчора";
        break;
      default:
        let day = inpDateValue.slice(8, 10);
        let month = inpDateValue.slice(5, 7);
        let year = inpDateValue.slice(0, 4);
        if(year === currentDate.toISOString().slice(0, 4))
          date = day + "-" + month;
        else
          date = day + "-" + month + "-" + year;
```

Рис. 3.11 Код компоненти ModalNewTask

Ця конструкція в даному випадку була обрана замість конструкції “if-else”, адже тут її використання призвело б до великої кількості однакового коду.

На рис. 3.12 видно продовження функції handlerAddTask, в якій створюється зміна з вже обробленими даними задачі та відправляється в масив сховища.

```
export function ModalNewTask() {
  function handlerAddTask() {
    const bodyLi = {
      id: taskCounterStore,
      name: inpNameValue,
      text: taTextValue,
      priority: selPriorityValue,
      note: inpNoteValue,
      date: date
    };

    dispatch(addTaskToList(bodyLi));
    dispatch(changeModalToDoStatus(!modalToDoStore));
    dispatch(incrementCounter());
  }


  return (
    <>
      <div
        className="sidb-plashka"
        onClick={() => dispatch(changeModalToDoStatus())}
      ></div>
      <div className="mod-nt-cont">
        <input type="text" id="md_name" placeholder="Введіть назву" />
        <select className="" id="md_priority">
          <option value="none">Пріорітетність</option>
          <option value="Дуже важливо">Дуже важливо</option>
          <option value="Важливо">Важливо</option>
          <option value="Помірно важливо">Помірно важливо</option>
          <option value="Не важливо">Не важливо</option>
        </select>
        <input type="date" id="md_date" />
        <textarea
          name=""
          id="md_text"
          placeholder="Введіть завдання"
        ></textarea>
        <input type="text" id="md_note" placeholder="Введіть примітку" />
        <button onClick={handlerAddTask}>Add task</button>
```

Рис. 3.12 Код компоненти ModalNewTask

Після функції демонструється UI самого модального вікна. Цікавим елементом тут є своєрідна “плашка”, яка затемнює всю сторінку, крім самого модального вікна. Така практика використовується у frontend-розробці по всьому світу для того підсвідомого розуміння користувача, що поки відкрито таке вікно, то вся інша сторінка не функціонує. Але при клікі на цю саму “плашку” вікно закривається.

На рис. 3.12 демонструється безпосередньо саме модальне вікно для створення нової задачі.

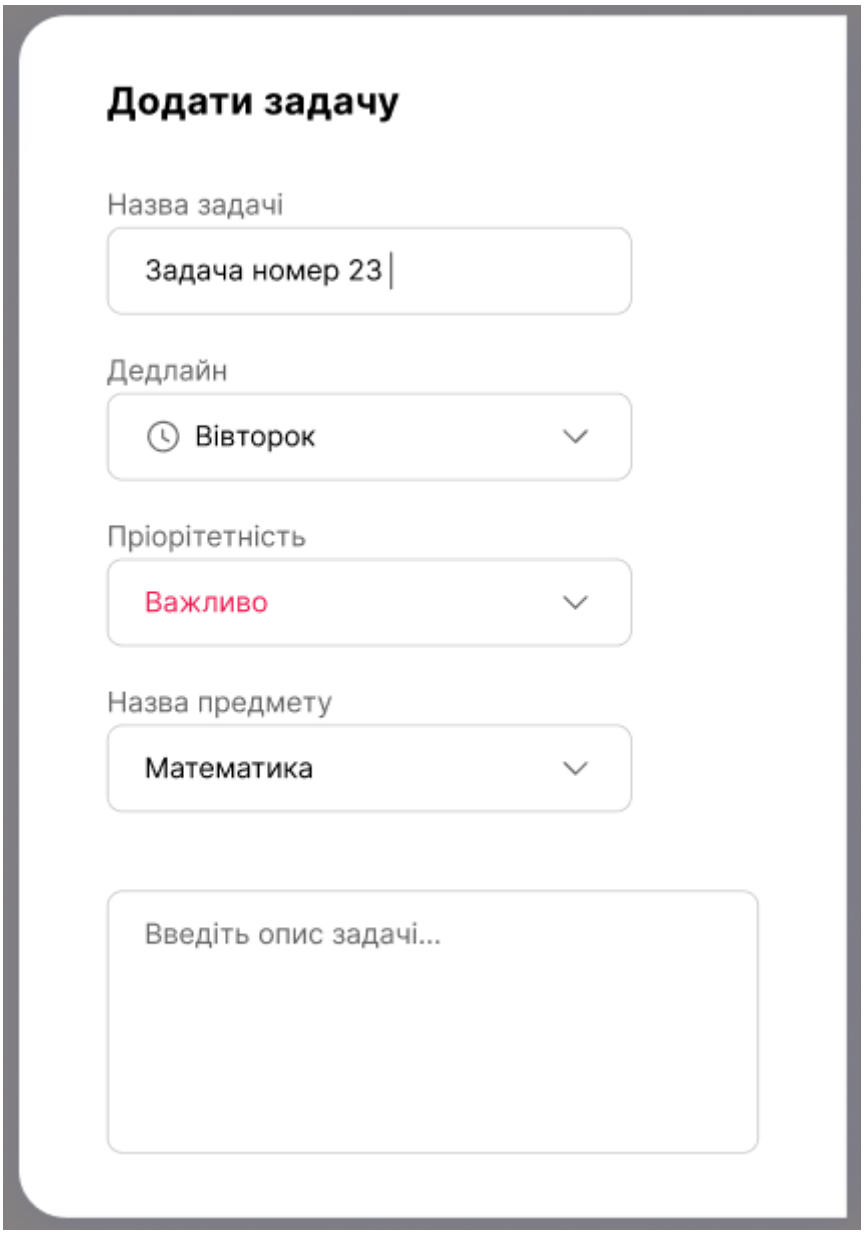



Рис. 3.12 Вікно для створення нової задачі

### 3.2.2 Структура компоненти SideBar

Компонента SideBar використовується, напевно, в кожному веб-додатку. Особливістю такого вікна є згрупування різних вкладок в одному місці. Якщо для великих екранів такий функціонал сформований для зручності, то для смартфонів чи планшетів – це необхідність.

В рамках проекту SideBar містить в собі наступні сторінки:

– список задач;

– нагадування;

– домашні завдання;

– розклад;
– оцінки;
– нотатки;
– налаштування;
– та кнопку входу/виходу.

Незважаючи на досить корисну функцію даної компоненти вона моє досить легку реалізацію, яка була використана при розробці. Рис. 3.13 є підтвердженням легкості реалізації даної задумки, на ньому повний код компоненти.

За допомогою списку були додані “кнопки” списку, що перемикають користувача на потрібну сторінку та до кожного елементу було прикріплено обробник подій, який перемикає користувача на потрібну сторінку. В сховищі є окремий редюсер, виділений під цю задачу, з його допомогою зміну, яку можна побачити на рис. 3.14, вбудований функціонал бібліотеки замінює на ту, що відповідає певній сторінці веб-застосунку. І от вже фреймворк разом з бібліотекою самостійно оновлюють контент сторінки на необхідний.

```jsx
export default function SideBar() {
  const dispatch = useDispatch();
  return (
    <>
      <div
        className="sidb-plashka"
        onClick={() => dispatch(changeSideBarStatus())}
      ></div>
      <div className="side-bar-cont">
        <div className="sb-list">
          <ul>
            <li
              onClick={() => {
                dispatch(changePage("todo"));
                dispatch(changeSideBarStatus());
              }}
            >
              <img src={ToDo} alt="" />
              Список задач
            </li>
            <li
              onClick={() => {
                dispatch(changePage("notifications"));
                dispatch(changeSideBarStatus());
              }}
            >
              <img src={Notes1} alt="" />
              Нагадування
            </li>
            <li>
              <img src={HT} alt="" />
              Д/З
            </li>
            <li>
              <img src={Shedule} alt="" />
              Розклад
            </li>
            <li
              onClick={() => {
                dispatch(changePage("marks"));
                dispatch(changeSideBarStatus());
              }}
            >
              <img src={Marks} alt="" />
```

Рис. 3.13 Код компоненти SideBar

```jsx
import { createSlice } from "@reduxjs/toolkit";

export const pageSlice = createSlice({
  name: "page",
  initialState: {
    value: "main",
  },
  reducers: {
    changePage: (state, action) => {
      state.value = action.payload;
    },
  },
});

export const { changePage } = pageSlice.actions;
export const selectPage = (state) => state.page.value;
export default pageSlice.reducer;
```

Рис. 3.14 Код файлу pageReducer.js

На рис. 3.15 демонструється безпосередньо меню, код якого представлений вище. Також варто звернути увагу на деяку анімацію при наведені, яка була реалізована за допомогою CSS. Також реалізація всіх ярликів була реалізована наступним чином

– до кожного елементу додавалось зображення відповідного ярлику;
– за допомогою CSS були застосовані відповідні розміри та відступи.

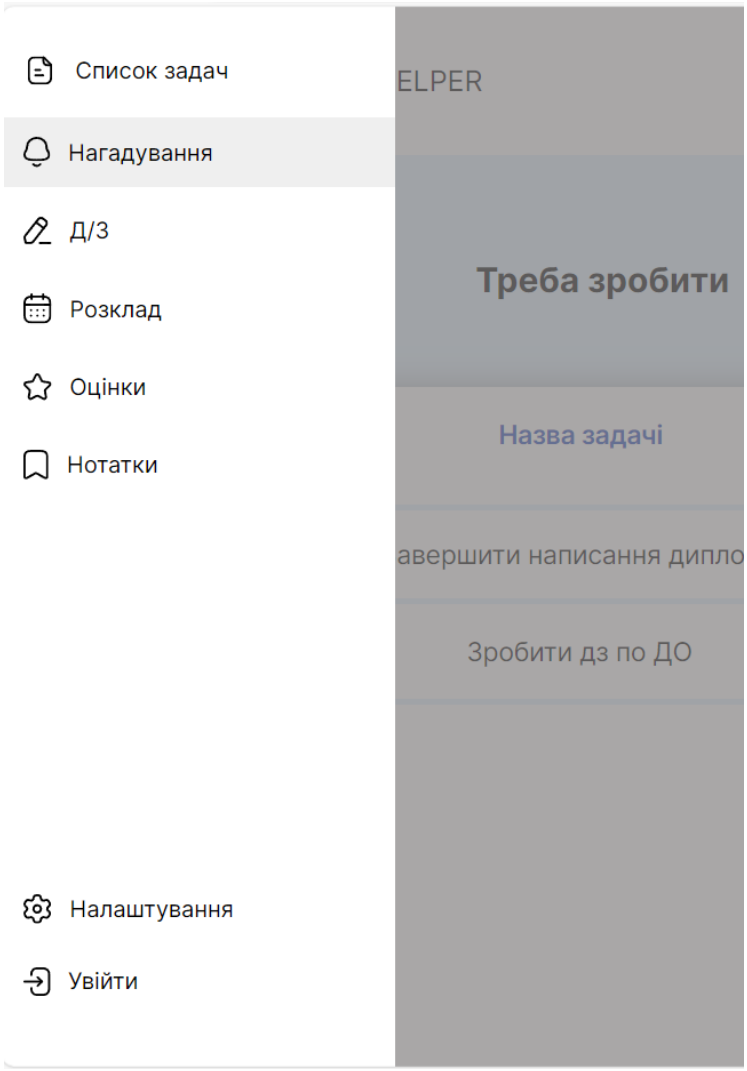



Рис. 3.15 Функціонуючий SideBar веб-додатку

До речі, зображення, які додавались до компонент інтерпретувались як зміні. Тобто, фреймворк імпортує зображення за вказаним шляхом, а розробник наділяє це зображення будь-якою змінною, після чого в коді можна просто вписувати назву змінної, а не постійно прописувати довгий шлях до місця, де зберігається ця змінна.

На рис. 3.13 можна побачити в тегу img в атрибуті src лежить не шлях до зображення, а просто імпортована змінна.

### 3.2.3 Структура компоненти DefaultPage

Компонента DefaultPage є головною сторінкою веб-застосунку. Вона несе в собі більш інформативний характер, ніж функціональний. На ній розташовані header та footer, а також “контейнери” з інформацією про кожну з функціональних сторінок.

На рис. 3.16 демонструється код даної веб-сторінки. Контент розділен на декілька умовних блоків з власним контентом. В кожному блоці є ще одна компонента з назвою «Container». Оскільки всі блоки на веб-сторінці мають майже однакову структуру, то повторювати один й той самий код чотири рази було б поганою ідеєю. Тож під час проектування було прийнято рішення виокремити певну частину UI, зробивши певний шаблон, який можна перевикористовувати. Завдяки такому рішенню вдалось оптимізувати код аж на 60 рядків та зберегти чистоту розробки.

В «Container» передається декілька змінних: назва, текст та зображення. І вже безпосередньо в цій компоненті ці дані займають свої місця. На рис. 3.17 можна побачити як це відбувається. Є відповідний шаблон з HTML-елементами, які мають власні класи, а дані, що потрапляють до компоненти з DafaultPage – оброблюються та додаються у відповідні місця.

Не дивлячись на схожість кожного блоків, в деяких є суттєва різниця в дизайні. Завдяки псевдокласам CSS “:nth-child()” можна до певного елемента за порядком додавати окремі класи, що ніяк не буде впливати на інші.

```jsx
export default function DefaultPage() {
  return (
    <>
      <div className="main-container">
        <Header></Header>
        <div className="f-block">
          <div className="carousel-block">
            <div className="cb-text-p">
              <div className="cb-title dp-title"></div>
              <div className="cb-text"></div>
              <div className="cb-btn"></div>
            </div>
            <div className="cb-image-p"></div>
          </div>
        </div>
        <div className="s-block">
          <div className="sb-col">
            <Container
              title={"Список задач"}
              text={
                "Забезпечує організацію інформації, підтримку навчального процесу та адм
              }
              image={Man1}
            ></Container>
          </div>
          <div className="sb-col">
            <Container
              title={"Нагадування"}
              text={
                "Забезпечує оповіщення про важливі події та дедлайни, для завчасного пла
              }
              image={Girl1}
            ></Container>
          </div>
        </div>
        <div className="th-block">
          <div className="nth-b-cont">
            <Container
              title={"Домашні завдання"}
              text={
                "Допомагає студентам та викладачам вчасно планувати виконання завдань, н
              }
              image={Girl2}
```

Рис. 3.16 Код компоненти DefaultPage

```jsx
export default function Container({title, text, image}){
  return (
    <>
      <div className="text-p">
        <div className="dp-title">{title}</div>
        <div className="dp-text">{text}</div>
        <div className="dp-btn">Перейти</div>
      </div>
      <div className="image-p">
        <img src={image} alt="" />
      </div>
    </>

  )
}
```

Рис. 3.17 Код компоненти Container

На рис. 3.18 демонструються ті блоки, про які вище зазначалось, і видно, що всі вони мають по зображенню, кнопці, назві та текст, але задумка UI/UX була створити якісний контент та урізноманітнити подачу контенту (хоча місцями він дещо однотипний). Тому біло вирішено десь його виділити, десь зменшити, десь по

іншому подати. І як можна спостерігати навіть після створення єдиного шаблону вдалось реалізувати індивідуальну стилізацію для кожного блоку.

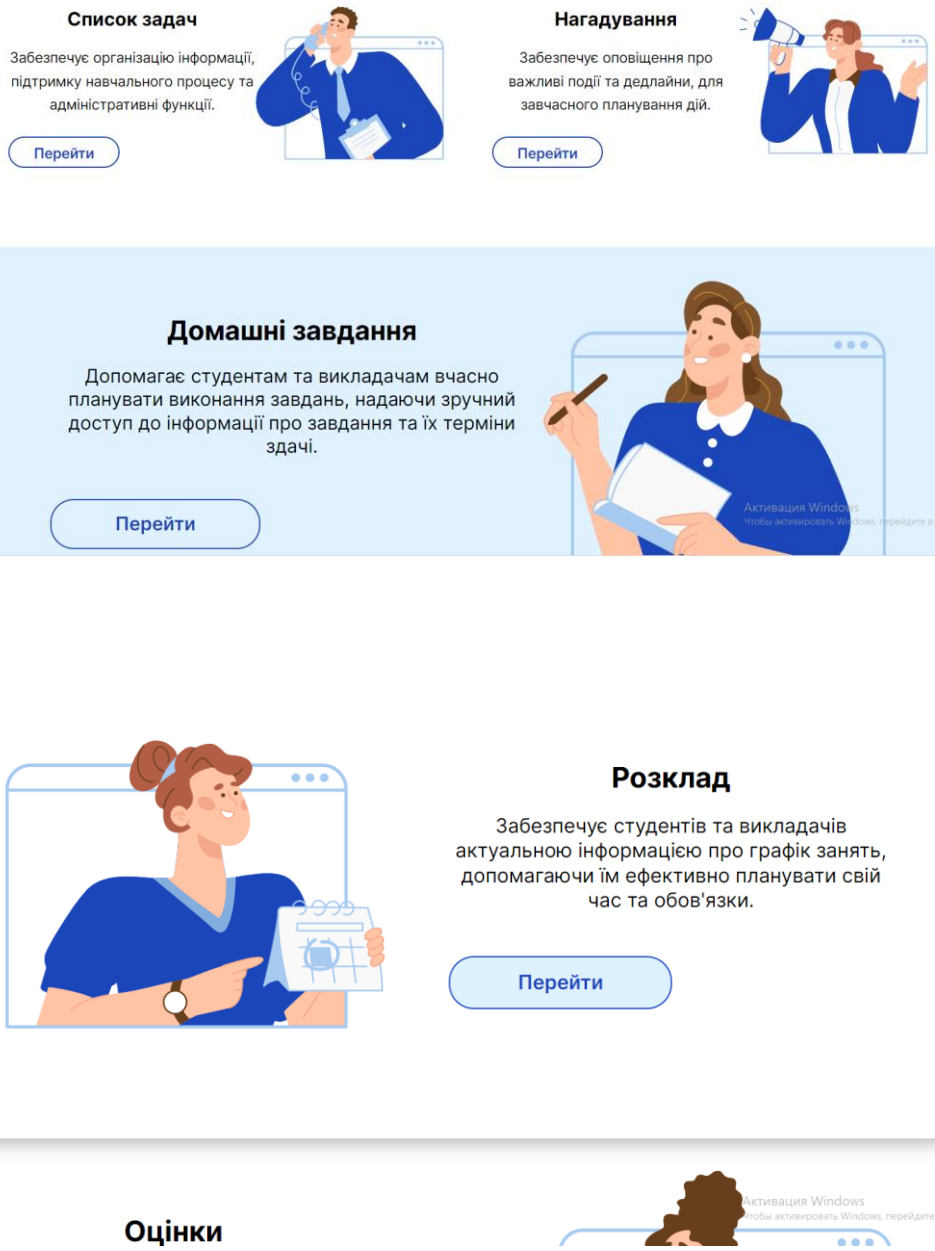



Рис. 3.18 Вигляд контентної частини Головної сторінки

### 3.2.4 Структура сховища store

Сховище store повністю створене та побудоване за допомогою за допомогою бібліотеки Redux. В проект додатково встановлюється Redux Toolkit для роботи з бібліотекою. В папці src створюється папка store, після чого обов’язково створюється папка reducers та файл store.js.

Файл store.js необхідний для того, щоб система зрозуміла та ідентифікувала його як сховище. В цьому файлі обов’язково ітеруються всі функції-редюсери (рис.

3.19), що застосовуються в проекті. Сам файл підключається та позиціонується як сховище в головному файлі всього проекту index.js (рис. 3.20).

```
export default configureStore({
  reducer: {
    page: pageReducer,
    sideBar: sideBarReducer,
    modalToDo: modalWToDoReducer,
    toDoList: toDoListReducer,
    toDoDoneList: toDoDoneListReducer,
    taskCounter: tasksCounterReducer,
  },
});
```

Рис. 3.19 Код файлу store.js

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);

reportWebVitals();
```

Рис. 3.20 Код файлу index.js

Всі функції-редюсери додаються в папку reducers, звідки потім імпортуються у веб-застосунку.

Самі редюсери створюються безпосередньо через вбудовану функцію бібліотеки createSlice. Наступні поля є обов'язковими для створення:

– ім'я (name);

– змінні (об'єкт initialState з переліком необхідних змінних);

– функції (об'єкт reducers, в якому реалізуються стрілкові функції з індивідуальною логікою).

Логіка кожної функції не обмежується, але я єдине виключення – значення потрібно оновлювати, а не замінювати. Тобто, якщо в initialState є певний масив з елементами, то і присвоювати потрібно змінений масив, а не видаляти з нього чи додавати елементи. Якщо з масиву потрібно видалити один елемент, то спочатку

копіюється старий, з копії видаляється масив, і вже потім копія присвоюється змінній. Це потрібно для того, аби бібліотека бачила які зміни відбулись в даних та самостійно внесла зміни в усі місця по коду, де використовуються ці дані.

Весь цей процес змін відбувається за допомогою state та action. Це вбудовані елементи редюсера (рис. 3.21), з якими розробники працюють безпосередньо в стрілковій функції. State – це стан всієї функції-редюсера, через нього треба звертатись до змінних (їх може бути необмежена кількість). Action – це дія, яка відбувається над змінною, вона містить в собі таку властивість як payload. За допомогою цієї властивості витягуються дані, які передаються в коді для змін та надається доступ для роботи з ними.

```
import { createSlice } from "@reduxjs/toolkit";

export const toDoDoneListSlice = createSlice({
  name: "toDoDoneList",
  initialState: {
    value: [],
  },
  reducers: {
    addTaskToDoneList: (state, action) => {
      state.value = [action.payload].concat(...state.value);
    },
    removeTaskFromDoneList: (state, action) => {
      let newList = state.value.slice();
      newList.forEach((item, index) => {
        if (item.id == action.payload) newList.splice(index, 1);
      });
      state.value = newList;
    },
  },
});

export const { addTaskToDoneList, removeTaskFromDoneList } = toDoDoneListSlice.actions;
export const selectToDoDoneList = (state) => state.toDoDoneList.value;
export default toDoDoneListSlice.reducer;
```

Рис. 3.21 Код файлу toDoDoneListReducer.js

# ВИСНОВКИ

В рамках кваліфікаційної роботи було досліджено сферу веб-розробки, її актуальні технології та рішення. Також було проведено аналіз та опитування емоційного стану студентів.

На основі отриманих даних було проведено аналіз та розробку проекту, який має на меті підвищити продуктивність, покращити дисципліну та емоційний стан студентів. В ході розробки веб-додатку були використані наступні технології: React.js, HTML, JavaScript, CSS, Redux Toolkit та Figma.

За допомогою вище перелічених технологій вдалось розробити прототип веб-застосунку, дотримуючись всіх правил UI/UX. На основі макету в Figma засобами веб-технологій було створено адаптивний інтерфейс веб-додатку та функціональні інструменти для організації робочого плану студента.

Веб-застосунок успішно пройшов всі тестування та відповідає поставленим вимогам на початку кваліфікаційної роботи.

В ході проекту були використані навички та знання, набуті за період чотирьох років навчання в закладі вищої освіти. За допомогою аналізу та критичного мислення по закінченню проекту отримано повноцінний веб-додаток з усім набором інструментів, що допоможе студентам організовувати своє робоче середовище.

## ПЕРЕЛІК ПОСИЛАНЬ


1. 9 time management statistics 2023: facts & key takeaways. Atto: Timesheet & Time Tracking Software. URL: https://attotime.com/blog/time-management-statistics (дата звернення: 05.04.2024).
2. Anthony A., Nathaniel M., Ari L. Fullstack React: The Complete Guide to ReactJS and Friends. Fullstack.io, 2017. 836 p.
3. Bibeault B., Resig J., Maras J. Secrets of the JavaScript Ninja. Manning Publications, 2016. 464 p.
4. Grant K. CSS in Depth. Manning Publications Co. LLC, 2018.
5. Harris A. HTML5 and CSS3 All-In-One for Dummies. CreateSpace Independent Publishing Platform, 2017.
6. Responsinator. Responsinator. URL: http://www.responsinator.com/ (дата звернення: 05.04.2024).
7. Robbins J. N. Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics. 4th ed. Beijing : O'Reilly, 2012. 603 p.
8. Verou L. CSS Secrets: Better Solutions to Everyday Web Design Problems. O'Reilly Media, Incorporated, 2015.
9. Вступ до JSX – react. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.legacy.reactjs.org/docs/introducing-jsx.html (дата звернення: 05.04.2024).
10. Лебединченко К.О., Піонтківський Є.Р., Фесенко М.А., Використання композиційного візуального мислення без тренувань в області різноманітних маніпуляцій з зображеннями / Науково-практична конференція «Проблеми комп’ютерної інженерії». К.: ДУІКТ, 2023 р. с. 34,35
11. Понад 70% українців відчувають стрес і знервованість – дослідження. The Village Україна. URL: https://www.village.com.ua/village/city/city-news/332171-ponad-70-ukrayintsiv-vidchuvayut-stres-ta-znervovanist-doslidzhennya (дата звернення: 05.04.2024).

12. Топ 10 програм для ефективного тайм-менеджемнта і планування. Wonder Web. URL: https://wonder-web.com.ua/blog/marketing-articles/top-10-program-dlya-efektivnogo-tajm-menedzhmenta-i-planuvannya/ (дата звернення: 05.04.2024).

13. Фесенко М.А., Лебединченко К.О., Піонтківський Є.Р. ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В МАРКЕТИНГУ / Науково-практична конференція «Сучасні досягнення компанії HEWLETT PACKARD ENTERPRISE в галузі IT та нові можливості їх вивчення і застосування». Збірник тез. – К.: ДУІКТ, 2023р., с. 39,40

14. Фесенко М.А., Лебединченко К.О., Піонтківський Є.Р. ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ У ВІЙСЬКОВІЙ СФЕРІ В УКРАЇНІ / Науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях ». Збірник тез. – К.: ДУІКТ, 2024р., с. 265,266

15. Фесенко М.А., Лебединченко К.О., Піонтківський Є.Р. ВИКОРИСТАННЯ БІБЛІОТЕКИ TENSORFLOW.JS ПРИ РОЗРОБЦІ ВЕБ-ДОДАТКУ/ Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. – К.: ДУІКТ, 2024р.,

# ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

**КВАЛІФІКАЦІЙНА РОБОТА**

НА ТЕМУ: «РОЗРОБЛЕННЯ ВЕБ-ДОДАТКУ ДЛЯ ОРГАНІЗАЦІЇ РОБОЧОГО ПЛАНУ СТУДЕНТА ЗА ДОПОМОГОЮ REACT.JS»

ВИКОНАВ: ЗДОБУВАЧ ВИЩОЇ ОСВІТИ ГР. ШІД-41

ЄВГЕН ПІОНТКІВСЬКИЙ

КЕРІВНИК: КАНДИДАТ ТЕХНІЧНИХ НАУК, ДОЦЕНТ

МАКСИМ ФЕСЕНКО

2024 рік

2

***Мета роботи:*** підвищення ефективності роботи студентів та організації часу.

***Об'єкт дослідження:*** процес розробки веб-додатку.

***Предмет дослідження:*** система веб-додатку.

3

## *Основні завдання кваліфікаційної роботи*

1. Проаналізувати ринок схожих інструментів.
2. Провести аналіз актуальних веб-технологій, що стануть найкращим рішенням в реалізації веб-застосунку.
3. Встановити картину актуальних потреб та емоційного стану студентів.
4. На основі проведеного аналізу розробити прототип веб-додатку.
5. За створеним прототипом розробити веб-застосунок з урахування всіх потреб та актуальних рішень UI/UX принципів.
6. Провести тестування аналізу.
7. Задокументувати проведену роботу.

# ОСНОВНІ FRONTEND ТЕХНОЛОГІЇ

4

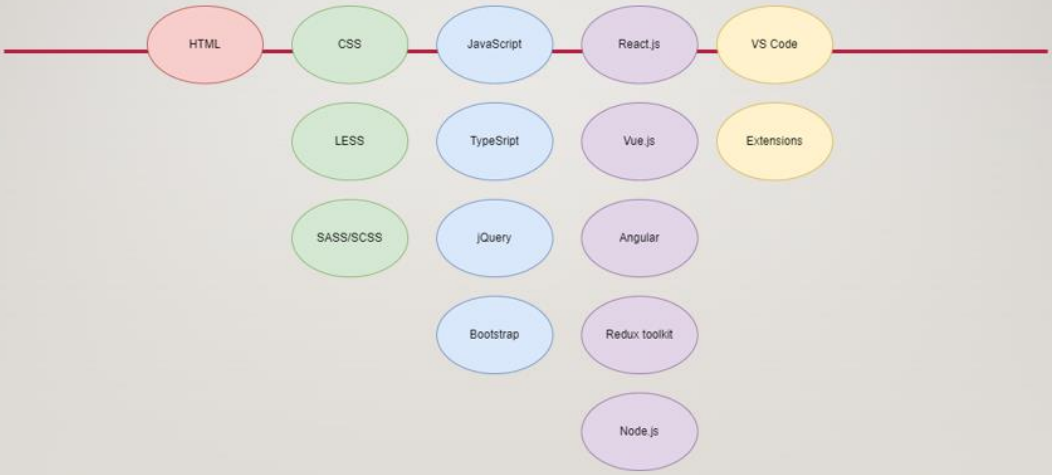



Рис. 1.1 – Існуючі frontend технології

# 5 ДОДАТОК ДЛЯ СТВОРЕННЯ МАКЕТІВ FIGMA

Рис. 1.2 – Додаток Figma

# 6 ПРОТОТИП ПРОЕКТУ

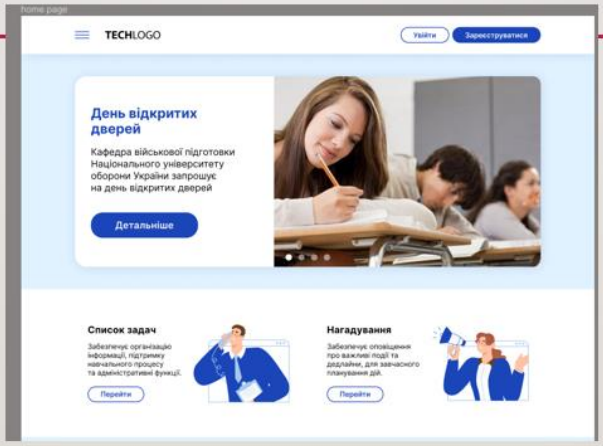



Рис. 1. 3 – Головна сторінка додатку (початок)

7

# ПРОТОТИП ПРОЕКТУ

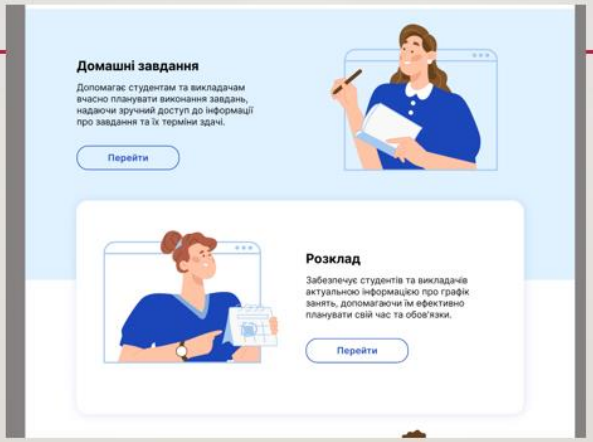



Рис. 1.4 – Головна сторінка додатку (продовження)

8

# ПРОТОТИП ПРОЕКТУ

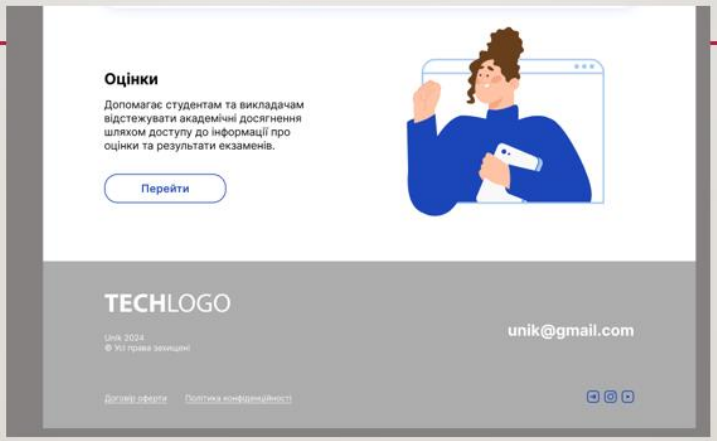



Рис. 1.5 – Кінець головної сторінки

9

# ПРОТОТИП ПРОЕКТУ


Рис. 1. 5 – Прототипи додатку та шаблони

10

# ПРОТОТИП ПРОЕКТУ

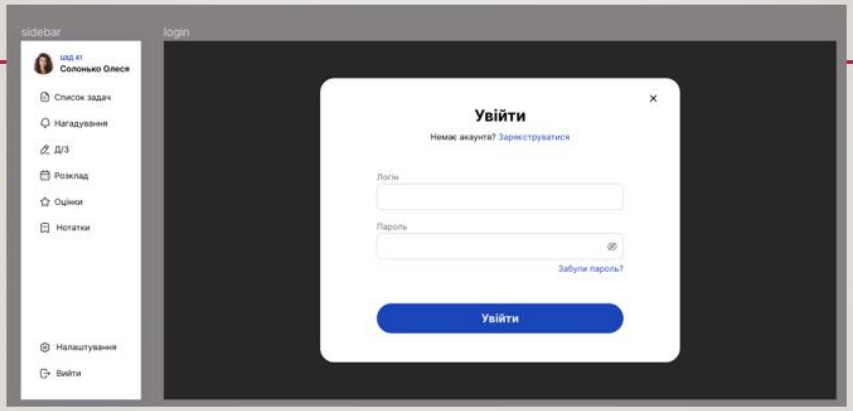



Рис. 1. 6 – Вікно входу та side-bar

# ПРОТОТИП ПРОЕКТУ

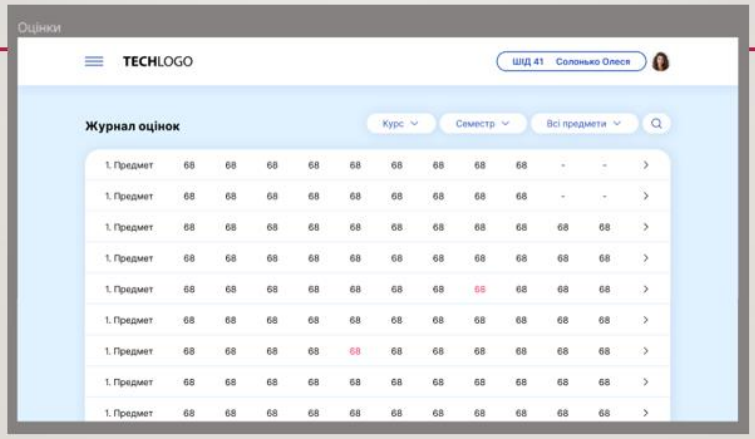



Рис. 1. 7 – Журнал оцінок



# ПРОТОТИП ПРОЕКТУ

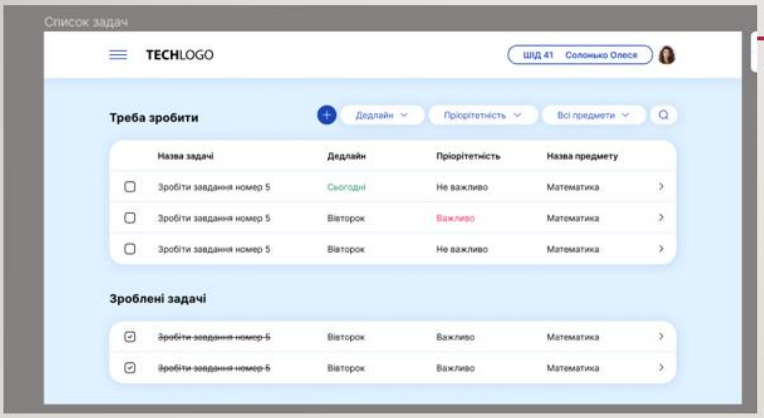



Рис. 1. 8 – ToDo List

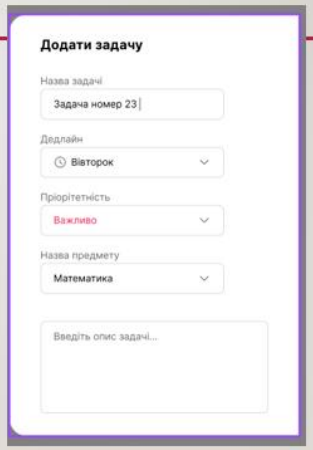



Рис. 1. 9 – Модальне вікно створення задачі в ToDo List

13

# ПРОТОТИП ПРОЕКТУ

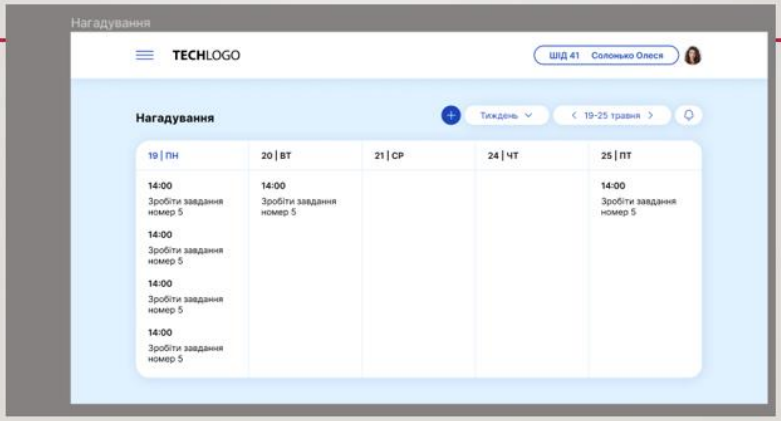



Рис. 1. 10 – Сторінка Нагадування

14

# ВИСНОВКИ

- В рамках кваліфікаційної роботи було досліджено сферу веб-розробки, її актуальні технології та рішення. Також було проведено аналіз та опитування емоційного стану студентів.

- На основі отриманих даних було проведено аналіз та розробку проекту, який має на меті підвищити продуктивність, покращити дисципліну та емоційний стан студентів. В ході розробки веб-додатку були використані наступні технології: React.js, HTML, JavaScript, CSS, Redux Toolkit та Figma.

- За допомогою вище перелічених технологій вдалось розробити прототип веб-застосунку, дотримуючись всіх правил UI/UX. На основі макету в Figma засобами веб-технологій було створено адаптивний інтерфейс веб-додатку та функціональні інструменти для організації робочого плану студента.