МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-08

Дипломний проєкт

здобувача освіти денної форми навчання
РП.08.05.000.ДП

ВОРОБЙОВА
СЕРГІЯ СЕРГІЙОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «*Інженерія програмного забезпечення*»**

Освітньо-професійна програма: «***Розробка програмного забезпечення***»

Група: **4РП-08**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту на тему:

Розробка програмного забезпечення для відстеження ступеня готовності виконання проєкту

Проєктний матеріал складається з пояснювальної записки на 76 сторінках та графічного (презентаційного) матеріалу на 15 аркушах (слайдах)

Дипломник _____ (Воробйов С.С.)

Керівник _____ (Залапін О.І.)

Консультанти:

з економічного розділу _____ (Канський М.Ю.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Краснокутська К.Г.)

Захист «30» 06 2025 р. Протокол ЕК № 3

Оцінка ЕК 4/85

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітньо-професійна програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР _____

Беркань І.В.

"12" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві (здобувачці) освіти *Воробйову Сергію Сергійовичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка програмного забезпечення для відстеження ступеня готовності виконання проєкту*

затверджена наказом по коледжу від "14" 11 2024 р. № 246

2. Термін здачі закінченого проекту _____

3. Вихідні данні до проекту (роботи) *1. Використання мови програмування для створення розмітки застосунку; 2. Використання мови програмування для налаштування стилів розмітки; 3. Використання мови програмування для веб-програмування: 4. Реалізація системи збереження даних.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити) *Огляд інструментів розробки використаних при створенні сайту; Розробка графічної частини сайту; Розробка функціональної частини сайту; Економічний розділ: Розділ охорони праці і безпеки.*

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Схема роботи сервісу Firebase. Візуальне зображення файлової структури застосунку. Скріншоти коду для реалізації додавання задач, додавання списків та функціоналу бічного меню. Скріншоти графічної частини сайту. Алгоритм реєстрації нових користувачів. Структура бази даних.

1. Консультанти по проекту (роботі), із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Залапін О.І.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

2. Дата видачі завдання 29 квітня 2025 р.

Керівник *Залапін О.І.* _____
(підпис)

Завдання прийняв до виконання *Воробйов С.С.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1.	*Постановка задач проектування*	19.05	виконано
2.	*Розробка користувацького інтерфейсу*	21.05	виконано
3.	*Реалізація інтерфейсу*	23.05	виконано
4.	*Створення файлової структури для впровадження JavaScript*	26.05	виконано
5.	*Реалізація функціоналу створення завдань*	28.05	виконано
6.	*Реалізація функціоналу додавання нових списків*	30.05	виконано
7.	*Впровадження Firebase*	03.06	виконано
8.	*Реалізація функціоналу аутентифікації користувачів*	06.06	виконано
9.	*Реалізація функціоналу управління дошками*	09.06	виконано
10.	*Написання економічного розділу*	10.06	виконано
11.	*Написання розділу з охорони праці*	11.06	виконано
12.	*Оформлення пояснювальної записки*	13.06	виконано
13.	*Оформлення презентації*	14.06	виконано
14.	*Малий захист*	16.06	виконано

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 08. 05 000. 00 ДП	Дипломний проект		
А4			РП 08. 05 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 08. 05 000. 00 ДП		
Розробив		Воробйов С.С.		16.06.25	Розробка програмного забезпечення для відстеження ступеня готовності виконання проєкту	Літ.	Аркуш 4 / Аркушів 76
Перевірив		Залапін О.І.		16.06.25		Н Д П	
Н. Контр.		Петрашова В.І.		16.06.25		ВСП "ОТФК ОНТУ" гр.4РП-08	
Затверд.		Кривченко Ю.В.		29.06			

ЗМІСТ

ВСТУП

У сучасному світі, де темпи життя прискорюються, а обсяги інформації постійно зростають, ефективне управління часом яке витрачено на виконання того чи іншого завдання є критично важливим як для окремих осіб, так і для робочих колективів. Програмні засоби планування допомагають структурувати розпорядок дня, оптимізувати виконання задач і підвищити продуктивність не тільки при виконанні робочих завдань, а й при вирішенні побутових задач.

Розробка таких програм передбачає врахування ряду ключових аспектів. Передусім, зручності користувацького інтерфейсу, який не створюватиме додаткових перешкод у користуванні. Гнучкості структури даних. Можливість збереження інформації, яку вводить користувач та подальше її відображення для використання. Забезпечення стабільності, безпеки та швидкої взаємодії між клієнтом та базою даних для оперативного оновлення даних також є невід'ємним елементом для програм, які надають можливість відстежувати ступінь готовності виконання завдань.

Створення ефективного застосунку такого типу сприяє підвищенню загальної продуктивності користувача, зменшенню кількості помилок в організації роботи. За допомогою подібних інструментів користувач отримує змогу своєчасно виявляти критичні моменти у виконанні роботи, оперативно реагувати на зміни пріоритетів та мінімізувати витрати часу на задачі, які не є зараз важливими, або терміновими до виконання. Це в свою чергу сприяє зниженню ризиків, пов'язаних з людським фактором та загальному покращенню управління часом.

У рамках дипломної роботи розглянуто процес проектування та реалізації програмного забезпечення для відстеження ступеня готовності виконання проєкту. Обґрунтовано вибір технологій які використовувалися у розробці, з урахуванням їхньої сумісності, надійності та можливостей масштабування. Створено зручний інтерфейс користувача та реалізовано необхідну логіку для введення, збереження та обробки даних, оскільки ці аспекти є ключовим аспектом для функціонування програм такого типу.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Огляд інструментів розробки використаних при створенні сайту

Нижче буде наведено опис обраних інструментів та технологій, які використовувались під час розробки програмного продукту. До них входять такі мови програмування, як: HTML, CSS та JavaScript.

Під час розробки програмного продукту були використані наступні інструменти для реалізації задачі:

- Visual Studio Code – використовувався у якості основної середи розробки.
- HTML – для створення розмітки на сайті.
- CSS – для стилізації і розміщення елементів у потрібних місцях.
- JavaScript – для реалізації функціональної частини застосунку.
- Firebase – для реалізації бази даних.

Нижче буде наведено опис згаданих інструментів та технологій.

1.1.1 Опис середи розробки Visual Studio Code

У світі сучасних інструментів розробки програмного забезпечення Visual Studio Code займає особливе місце, ставши справжнім стандартом для мільйонів розробників по всьому світу. Цей редактор коду, створений Microsoft у 2015 році, пройшов значний шлях від просто текстового редактора до потужної інтегрованої середовища розробки, що поєднує в собі легкість використання і простоту інтерфейсу з дійсно широким вибором доступного функціоналу.

Популярність Visual Studio Code серед розробників різних рівнів кваліфікації та спеціалізацій обумовлена цілим рядом унікальних характеристик. На відміну від традиційних IDE, які часто відрізняються громіздкістю та високими вимогами до системних ресурсів, Visual Studio Code зберігає вражаючу швидкодію навіть на малопотужних комп’ютерах. Це досягається завдяки оптимізованій архітектурі та використанню сучасних веб-технологій у його основі.

Однією з ключових переваг Visual Studio Code є його гнучкість та адаптивність. Система розширень зволяє трансформувати базовий редактор у спеціалізоване середовище розробки для практично будь-якої технології чи мови програмування. На сьогоднішній день для Visual Studio Code доступно понад 20 тисяч розширень для завантаження, що охоплюють всі аспекти сучасної розробки, від веб-дизайну до системного програмування, від роботи з базами даних до розробки моделей штучного інтелекту.

Важливою частиною Visual Studio code, як середи для розробки є система IntelliSence, яка забезпечує функціонал автодоповнення коду. На відміну від простих підказок, IntelliSence аналізує контекст, надаючи релевантні пропозиції на основі типів даних, структур проектів та навіть документації API. Це значно прискорює процес написання коду та знижує ймовірність помилок, особливо при роботі з незнайомими бібліотеками чи фрейморками.

Для команд розробників, які працюють над одним проектом разом дуже важливою та зручною є інтеграція з системами контролю версій, зокрема Git. Visual Studio Code дозволяє виконувати більшість операцій з управління версіями додатку, який розробляється безпосередньо в інтерфейсі редактора: переглядати зміни, переглядати та створювати коміти та керувати гілками.

Функціонал відповідальний за налагодження коду у Visual Studio Code забезпечує швидку та зрозумілу взаємодію з кодом для виявлення помилок. Редактор підтримує повноцінний цикл налагодження для більшості популярних мов програмування, включаючи встановлення точок зупину, покрокове виконання коду, оцінку виразів у режимі реального часу та аналіз стеку викликів. Особливо варто відзначити підтримку віддаленого налагодження, яка дозволяє працювати з кодом, що виконується на серверах.

Також Visual Studio Code включає у себе потужні засоби для розробки з контейнерами (Docker), хмарними сервісами (Azure), базами даних та навіть машинним навчанням. Вбудований термінал з підтримкою кількох сесій, інтеграція з системами збірки та інструментами тестування роблять його універсальним інструментом для розробки програм.

Окрім всього, що було зазначено, VS code має активну спільноту, яка займається розширенням його можливостей. Це забезпечується за рахунок відкритої архітектури та чіткої документації, використовуючи ці інструменти розробники можуть створювати власні розширення, адаптуючи середу під власні потреби.

Зазначаючи все вище сказане можна дійти висновку, що VS code це просте, але дуже технологічне рішення для створення веб-додатків.

1.1.2 Опис мови розмітки HTML

HTML (hypertext markup language) – це мова, яка лежить в основі будь-якої веб-сторінки. Вона не є мовою програмування у класичному розумінні, а скоріше мовою розмітки, призначеною для структурування контенту на сторінці. При відкритті будь якого сайту у браузері, від простого блогу до складного веб-додатку – перше, що завантажується та обробляється, це саме HTML-код. Він визначає, де має бути заголовок, де розміщений текст, де знаходиться зображення чи відео, а де – кнопка або форма для введення даних.

HTML-код будується на використанні спеціальних позначок – тегів. Теги – це інструкція для браузера, які вказують, як саме відображати той чи інший елемент сторінки. Наприклад тег h1 позначає головний заголовок, тег p абзац тексту, img зображення, а посилання на іншу сторінку. Ці теги можуть містити атрибути, які уточнюють їх поведінку або зовнішній вигляд. Наприклад, тег img зазвичай має атрибут src, що вказує, звідки брати зображення, а також alt, який містить опис для доступності.

HTML виник у 1990-х роках завдяки Тіму Бернерсу-Лі. Спочатку він був дуже простим і дозволяв лише базове форматування тексту та створення посилань. Однак з часом мова розвивалася, з’явилися нові версії (HTML, XHTML, HTML5), і сьогодні вона надає розробникам значно більше можливостей. Сучасний HTML5 дозволяє вбудовувати відео та аудіо без додаткових плагінів, створювати інтерактивні форми, використовувати графіку за допомогою тегів csnvas і svg, а також забезпечує підтримку локального сховища даних.

Одна з ключових переваг HTML це його універсальність. Він працює на будь-якому пристрої – від комп’ютера до смартфона, і в будь-якому сучасному браузері.

Підводячи підсумок HTML – це основний інструмент для створення розмітки сайту, який відповідає за визначення положення тих чи інших елементів, які відображаються на сайті.

1.1.3 Мова оформлення та стилізації CSS

CSS (cascading style sheets) – також не є мовою програмування, у звичному розумінні. Це мова стилів, яка надає HTML-розмітці візуального оформлення, до сторінки можна додавати кольори, налаштовувати шрифти, задавати розміри елементів, додавати анімації. Ця мова з’явилася у 1996 році як відповідь на зростаючу потребу у відокремленні змісту сторінки від її зовнішнього вигляду.

Сутність CSS полягає у використання правил стилізації, які дозволяють керувати практично всіма аспектами візуальної складової веб-сторінки. Від найпростіших властивостей: кольору тексту чи розміру шрифту, до складних тривимірних об’єктів та плавних анімацій.

Сучасний CSS еволюціонував у потужний та зручний інструмент, що включає такі можливості:

1. Гнучке керування розміщенням елементів завдяки технологіям flexbox та grid, які дозволяють створювати складні та адаптивні сторінки.

2. Медіа-запити (media queries), що роблять сайти повністю адаптивними – вони автоматично підлаштовуються під розмір екрану пристрою, будь то широкоформатний монітор чи телефон.

3. CSS-змінні, які дозволяють створювати динамічні теми оформлення та централізовано керувати стилями через JavaScript.

4. Складні анімації та переходи, що реалізуються без необхідності використання додаткових плагінів чи бібліотек.

5. Фільтри, ефекти накладання (blend modes) та інші візуальні ефекти.

Важливою особливістю сучасного CSS є його система класів – можливість давати їм зрозумілі назви, що значно полегшує підтримку коду.

Таким чином, CSS являється основним інструментом для налаштування зовнішнього виду додатку.

1.1.4 Мова програмування JavaScript

JavaScript виник у 1995 році як невелика скриптова мова для додавання інтерактивності веб-сторінкам у браузері Netscape Navigator. Його створив Брендан Айк. Спочатку JavaScript задумувався, як простий інструмент валідації форм та створення елементарних анімацій, але згодом він перетворився на універсальну мову програмування, що лежить в основі веб-програмування.

Сутність JavaScript полягає у його здатності динамічно керувати об'єктною моделлю документа, вона ж DOM. Це дозволяє змінювати вміст сторінки, її стилі та поведінку в реальному часі без необхідності її перезавантаження. Така особливість стала справжньою революцією для веб-програмування, перетворивши статичні сторінки на повноцінні інтерактивні додатки.

Архітектура JavaScript базується на кількох ключових принципах, які роблять його унікальним серед інших мов програмування. Подієва модель JavaScript побудована на концепції циклу подій (event loop), що дозволяє ефективно обробляти асинхронні операції, ця особливість буде активно використовуватись під час написання програмного продукту "Task Manager", особливо під час взаємодії додатку з базою даних. Крім того, функції в JavaScript є об'єктами першого класу – їх можна передавати як аргументи, повертати з інших функцій і зберігати у змінних, що також активно буде використовуватись при подальшій розробці.

Синтаксис JavaScript поєднує в собі мови C та Java, але має свої унікальні особливості. Мова слабко типізована, з динамічною типізацією, що спрощує написання коду на початкових етапах, але може ускладнювати його підтримку у великих проектах. Для вирішення цієї проблеми були створені надбудови над JavaScript, такі як TypeScript, які додають статичну типізацію та інші можливості для розробки великих систем. Останні версії мови (ES 6) додали такі важливі функції, як класи, проміси, async / await, модулі та багато іншого, значно

підвищивши виразність і читабельність коду. Зокрема async / await також буде використовуватись у багатьох моментах розробки

Сукупність всіх цих факторів робить JavaScript зручною та багатофункціональною мовою програмування, для реалізації такої задачі, як створення веб-застосунку по типу task Manager.

1.1.5 Огляд Firebase, як рішення для збереження даних

Firebase це хмарний сервіс від Google. Його історія почалася ще у 2011 році, у якості стартапу, який згодом був куплений компанією Google і перетворився на один із найпопулярніших інструментів backend-розробки. Головна особливість Firebase полягає в тому, що він дозволяє розробникам зосередитися на створенні якісного клієнтського додатку, повністю позбавляючи їх від необхідності писати та підтримувати власний серверний код.

Основним компонентом Firebase є Realtime Database, вона представляє собою NoSQL базу даних, яка працює за принципом зберігання даних у форматі JSON та їх автоматичної синхронізації між усіма підключеними клієнтами в режимі реального часу. Однією з ключових переваг Realtime Database є її простота. При його використанні не потрібно налаштовувати складні серверні API для обміну даними – вся логіка синхронізації вже вбудована в платформу.

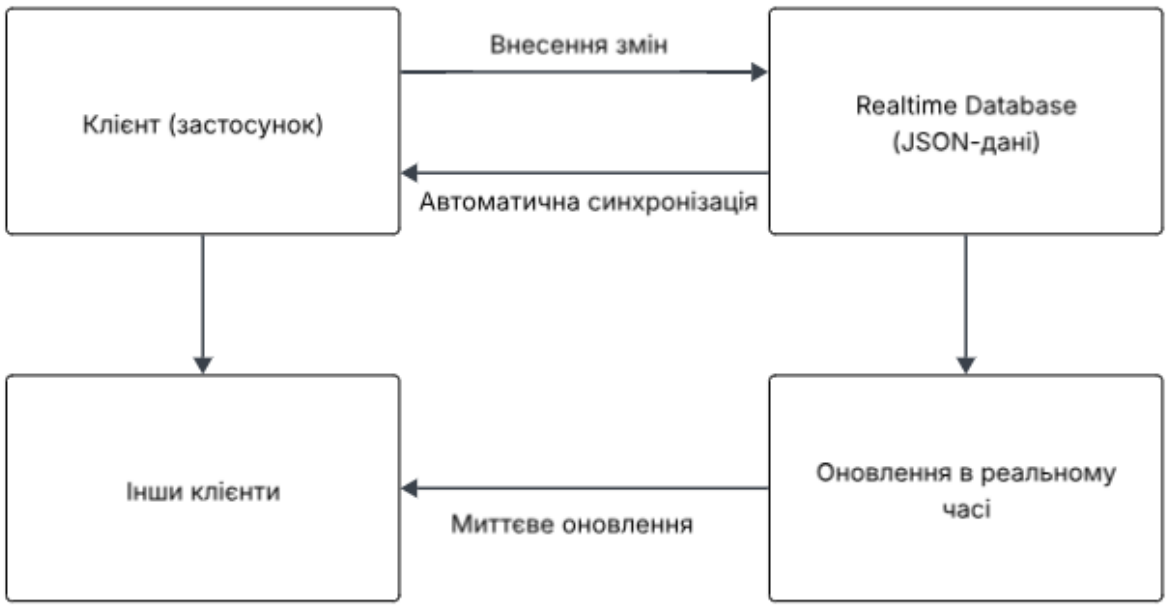


Рисунок 1.1 Схема роботи Realtime Database

Архітектура бази даних побудована на деревоподібній структурі, де кожен вузол може містити як значення, так і інші вузли. Така організація дає гнучкість у проектуванні структури даних, але водночас вимагає ретельного планування, оскільки надмірно глибокі або складно організовані структури можуть ускладнити роботу з даними.

Важливою особливістю Realtime Database є підтримка офлайн роботи. Коли пристрій втрачає з'єднання з інтернетом, база даних продовжує працювати з локальною копією даних, а при відновленні зв'язку автоматично синхронізує всі зміни з сервером.

Безпека даних у Realtime Database забезпечується за допомогою спеціальних правил, які можна налаштувати прямо в консолі Firebase. Ці правила дозволяють контролювати, хто може читати та записувати дані в різні частини бази. Наприклад, можна дозволити всім користувачам читати публічні дані, але обмежити запис тільки для авторизованих користувачів або навіть для конкретних ролей. Це дозволяє створювати складні системи прав доступу без необхідності прописувати додатковий серверний код.

У Firebase є система аутентифікації, яка підтримує всі сучасні методи вводу, включаючи електронну пошту, соціальні мережі, а також анонімний доступ. Це дозволяє легко додавати авторизацію у додатки без необхідності реалізовувати власні системи перевірки користувачів.

Однією з проблем Realtime Database є обмежені підтримка складних запитів. На відміну від традиційних SQL-баз, де можна виконувати складні об'єднання таблиць, у Realtime Database запити обмежені фільтрацією та сортуванням по одному полю. Це означає, що для складніших сценаріїв роботи з даними може знадобитися або замінити структуру бази, або використовувати додаткові індекси.

У висновку можна сказати, що Firebase Database це зручний інструмент, який ідеально підходить для створення інтерактивних і динамічних додатків з оновленням внесених змін у реальному часі. Його простота, автоматична синхронізація та підтримка офлайн роботи, що забезпечує коректну роботу

додатку навіть при відсутньому інтернет з’єднанні. Все це робить Realtime Database ефективним і простим рішенням для реалізації збереження даних застосунку.

1.2 Розробка графічної частини сайту

Графічна частина веб-додатку «Task Manager» реалізована за допомогою таких інструментів, як: HTML та CSS, що забезпечує зручність користування та адаптивність інтерфейсу. Основна увага приділяється взаємодії користувача з елементами та логічному розподілу функціональних зон. Для візуального оформлення використано стилі, які надають елементам інтерфейсу чіткої структури, зручної навігації та візуального виділення активних станів. Нижче наведено детальний опис рішень, які були застосовані для реалізації графічної частини застосунку.

1.2.1 Створення HTML розмітки

Основним і єдиним файлом, що відповідає за усю розмітку на сторінці є index.html. Він забезпечує відображення всіх необхідних елементів, а саме: верхня навігаційна панель, бокове меню, та основний робочий простір на якому відображаються списки задач.

У секції head> за допомогою тегу link виконується підключення усіх необхідних CSS-файлів, а саме:

1. Board.css – відповідає за основну робочу область, включаючи розташування колонок та їх адаптивність.

2. Cards.css – містить стилі для карток, які всередині містять завдання, включаючи їх інтерактивність та анімації.

3. Columns.css – визначає оформлення колонок, всередині яких містяться картки.

4. Header.css – задає стилі для верхньої панелі навігації.

5. Main.css – включає базові налаштування стилів.

6. Menu.css – відповідає за зовнішній вигляд контекстних меню та їх анімації.

7. Modals.css – відповідає за оформлення модальних вікон, включаючи форми та кнопки, які знаходять всередині.

8. Sidebar.css – відповідає за оформлення бічного меню з його елементами, які знаходяться всередині.

Далі за допомогою тегу script відбувається підключення сервісів Firebase, за допомогою яких у подальшому буде реалізовано процес реєстрації нових користувачів або аутентифікації тих, хто вже має створений акаунт, окрім цього забезпечується також взаємодія між додатком та базою даних для збереження інформації про створені дошки, списки задач, тощо.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Task Manager</title>

    <link rel="stylesheet" href="./css/board.css">
    <link rel="stylesheet" href="./css/cards.css">
    <link rel="stylesheet" href="./css/columns.css">
    <link rel="stylesheet" href="./css/header.css">
    <link rel="stylesheet" href="./css/main.css">
    <link rel="stylesheet" href="./css/menu.css">
    <link rel="stylesheet" href="./css/modals.css">
    <link rel="stylesheet" href="./css/sidebar.css">

    <script src="https://www.gstatic.com/firebasejs/9.6.0/firebase-app-compat.js"></script>
    <script src="https://www.gstatic.com/firebasejs/9.6.0/firebase-auth-compat.js"></script>
    <script src="https://www.gstatic.com/firebasejs/9.6.0/firebase-database-compat.js"></script>
</head>
```

Рисунок 1.2 Зміст тегу head

Уся розмітка зосереджена всередині тегу body. Вона поділена на основні елементи, такі як:

1. Верхня навігаційна панель.

2. Бічне меню.

3. Основний робочий простір.

4. Модальне вікно для налаштування картки.

5. Модальне вікно для налаштування колонок.

6. Вікно для реєстрації нових користувачів та авторизації тих, хто має акаунт.

Самий же тег body, якщо можна так сказати, поділений на дві частини. Перша частина загорнута у контейнер з класом app-content, вона містить увесь

функціональний інтерфейс, і за замовчуванням вона скрита для неавторизованих користувачів. Друга частина це вікно для авторизації користувачів.

Розмітка структури інтерфейсу починається з фіксованого елементу header, що включає у себе навігаційну панель nav, яка розділена на дві частини. Ліва частина містить кнопку активації бічного меню та назву застосунку, тоді як права включає в себе дві кнопки: кнопку доступу до профілю користувача, та створення нової колонки.

```
<div class="header">
    <nav class="navbar">
        <div class="navbar-side">
            <span class="open-btn"><img class="side-menu-png" src="./png/sidemenu.png" alt=""></span>
            <a class="navbar-brand">Task Manager</a>
        </div>
        <div class="navbar-btn">
            <button class="auth-btn">Авторизуватися</button>
            <button class="profile-btn" style="display: none;">Профіль</button>
            <button class="new-column">Нова колонка</button>
        </div>
    </nav>
</div>
```

Рисунок 1.3 Зміст контейнеру header

Далі у структурі розміщено елемент side-menu, який реалізований за допомогою негативного зміщення, яке задано властивостями CSS, що дозволяє приховувати його за межами вікна перегляду до моменту активації за допомогою відповідної кнопки. Структурно меню поділене на три блоки:

1. Заголовковий блок, який має клас sidemenu-header, включає у себе назву бічного меню, а також кнопку, яка дозволяє його закривати.

2. Основний блок, який має клас column-list. Представляє собою основну функціональну область з динамічним списком доступних робочих дошок. Зміст цього блоку регулюється за допомогою JavaScript. Дошки можна додавати, видаляти, та змінювати їхні назви.

3. Нижній блок, який має клас sidemenu-bottom. Включає у себе кнопку для створення нових дошок.

```
<div class="side-menu" id="sideMenu">
    <div class="sidemenu-header">
        <h3 class="sidemenu-title">Управління дошками</h3>
        <div class="close-btn">X</div>
    </div>

    <div class="column-list"></div>

    <div class="sidemenu-bottom">
        <button class="new-board">Створити нову дошку</button>
    </div>
</div>
```

Рисунок 1.4 Зміст контейнеру side-menu

Центральним функціональним контейнером сторінки є блок board, що виконує роль головної робочої області, всередині можуть міститися колонки. Він складається з:

1. Заголовок, який містить поле з назвою колонки, яке можна змінювати, а також кнопку, при натисненні на яку з'являється функціональне меню, всередині якого три доступні функції: перша – «Налаштування колонки», при натисненні з'являється модальне вікно, за допомогою якого користувач може налаштувати колонку. Друга – «Видалити колонку» видаляє колонку з дошки. І третя – «Додати картку» при натисненні додає нову картку до колонки.

2. Основна частина, це контейнер, у якого клас card-list. На початку він пустий, проте до нього можна додавати картки, всередині яких міститься текст, який ввів користувач. Ця частина є динамічною і регулюється за допомогою JavaScript.

3. Нижня частина, яка складається з двох кнопок, перша служить для додавання нових карток, друга відповідальна за переміщення колонок всередині дошок.

```
<div class="board">
    <div class="column">
        <div class="column-header">
            <h3 class="column-title" contenteditable="true">Потрібно зробити</h3>
            <button class="menu-button" aria-expanded="false" aria-controls="list">
                <img class="menu-png" src="./png/menu.png" alt="">
            </button>
            <ul class="menu" id="list">
                <div class="menu-header">
                    <h3 class="menu-title">Дії з колонкою</h4>
                    <button class="menu-clouse">X</button>
                </div>
                <button class="menu-item item1">Налаштування колонки</button>
                <button class="menu-item item3">Видалити колонку</button>
                <button class="menu-item item4">Додати картку</button>
            </ul>
        </div>
        <div class="card-list"></div>
        <div class="button-container">
            <button class="add-card">+ Додати картку</button>
            <button class="drag-button">
                <img class="drag-button-png" src="./png/movecolumn.png" alt="" draggable="false">
            </button>
        </div>
    </div>
```

Рисунок 1.5 Зміст контейнеру board

Наступним структурним блоком є система модальних вікон. Вона включає три окремі функціональні модулі:

1. Контейнер з класом modal1, який представляє собою вікно для редагування карток, відкривається натисненням на картку.

2. Контейнер з класом modal2, який представляє собою вікно для редагування колонок, відкривається за допомогою кнопки, яка розміщена у функціональному меню.

3. Контейнер з класом modal-auth, який представляє собою вікно для аутентифікації користувача.

Кожне модальне вікно містить всередині заголовок, який вказує на призначення цього меню. Основну функціональну частину, всередині якої знаходяться поля для введення, або редагування відповідної інформації. А також нижню частину всередині якої знаходяться кнопки для збереження змін, або для видалення обраного елементу.

```
<div class="modal modal1">
    <div class="modal-main">
        <div class="modal-header">
            <h2 class="modal-title">Налаштування картки</h2>
            <button class="modal-close">X</button>
        </div>
        <div class="modal-content">
            <div class="modal-content-title">
                <label>Текст:</label>
                <textarea class="modal-textarea"></textarea>
            </div>
            <div class="modal-content-title">
                <label>Колір фону:</label>
                <input type="color" class="modal-color">
            </div>
            <div class="modal-content-title">
                <label>Замітки:</label>
                <input type="text" class="modal-tags">
            </div>
            <div class="modal-content-title">
                <label>Термін виконання:</label>
                <input type="date" class="modal-deadline">
            </div>
            <div class="modal-content-title">
                <label>Статус виконання:</label>
                <select class="modal-status">
                    <option value="not_done">Не виконано</option>
                    <option value="done">Виконано</option>
                </select>
            </div>
        </div>
        <div class="modal-button-container">
            <button class="modal-save">Зберігти</button>
            <button class="modal-delete">Видалити</button>
```

Рисунок 1.6 Зміст модального вікна для налаштування картки

1.2.2 Додавання CSS стилів

Система стилізації веб-додатку реалізована на основі CSS та побудована відповідно до модульної архітектури, що передбачає логічне розділення

відповідальності між окремими файлами стилів. Кожен CSS файл відповідає за оформлення конкретного блоку інтерфейсу, що значно спрощує подальший процес зміни необхідних стилів, або додавання нових.

Файл board.css містить стилі основної робочої зони – дошки, де розміщено колонки із завданнями. Для початкового розташування елементів використано властивість display:flex, що забезпечує вирівнювання елементів при стандартних позиціях колонок. Використана далі властивість flex-wrap: wrap дозволяє переносити елементи на новий рядок, якщо ті не влізають у один рядок. Властивість align-content: center вирівнює колонки по вертикалі, justify-content: center в свою чергу вирівнює по горизонталі. Завдяки цим двом властивостям колонки, які щойно були створені розміщуються по центру екрану. Position: relative використовується для того, щоб при активації переміщення колонок не ламалася основна структура. Наостанок задається висота дошки з використанням heigh: calc(100vh - 20px), де 100 vh – 100% від висоти дошки, а 20px це фіксоване значення, яке віднімається від загальної висоти. Грубо кажучи це формула яка відповідає за розрахунок висоти дошки, робиться це для коректної роботи опції перетягування колонок.

```
.board
{
    display: flex;
    flex-wrap: wrap;
    align-content: center;
    justify-content: center;
    align-items: flex-start;
    padding-right: 0;
    position: relative;
    height: calc(100vh - 20px);
}
```

Рисунок 1.7 Налаштування стилів для дошки

Файл cards.css відповідає за оформлення зовнішнього виду карток разом з їх вмістом. Кожна картка розміщується в межах певної колонки та має індивідуальне оформлення, що включає: колір фону, рамку, тіні, ефекти при наведенні та активному стані. Колір фону забезпечується властивістю background-color і за замовчуванням вона є білою, але її колір можна змінити через

використання модального вікна з відповідним налаштуванням. Заокруглення кутів забезпечується рядком border-radius: 5px. Також присутні ефекти, які з’являються при наведенні курсора на картку, наприклад transform: translate відповідає за візуальний підйом картки над іншими, при цьому з’являється тінь, для кращого візуального ефекту. Оскільки колонки мають обмеження по максимальній висоті, то при її досягненні з’являється полоса прокрутки, щоб був доступ до усіх карток, які є всередині колонки, вона також стилізована, щоб при появі не вибивалася з зовнішнього вигляду. Для її стилізації використовувалися ::-webkit-scrollbar, ::webkit-scrollbar-thumb, які вказували на смугу, а всередині вже налаштовувались кольори, та розмір з заокругленням кутів.

Рисунок 1.8 Зовнішній вигляд картки

Стилі для окремих колонок реалізовано у файлі columns.css. Кожна колонка має оформлення, яке включає фоновий градієнт, заокруглені кути, та внутрішні відступи, які задаються параметром padding, робиться це для того, щоб елементи які знаходяться всередині колонки не прилипали до країв. Для кожної колонки передбачено панель управління кнопками, стилізованими відповідно до загального стилю додатку.

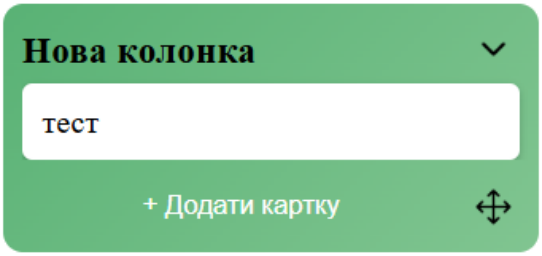


Рисунок 1.9 Зовнішній вигляд колонки з карткою всередині

Файл header.css містить стилі для верхньої навігаційної панелі, яка виконує функції глобального керування застосунком. Основною особливістю є фіксоване позиціонування панелі у верхній частині вікна браузера, реалізоване за допомогою властивості position: fixed, що дозволяє зберігати доступ до елементів управління незалежно від положення користувача на сторінці. Для візуального виокремлення header використано фон із градієнтним переходом між двома

відтінками зеленого кольору. Інтерактивні елементи навігаційної панелі, зокрема кнопки авторизації, відкриття меню, доступу до профілю стилізовані із застосуванням псевдо-елементів ::before та ::after, що забезпечують візуальні ефекти наведення, такі, як контурні підсвічування, зміну кольору меж або тіней. Для досягнення плавності переходів між станами активності використовуються властивості transition із заданими часовими параметрами для додавання плавності змінам. Також додана тінь, що візуально відділяє панель від основного контенту сторінки та створює відчуття багатошаровості.

Рисунок 1.10 Зовнішній вигляд верхньої панелі

Файл main.css містить основні стилі, які стосуються загального робочого простору, наприклад тут прописано, що на початку основний контент не відображається, а видно лише вікно для реєстрації користувача, але це за умови, якщо не виконаний вхід до акаунту. Забезпечується це властивістю display: none.

Файл menu.css містить стилі для контекстних меню, які активуються при натисканні на кнопку з трьома крапками, яка знаходиться в заголовках колонок. Меню позиціонується відносно батьківського елемента за допомогою position: absolute та з високим значенням z-index, що гарантує його появу поверх інших елементів, що забезпечить доступність до його кнопок. Анімацію появи реалізовано за допомогою @keyframes із поступовим збільшенням прозорості та масштабування. Кожен пункт меню оформлено зі збереженням достатнього внутрішнього простору, який забезпечує параметр padding, чіткою межею між пунктами, яка забезпечується за допомогою параметру border-bottom. Також реалізовано зміну кольору разом з підняттям елементу вгору при наведенні. Властивість pointer-events використовується для керування активністю пунктів залежно від стану елементу.

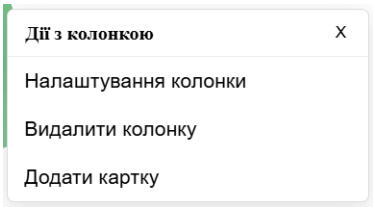


Рисунок 1.11 Зовнішній вигляд меню

Файл modals.css охоплює стилізацію модальних вікон, включаючи вікна для редагування карток, налаштування колонок і форму авторизації. Напівпрозорий фон забезпечується властивістю background-color: rgba, у якій зазначено необхідний колір. Розміщення вікна по центу досягається за допомогою поєднання властивостей flexbox, таких, як: display: flex, justify-content: center, align-items: center, та абсолютного позиціонування. Всі поля введення мають стилізовані рамки, які змінюють колір у фокусі. Кнопки збереження змін та видалення елементу мають візуальне розділення через градієнтну заливку, тінь та зміну кольору при наведенні. Це дозволяє легко розрізняти функціональні дії навіть без додаткового текстового опису.

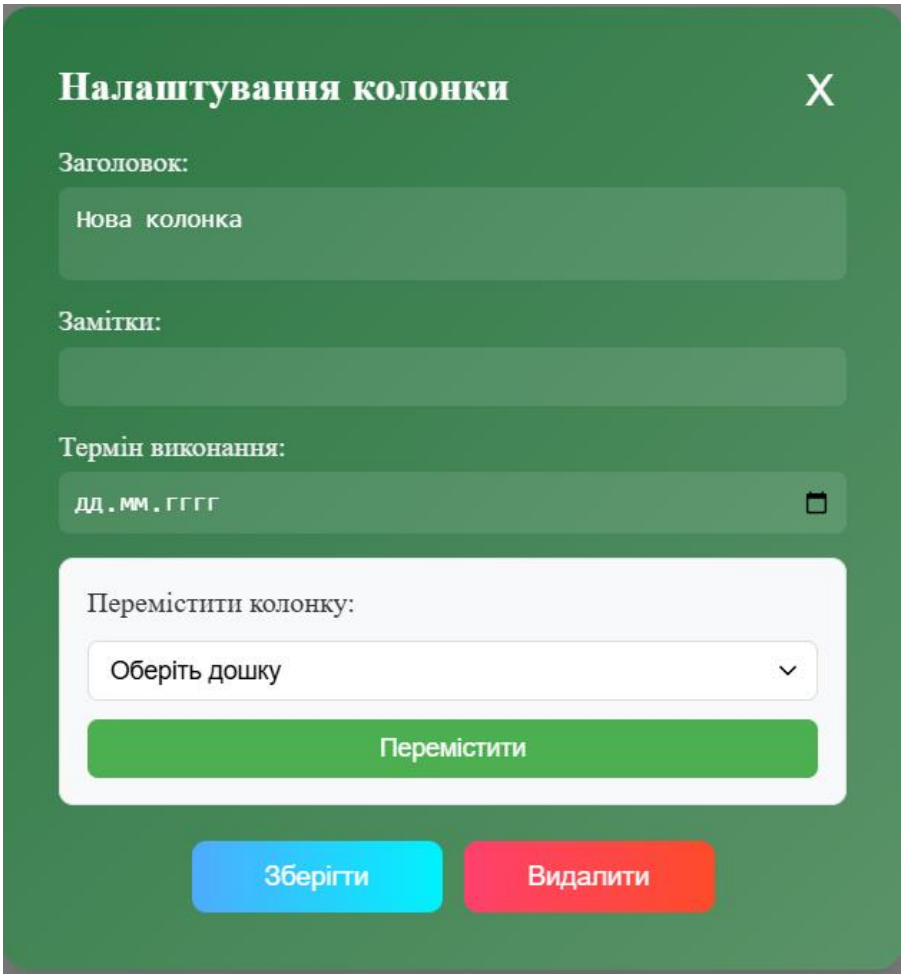


Рисунок 1.12 Зовнішній вигляд модального вікна для налаштування колонки

Файл sidebar.css відповідає за оформлення бічного меню. Реалізовано приховане положення за межами екрана з плавною анімацією появи меню, через зміну властивості left з -300px до 0. Елементи меню мають стандартні ефекти наведення, індикатор активної дошки та стилізовані іконки для перейменування

та видалення колонок. Всі переходи виконуються плавно, із затримкою 200-300мс, що сприяє приємнішій взаємодії з інтерфейсом.

Окрім цього для кнопок, які відповідають за перейменування та видалення дошок зроблено наступне управління прозорістю: для дошки, яка зараз активна виконується легке підсвітлення, для більш простого орієнтування, та для кнопок прибирається прозорість (властивість opacity), щоб їх було видно. А при неведені курсору на іншу дошку відбуваються такі самі дії, легке підсвічування та поява кнопок.

Рисунок 1.13 Зовнішній вигляд бокової панелі

1.3 Розробка функціональної частини сайту

Вся логіка для працездатності додатку написана на мові програмування JavaScript. Весь код розбитий на окремі файли, які в свою чергу розбиті на окремі папки, що утворює розгалужену систему файлів. Це забезпечує чітку модульну систему у якій легко орієнтуватись, та за потреби додавати нові функціональні модулі, або оновлювати вже існуючі для реалізації нового функціоналу.

1.3.1 Розробка функціоналу додавання карток до списку задач, та можливість їх редагування

Для створення нових карток було розроблено наступний алгоритм:

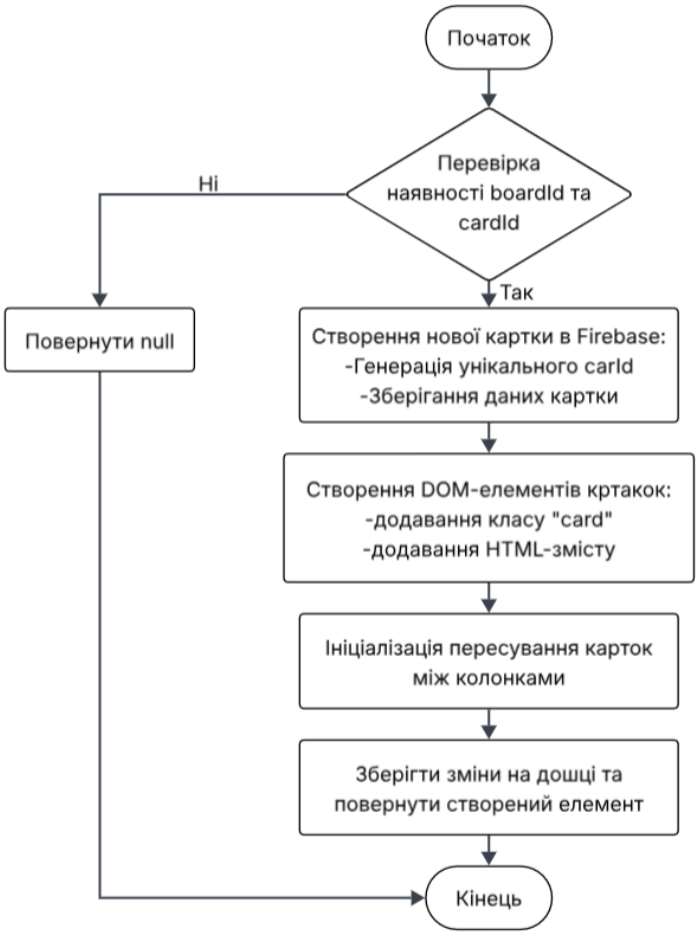


Рисунок 1.14 Розробка алгоритму для створення карток

Функціонал для додавання нових карток у колонки прописаний у файлі addcard. Код, який відповідальний за це знаходиться всередині функції createCard. Спочатку функція отримує ідентифікатор активної дошки (const boardId), потім отримується ідентифікатор колонки (columnId), до якої користувач хоче додати нову картку. Після цього виконується перевірка отриманих даних, через використання оператору з умовою if(!board || !columnId) і якщо будь-який з цих ідентифікаторів не буде знайдений, то функція припинить своє виконання. У випадку ж, коли усі необхідні ідентифікаторі були знайдені виконується наступний етап – створення нового запису у базу даних за допомогою методу push, з подальшою генерацією унікального id для картки. Після цього у базі даних створюється новий об’єкт, всередині якого зберігаються дані про картку, а саме:

текст картки, її колір, замітки до неї, термін виконання і статус виконання. По завершенню запису даних формується новий елемент картки – це div із класом card, всередині якого зберігається введений текст.

```
export async function createCard(column, cardText = "") {
    const boardId = window.boardManager?.currentBoardId;
    const columnId = column.dataset.columnId;

    if (!boardId || !columnId) {
        return null;
    }

    const cardRef = database.ref(`boards/${boardId}/columns/${columnId}/cards`).push();
    const cardId = cardRef.key;

    await cardRef.set({
        text: cardText.trim(),
        color: "#ffffff",
        tags: "",
        deadline: "",
        status: "not_done",
        createdAt: new Date().toISOString()
    });

    const card = document.createElement("div");
    card.classList.add("card");
    card.dataset.cardId = cardId;
    card.innerHTML = `
        <p class="card-text">${cardText}</p>
    `;

    initDragForCard(card);

    if (window.boardManager) {
        await window.boardManager.saveCurrentBoard();
    }

    return card;
}
```

Рисунок 1.15 Код функції addCard

Для можливості редагування карток було створено модальне вікно, яке активується при натисненні на картку. Всередині якого можна змінювати текст картки, колір, замітки, термін виконання задачі, а також її статус.

На початковому етапі код отримує об’єкт конфігураційного файлу Firebase, що забезпечує доступ до зберігання даних та їх оновлення у реальному часі. Функція initCardModal виступає основним елементом коду і виконує ініціалізацію всіх необхідних компонентів при виклику функції.

Першим кроком у тілі функції відбувається пошук елементів модального вікна за допомогою методу querySelector. Якщо відповідний елемент не був знайдений, то функція припиняє своє виконання, але при знаходженні потрібного елементу функція продовжується, і починається пошук внутрішніх елементів модального вікна для налаштування картки: поля введення тексту, зміни кольору,

зміна статусу виконання, а також кнопки, які відповідають за збереження змін, видалення картки або закриття модального вікна.

```
const modal = document.querySelector(".modal1");
if (!modal) {
    return;
}

const modalTitle = modal.querySelector(".modal-title");
const modalTextarea = modal.querySelector(".modal-textarea");
const modalColorPicker = modal.querySelector(".modal-color");
const modalTagInput = modal.querySelector(".modal-tags");
const modalDeadline = modal.querySelector(".modal-deadline");
const modalStatus = modal.querySelector(".modal-status");
const saveButton = modal.querySelector(".modal-save");
const deleteButton = modal.querySelector(".modal-delete");
const closeButton = modal.querySelector(".modal-close");
```

Рисунок 1.16 Код для пошуку полів вводу

Зміна стилів модального вікна для його відображення здійснюється через властивість cssText. Всередині задаються параметри:

1. display: flex – для правильного позиціонування,

2. visibility: hidden – для зміни видимості елементу на сторінці,

3. opacity: 0 – для регулювання прозорості елементу,

4. transition – для забезпечення плавності при відкритті та закритті модального вікна.

Код, який реалізує цю частину:

```
modal.style.cssText = `
    display: flex;
    visibility: hidden;
    opacity: 0;
    transition: opacity 300ms ease-in-out;
`;
```

Далі йде функція openModal, яка реалізує відкриття вікна по натисненню на картку. На початку вона отримує об’єкт картки, у вигляді параметру, далі йде перевірка на наявність картки, якщо її не було знайдено, то функція припиняє своє виконання. Далі відбувається заповнення полів для редагування інформацією, яка зберігається у базі даних, або виводяться стандартні значення, у разі, якщо зміни не вносилися. Після чого відбувається оновлення стилів, для того, щоб модальне вікно могло відображатись на сторінці.

```
function openModal(card) {
    if(!card) {
        return;
    }

    currentCard = card;

    modalTitle.textContent = "Налаштування картки";
    modalTextarea.value = (card.querySelector(".card-text")?.textContent || "").trim();
    modalColorPicker.value = card.dataset.color || "#ffffff";
    modalTagInput.value = card.dataset.tags || "";
    modalDeadline.value = card.dataset.deadline || "";
    modalStatus.value = card.dataset.status || 'not_done';

    updateCardAppearance();
    modal.style.visibility = "visible";
    modal.style.opacity = 1;
    window.addEventListener("keydown", handleKeyDown);
}
```

Рисунок 1.17 Код функції openModal

Далі йде функція closeModal, яка відповідає за закриття модального вікна, через зміну параметрів opacity та visibility, Анімація закриття відбувається плавно завдяки параметру setTimeout. Окрім цього в кінці функції відбувається скидання посилання на поточну картку, та видаляється обробник для подій, які викликаються по натисненню клавіш на клавіатурі.

Код функції closeModal:

```
function closeModal() {
    modal.style.opacity = 0;
    setTimeout(() => {
        modal.style.visibility = "hidden";
        currentCard = null;
    }, 300);
    window.removeEventListener("keydown", handleKeyDown);
}
```

Також для більш зручного управління модальним вікном була реалізована функція для закриття вікна по натисненню клавіші esc.

Код функції handleKeyDown:

```
function handleKeyDown(e) {
    if (e.code === "Escape") {
        closeModal();
    }
}
```

Потім йде функція, яка відповідає за управління станом виконання завдання. Вона отримує статус виконання задачі, і виходячи з його параметру встановлює стилі. Якщо задача позначена, як виконана, то текст всередині картки

закреслюється, сама картка стає більш прозорою та змінює свій колір на сірий, Якщо ж картка позначена, як ще не виконана, то стилі не змінюються і картка має стандартний зовнішній вигляд.

```
function updateCardAppearance() {
    if (!currentCard) return;

    const isDone = modalStatus.value === 'done';
    const cardText = currentCard.querySelector('.card-text');

    if (isDone) {
        cardText.style.textDecoration = 'line-through';
        currentCard.style.opacity = '0.7';
        currentCard.style.backgroundColor = '#f0f0f0';
    } else {
        cardText.style.textDecoration = 'none';
        currentCard.style.opacity = '1';
        currentCard.style.backgroundColor = modalColorPicker.value;
    }
}
```

Рисунок 1.18 Код функції updateCardAppearance

Далі йде основна функція, логіка якої відповідає за збереження змін, які були внесені. Вона працює асинхронно та взаємодіє з базою даних, поті оновлює відображення картки на сторінці.

Функція починається з перевірки, чи існує картка, до якої потрібно застосувати зміни, і якщо картка не була знайдена, функція завершує своє виконання. Далі відбувається перевірка на наявність решти необхідних елементів, таких, як: дошка, колонка, картка. Після чого знову відбувається перевірка на наявність необхідних елементів, і якщо хоча б один із необхідних елементів відсутній, то виводиться попередження про помилку та функція припиняє своє виконання.

Потім ортимуються дані для заповнення полів модального вікна, до нього входять: текст картки, колір картки, замітки, термін виконання і статус виконання. Після чого виконується перевірка тексту картки, і якщо він опиняється порожнім, то виводиться відповідне попередження, яке інформує, що картка не може бути пустою.

При натисненні кнопки «зберегти» змінюється її стан і вона робиться неактивною, щоб запобігти повторному натисненню, потім текст всередині

кнопки змінюється на «Збереження…», для того, щоб було зрозуміло, що йде процес збереження.

Викликається функція updatedCardAppearance, яка змінює зовнішній вид картки, у разі зміни статусу виконання завдання.

Потім створюється об’єкт updates, який містить всередині усі оновленні данні про картку. На наступному рядку коду виконується асинхронний запит до бази даних на оновлення інформації стосовно картки, після цього запиту відбувається пошук необхідних елементів для оновлення інформації, яка відображається на сторінці. Після процесу зберігання та оновлення інформації модальне вікно автоматично закривається.

Найважливіші рядки коду з функції saveChanges:

```
const updates = {
  text: newText,
  color: newColor,
  tags: newTags,
  deadline: newDeadline,
  status: newStatus,
  updatedAt: new Date().toISOString()
};

await database.ref(`boards/${boardId}/columns/${columnId}/cards/${cardId}`).update(updates);
```

Наступна функція під назвою deleteCard відповідає за видалення картки з бази даних і як елементу, який відображається на сторінці, вона, як і минула функція використовує асинхронний підхід для взаємодії з базою даних і оновлює інтерфейс користувача після успішного виконання операції.

Починається усе з перевірки на наявність поточної картки і якщо вона не була знайдена, то функція припиняє своє виконання. Далі зчитуються ідентифікатори для отримання дошки, колонки, та картки, і потім знову перевірка на наявність усіх необхідних елементів і якщо хоча б один елемент не буде знайдений функція припинить своє виконання і виведе вікно з відповідним повідомленням. Якщо ж усе успішно і всі елементи були знайдені, то функція продовжує своє виконання і на екран буде виведено повідомлення з підтвердженням про видалення колонки.

Якщо користувач підтверджує дію, то стан кнопки оновлюється і напис змінюється на «Видалення». Потім відбувається запит до бази даних про видалення колонки, після чого видаляється сама колонка, як елемент, який може відображатись на сторінці.

Після проведення вищезазначених процедур відбувається збереження стану поточної дошки для синхронізації змін. По закінченню всіх необхідних процедур модальне вікно закривається.

```
async function deleteCard() {
    if (!currentCard) return;

    const boardId = window.boardManager?.currentBoardId;
    const columnId = currentCard.closest('.column')?.dataset.columnId;
    const cardId = currentCard.dataset.cardId;

    if (!boardId || !columnId || !cardId) {
        alert("Не удалось удалить карточку. Обновите страницу.");
        return;
    }

    if (!confirm("Ви впевнені, що хочете видалити цю картку?")) {
        return;
    }

    deleteButton.disabled = true;
    deleteButton.textContent = "Видалення...";

    await database.ref(`boards/${boardId}/columns/${columnId}/cards/${cardId}`).remove();

    currentCard.remove();

    if (window.boardManager) {
        await window.boardManager.saveCurrentBoard();
    }

    closeModal();

    deleteButton.disabled = false;
    deleteButton.textContent = "Видалити";
}
```

Рисунок 1.19 Код функції deleteCard

У якості додаткової можливості управління картками було створено окремий файл, який реалізує переміщення карток між колонками, з повною синхронізацією між тим, що відбувається на екрані та базою даних.

Починається файл з функції hanaleDragStart, перш за все всередині функції визначається картка, яку користувач починає переміщувати, далі відбувається пошук колонки з якої почався процес перетягування картки. На наступному рядку коду відбувається перевірка, яка визначає, чи всі потрібні елементи були знайдені, і якщо ні, то функція завершує своє виконання.

Наступним кроком створюється об’єкт sourceCardData, який зберігає дані картки, яку переміщують.

Наприкінці додаються класи стилізації для візуального виділення картки.

```
function handleDragStart(e) {
    draggedCard = this;
    const column = this.closest('.column');

    if (!column || !column.dataset.columnId) {
        return;
    }

    sourceColumnId = column.dataset.columnId;
    sourceCardData = {
        text: this.querySelector('.card-text')?.textContent || "",
        color: this.style.backgroundColor || '#ffffff',
        tags: this.dataset.tags || "",
        deadline: this.dataset.deadline || "",
        createdAt: this.dataset.createdAt || new Date().toISOString()
    };

    this.classList.add('dragging');
    setTimeout(() => this.classList.add('invisible'), 0);
}
```

Рисунок 1.20 Код функції handleDragStart

У свою чергу, функція handleDrop завершує процес перетягування. Першим кроком виступає скасування стандартної поведінки браузера для події drop, досягається це викликом e.prevenrDefault(); ця дія виконується, щоб надалі реалізувати власну обробку події.

Видаляється клас drag-over з поточного елементу. Потім йде перевірка на наявність всіх необхідних даних для скидання картки.

Визначається цільова колонка, на яку картка буде переміщена, і знову перевірка на наявність даних.

Отримується ідентифікатор для визначення колонки, картки, дошки. Якщо не було знайдено один з елементів, функція припиняється.

Створюється об’єкт для оновлення, спочатку зберігається посилання на картку, переміщення якої зараз відбувається. Отримується статус її виконання, та актуальний колір. Створюється новий об’єкт з назвою updates, що містить оновлення для бази даних, спочатку він видаляє картку з початкової дошки і додає картку у колонку, до якої відбулося переміщення.

Виконується запит до бази даних і зміни отриманої інформації. Вкінці функція виконує очищення змінних, щоб підготувати функцію до наступного використання.

1.3.2 Розробка додавання нових списків та можливість їх налаштування

Для додавання нових колонок розроблено наступний алгоритм:

Рисунок 1.21 Алгоритм для додавання нових колонок

Для додавання нових списків створено окремий файл з назвою addcolumn. Починається файл з того, що в нього імпортуються необхідні функції з інших модулів, які будуть використовуватися у подальшому для ініціалізації, щоб новостворений елемент мав повну функціональність:

1. Імпортується об’єкт database з файлу firebase-config.

2. На другому рядку відбувається імпорт двох функцій з файлу dragdropcard, які будуть відповідати за те, щоб у новостворених колонках була можливість виконувати переміщення картки між іншими колонками.

3. З файлу positioncolumn імпортується функція observeColumnChanges, вона використовується для відстеження положення колонки, щоб вона не виходила за границі робочої області.

4. Функція initcolumncard, яка імпортується з файлу columnmodal використовується для ініціалізації модального вікна.

5. setupColumnDragFeature з файлу dragdropcolumn відповідає за те, щоб обрана колонка мала функціонал пересування всередині дошки.

6. initColumnRenaming з файлу renamecolumn реалізовує функцію перейменування колонки.

7. initcardmodal робить робочими модальні вікна для карток у створених колонках.

8. createCard надає можливість додавання карток.

9. adjustMenuPosition та initMenusPosition відповідають за правильне позиціонування меню, яке відкривається по натисненню кнопки.

Далі йде основна функція для додавання нових колонок під назвою createNewColumn. На її початку відбувається пошук активної дошки, на котрій зараз знаходиться користувач. Після відбувається перевірка на наявність такого елементу, якщо він знайдений не був, повертається null і функція не виконується далі.

При успішному знаходженні необхідного елементу функція просувається далі і створює новий запис у Firebase на активній дошці, при цьому використовується метод push(), що додає новий елемент до вже існуючого масиву, також генерується унікальний id-ключ для подальших взаємодій з колонкою. Наступним кроком отримується id-ключ, який щойно був згенерований

Потім отримується загальна кількість існуючих колонок, які є у Firebase. Робиться це для визначення правильної позиції при створенні нової колонки.

Наступним кроком створюється об’єкт columnData для збереження даних колонки, яка буде створюватися. До них входять:

1. name – назва колонки.

2. position – позиція колонки на дошці

3. cards – використовується для збереження карток, на початку пустий.

4. createdAt – дата створення колонки.

Як тільки вище згадана інформація була записана відбувається запит до бази даних, який записує об’єкт columnData у новостворену колонку.

Після цього відбувається створення колонки уже на сторінці, щоб її можна було побачити, зокрема це новий div, якому надається клас column, а також вже згенерований унікальний id. Потім відбувається наповнення HTML-розміткою для відображення на сторінці, а також додавання стилів.

Наприкінці функції викликається initColumnEventHandlers(newColumn); яка відповідає за ініціалізацію всіх імпортованих функцій з інших файлів. Зберігається стан дошки на якій відбулись зміни, і у самому кінці повертається нова колонка, для подальших маніпуляцій.

Налаштування колонок, як і у випадку з картками відбувається через взаємодію з модальним вікном, але, якщо у випадку з карткою для відкриття вікна достатньо просто натиснути на неї, то для того, щоб відкрити модальне вікно колонки потрібно спочатку перейти до меню, яке відкривається по клавіші розташованій у верхньому правому куті. Тож спочатку розберемо файл menu, який відповідальний за відкриття та закриття меню і функціональність кнопок, які розташовані всередині.

Файл починається з імпортування необхідних функцій з інших файлів:

1. adjustMenuPosition відповідає за те, щоб при відкритті меню не виходило за межі робочого простору.

2. database відповідає за зв’язок з базою даних.

Файл складається з єдиної функції – initColumnMenus, яка призначена для ініціалізації меню, його вмісту, а також закриття меню.

Функція починається з пошуку кнопки меню при натисненні на яку відбувається його відкриття, пошук здійснюється за допомогою селектору .menu-button.

При натисненні на кнопку відкриття, виконується перевірка на те, чи є вона активною, якщо кнопка неактивна, вона змінює свій клас на активний і меню починає відображатись, його позиція коригується за допомогою імпортованої функції adjustMenuPosition. При цьому, якщо було відкрите якесь меню іншої колонки воно буде закрито автоматично для уникнення плутанини.

У меню присутня кнопка для закриття, вона має клас .menu-close. Для виконання процедури закриття налаштовується подія кліку, при якому меню закривається, а кнопка стає неактивною для уникнення повторного натискання.

У меню присутня кнопка з написом «Видалити колонку», вона має клас .menu-item.item3, вона відповідає за видалення колонки. При натисненні на кнопку з’являється вікно, всередині якого користувач має підтвердити видалення колонки. У разі підтвердження дії відбувається запит до бази даних, там знаходиться запис про необхідну колонку, і вона видаляється разом з усіма картками, що знаходились в ній на момент підтвердження операції. Потім колонка видаляється і на сторінці, щоб вона не відображалась, завершується процес видалення збереженням актуального стану дошки з усім її наявним змістом.

Вкінці коду реалізована ще одна функція для закриття меню, але вже коли відбувається клік мишкою за межами відкритого меню.

Налаштування колонки здійснюється завдяки модальному вікну, код для роботи якого було винесено у окремій файл під назвою columnmodal. Відкривається воно через меню, роботу якого розібрано вище, за допомогою кнопки «Налаштування колонки».

Починається файл columnmodal з імпорту елементу для взаємодії з базою даних. Далі відбувається ініціалізація елементів модального вікна, до яких входять: заголовок, текстове поле для редагування назви колонки, поле для заміток, поле для введення терміну виконання, невелику форму для реалізації

переміщення колонок між дошками і дві кнопки, для збереження змін і для видалення колонки. Потім відбувається зміна стилів для модального вікна, аналогічна до тої, що використовується для вікна налаштування карток.

Функція, яка відповідає за відкриття вікна називається openModal. Починається з того, шо вона отримує поточну колонку, для якої потрібно відкрити вікно. Також визначаються необхідні елементи (дошка, колонка, картка). Далі перевірка на наявність елементів, і якщо вони не були знайдені, функція завершується.

Далі відбувається зчитування даних з Firebase, зокрема заповнюються поля з інформацією про встановлену дату виконання, а також поле з замітками. Паралельно з цими діями викликається окрема функція loadBoards(), яка відповідає за завантаження списку усіх доступних дошок для можливості переміщення колонки між ними.

На фінальному етапі функції відбувається зміна CSS-властивостей для коректного відображення вікна, а саме застосовується параметр visible та встановлюється opacity = 1. Додавання обробника для реалізації закриття вікна при натисненні клавіші Esc.

```
function openModal(column) {
    currentColumn = column;
    currentColumnId = column.dataset.columnId;
    currentBoardId = window.boardManager?.currentBoardId;

    if (!currentColumnId || !currentBoardId) {
        return;
    }

    modalTitle.textContent = "Налаштування колонки";
    modalTextarea.value = column.querySelector('.column-title')?.textContent || '';

    database.ref(`boards/${currentBoardId}/columns/${currentColumnId}`).once('value')
        .then(snapshot => {
            const columnData = snapshot.val() || {};
            modalTags.value = columnData.tags || '';
            modalDeadline.value = columnData.deadline || '';
        });

    loadBoards();

    modal.style.visibility = "visible";
    modal.style.opacity = 1;
    window.addEventListener("keydown", handleKeyDown);
}
```

Рисунок 1.22 Код функції openModal

Для збереження внесених змін було реалізовано окрему функцію saveChanges(), вона є асинхронною та в основному взаємодіє з базою даних для оновлення інформації.

Спочатку функція отримує поточного користувача за допомогою auth.currentUser, далі відбувається перевірка за допомогою умовного оператору if, і якщо користувач не був знайдений, виконання закінчується.

Наступним кроком створюється об’єкт updates, у який поступово будуть додаватися нові значення для оновлення у базі даних. Серед них:

1. newTitle – нова назва для колонки, якщо вона змінювалась.
2. newTags – нові замітки.
3. newDeadLine – оновлене значення дати виконання.

Далі відбувається заповнення об’єкта updates про внесені зміни до параметрів колонки, які зазначенні вище. Після того, як усі поля були заповнені інформацією відбувається запит до бази даних на оновлення наявних змін. По закінченню внесення змін до бази даних викликається функція closeModal(), яка автоматично закриває вікно.

```
async function saveChanges() {
    const user = auth.currentUser;
    if (!user) return;

    const updates = {};
    const newTitle = modalTextarea.value.trim();
    const newTags = modalTags.value.trim();
    const newDeadline = modalDeadline.value;

    if (currentColumn) {
        const titleElement = currentColumn.querySelector('.column-title');
        if (titleElement) titleElement.textContent = newTitle;
    }

    updates[`boards/${currentBoardId}/columns/${currentColumnId}/name`] = newTitle;
    updates[`boards/${currentBoardId}/columns/${currentColumnId}/tags`] = newTags;
    updates[`boards/${currentBoardId}/columns/${currentColumnId}/deadline`] = newDeadline;
    updates[`boards/${currentBoardId}/columns/${currentColumnId}/updatedAt`] = new Date().toISOString();

    await database.ref().update(updates);
    closeModal();
}
```

Рисунок 1.23 Код функції saveChanges

Для видалення колонок через модальне вікно було створено окрему функцію deleteColumn(). Першим ділом користувачу виводиться повідомлення, в якому йому потрібно підтвердити видалення обраної колонки.

Наступним етапом є видалення колонки з бази даних, завдяки асинхронному запиту. Вкінці запиту використовується метод remove(), він забезпечує повне видалення вузла з бази даних, разом з усіма елементами, що містяться у ньому. Тобто разом з колонкою будуть видалятися усі картки, які містяться всередині.

По закінченню видалення колонки з бази даних починається оновлення інтерфейсу, на початку цього оновлення виконується перевірка на наявність необхідної колонки, після перевірки знову використовується метод remove(), але у цей раз він видаляє колонку з HTML-розмітки, щоб вона більше не відображалась.

На фінальному етапі функції викликається функція closeModal() для закриття вікна, та виконується збереження актуального стану дошки для подальших маніпуляцій.

У якості ще одної опції налаштування колонок розроблено функціонал для її переміщення між дошками. На першому етапі виконання відбувається низка перевірок:

1. Перевірка на те, чи є авторизованим користувач, якщо користувач не авторизований то з’являється відповідне повідомлення і функція припиняє виконання.

2. Перевірка на наявність необхідних даних для виконання процедури переміщення, а саме: дошка на якій знаходиться колонка, та сама колонка. Якщо один з цих елементів не був знайдений, то виконання припиняється і з’являється відповідне повідомлення.

3. Перевірка на наявність дошки, на яку користувач хоче перемістити колонку, у разі не знаходження виводиться повідомлення про необхідність обрати іншу дошку. Також виконання припиняється.

Після успішного виконання усіх перевірок відбувається підготовка інтерфейсу: блокується кнопка для уникнення повторного натиснення, змінюється напис всередині для того, що показати, що процес переміщення

відбувається. Наступним етапом є реалізація основної логіки для взаємодії з базою даних:

1. Отримання повної копії даних потрібної колонки, завдяки методу once(‘value’).

2. Перевірка наявності даних колонки (дані, які були внесені у модальне вікно, її назва, картки, які містяться всередині).

Після чого формується об’єкт updates для оновлення бази даних, він включає у себе:

1. Видалення колонки з актуальної дошки, за допомогою встановлення значення null.

2. Додавання колонки до іншої дошки, яка була обрана користувачем зі збереженням усіх оригінальних даних.

3. Оновлення позиції, яку вона займає.

Далі відбувається робота з глобальним станом додатку, вона включає:

1. Видалення елементу, включно з HTML-розміткою з поточної дошки.

2. Оновлюються локальні дані про обидві дошки (дошку з якої переміщується колонка, та дошку на яку переміщується колонка).

На фінальному етапі виконання функції виводиться повідомлення про успішне завершення операції переміщення. Виклик функції для автоматичного закриття модального вікна, і оновлення стану дошок.

Найважливіші рядки коду з функції moveColumn:

```
const targetSnapshot = await database.ref(`boards/${targetBoardId}/columns`).once('value');
const targetColumns = targetSnapshot.val() || {};

const updates = {
  [`boards/${currentBoardId}/columns/${currentColumnId}`]: null,
  [`boards/${targetBoardId}/columns/${currentColumnId}`]: {
    ...columnData,
    position: Object.keys(targetColumns).length,
    updatedAt: new Date().toISOString()
  }
};

await database.ref().update(updates);
```

Окрім переміщення між колонками було розроблено можливість пересування колонок всередині дошки, це окремий функціонал, який знаходить всередині файлу dragdropcolumn. Файл складається з однієї великої функції, тому увагу буде приділено лише основним моментам.

На початку функції відбувається оголошення змінних, наприклад: initiaX/Y відповідає за початкові координати курсору при пересуванні, offsetX/Y – зміщення курсору відносно верхнього лівого кута колонки, який вважається нульовою точкою для початку розрахунку положення колонки, isDragging – визначає активність процесу перетягування, lockAxis – відповідає за те, яка з осей при натисненні клавіші shift буде заблокована, для пересування лише по одній.

Основною функцією для реалізації перетягування є onMousMoveю Першим ділом виконується перевірка на активність процедури перетягування та елементу, який потрібно перетягнути, якщо один з них не буде знайдений, функція виконуватися не буде.

Відбувається обчислення нових координат позиції курсору, за допомогою вище згаданих offsetX, offset.

Виконується перевірка на те, що одна з осей була зафіксована, при натисненні клавіші shift. Тобто пересування доступне лише по вертикалі або лише по горизонталі. Після визначення того, яка з осей була обрана для переміщення відбувається блокування однієї з двох доступних осей.

Відбувається визначення границь дошки для того, щоб колонка не виходила за межі доступного простору, а також, щоб при переміщенні вона не могла опинитися під верхньою панеллю навігації, а при наближенні до правої частини сторінки відбувалось розширення колонки.

Після чого відбувається призначення нових координат для елементу. Окрім цього реалізовано функцію getSnapPosition, яка реалізовує прилипання по відношенню до інших дошок. Тобто при наближенні однієї колонки до іншої по вертикалі або горизонталі відбувається вирівнювання колонок відносно одна одної як по горизонталі, так і по вертикалі. Це дозволяє розташовувати колонки рівно в ряд.

Також є окрема перевірка для розширення дошки, якщо правий край колонки знаходиться на відстані меншій за 20 пікселів від правої сторони дошки відбувається розширення колонки на 100 пікселів. Це дозволяє розширювати робочий простір у ситуаціях, коли колонки вже не влізають у стандартну ширину дошки. Також присутня функція, яка дозволяє зменшити ширину дошки, якщо та вже була розширена. Разом з цим реалізовано процес автоматичного прокручування сторінки при розширенні робочої зони.

```
function onMouseMove(e) {
    if (!isDragging || !draggingElement) return;

    let newX = e.clientX - offsetX;
    let newY = e.clientY - offsetY;

    if (isShiftPressed && lockAxis === null) {
        const moveX = Math.abs(e.clientX - initialX);
        const moveY = Math.abs(e.clientY - initialY);
        lockAxis = moveX > moveY ? "x" : "y";
    }

    if (lockAxis === "x") {
        newY = draggingElement.offsetTop;
    } else if (lockAxis === "y") {
        newX = draggingElement.offsetLeft;
    }

    const board = document.querySelector(".board");
    const boardRect = board.getBoundingClientRect();
    const columnRect = draggingElement.getBoundingClientRect();
    const headerRect = document.querySelector(".header").getBoundingClientRect();
    const viewportWidth = window.innerWidth;
    const scrollLeft = window.scrollX;

    newX = Math.max(boardRect.left + scrollLeft, newX);
    newY = Math.max(
        headerRect.bottom,
        Math.min(newY, boardRect.bottom - columnRect.height)
    );

    draggingElement.style.left = `${newX}px`;
    draggingElement.style.top = `${newY}px`;

    const snapPosition = getSnapPosition(newX, newY, draggingElement);
    draggingElement.style.left = `${snapPosition.x}px`;
    draggingElement.style.top = `${snapPosition.y}px`;

    if (columnRect.right > boardRect.right - 20) {
        board.style.width = `${board.scrollWidth + 100}px`;
    }

    if (e.clientX > viewportWidth - scrollThreshold) {
        startAutoScroll(scrollSpeed, scrollStep);
    } else if (e.clientX < scrollThreshold) {
        startAutoScroll(-scrollSpeed, -scrollStep);
    } else {
        stopAutoScroll();
    }
}
```

Рисунок 1.24 Код функції onMoseMove

Наступна функція onMouseUp реалізує процес завершення переміщення колонок всередині дошки. Спочатку виконується перевірка прапорця isDragging, що запобігає випадковому спрацюванню функції при звичайних кліках мишею.

Далі за допомогою removeEventLisyener відбувається видалення обробників подій, а саме обробник для відстеження рухів миші, та відстеження за перетягуванням.

Далі відбувається збереження нової позиції колонки, через виклик saveColumnPosition, по завершенню процедури зберігання даних здійснюється відновлення стандартних стилів елементу:

1. zIndex – скасовує підняття елементу над іншими.
2. cursor – робить зовнішній вигляд курсору стандартним.
3. boxShadow – прибирає ефект тіні, який з'являється у процесі перетягування.

Також відбувається оновлення змінних, для припинення поточного процесу перетягування і безперешкодній ініціалізації нового. Наприкінці відбувається синхронізація стану сторінки, тобто відбувається збереження актуального стану дошки.

```
async function onMouseUp() {
    if (isDragging) {
        document.removeEventListener('mousemove', onMouseMove);
        document.removeEventListener('mouseup', onMouseUp);

        if (draggingElement) {
            await saveColumnPosition(draggingElement);

            draggingElement.style.zIndex = '';
            draggingElement.style.cursor = '';
            draggingElement.style.boxShadow = '';
        }

        draggingElement = null;
        lockAxis = null;
        isDragging = false;

        if (window.boardManager) {
            window.boardManager.saveCurrentBoard();
        }
    }
}
```

Рисунок 1.25 Код функції onMouseUp

1.3.3 Розробка системи управління дошками

Для управління дошками було розроблено файл boardmanager, який реалізовує створення, редагування, видалення та відображення дошок, які доступні користувачу. Всередині реалізовано клас, з аналогічною назвою у якому і відбувається уся необхідна взаємодія.

Клас починається з конструктору, який ініціалізує основні властивості із якими у подальшому буде відбуватись взаємодія: userId – ідентифікатор поточного користувача, boards – об’єкт для зберігання усіх дошок, currentBoardId – ідентифікатор активної дошки. Далі викликаються методи initElements(), який знаходить усі необхідна елементи, та initEventListeners() для налаштування обробників подій.

Робота з дошками починається з функції loadBoards, вона завантажує дані користувача з бази даних. Починається усе з перевірки на наявність активного користувача, якщо його не було знайдено, функція припиняється. Далі відбувається запит до бази даних на пошук усіх дошок, які належать користувачу, але якщо це тільки но зареєстрований користувач, то дошок у нього немає, тому викликається функція createInitalBorad, яка автоматично створює стартову дошку. Наступним кроком є завантаження даних кожної дошки. Створюється пустий об’єкт this.boards, заповнення якого відбувається через цикл for, у цьому циклі відбувається запит до бази даних, і якщо дошка існує, то вона зберігається, і на виході отримується об’єкт заповнений інформацією про усі актуальні дошки для користувача.

Наступним етапом буде визначення активної дошки, вона отримується завдяки значенню lastActiveBoard з бази даних, там зберігається ідентифікатор останньої дошки, яка використовувалась.

Останнім кроком буде виділення активної дошки у боковому меню завдяки невеликій зміні кольору, це зроблено для того, щоб легше орієнтуватися, на якій саме дошці зараз знаходиться користувач. Відбувається це через визначення this.curretBoardId і якщо вона є активною, то викликається метод renderCurrentBoard() для її відображення.

За створення нових дошок відповідальна функція createNewBoard, окрім цього вона також створює колонки на дошці, яка була щойно створена. Для цього виконуються наступні кроки:

1. Генерація унікального ідентифікатора дошки за допомогою const newBoardRef = database.ref('boards').push(); та const boardId = newBoardRef.key.

2. Створення назви дошки. По замовчуванню це дошка плюс її номер серед уже існуючих.

3. Відбувається генерація унікального id за допомогою const columnId = database.ref().push().key;

4. Створюється HTML-розмітка у boardHML для відображення колонки на дошці.

5. Створюється об’єкт boardData у якому зберігаються дані про дошку, яка була створена, а саме: назва дошки, id користувача, який створив дошку, HTML-розмітка дошки і окремий об’єкт columns у якому знаходиться інформація про колонки.

6. Збереження даних у Firebase, а також створення нових записів для коректної взаємодії у майбутньому.

7. Оновлення інтерфейсу для коректного відображення на сторінці, а також оновлення списку дошок у бічному меню.

Ключові рядки коду для створення нового запису у базу даних:

```
const boardData = {
   name: boardName,
   owner: this.userId,
   html: boardHTML,
   createdAt: new Date().toISOString(),
   updatedAt: new Date().toISOString(),
   columns: {
      [columnId]: {
         name: "Нова колонка",
         position: 0,
         cards: {},
         createdAt: new Date().toISOString()
         }
      }
   };

await newBoardRef.set(boardData);
```

```
await database.ref(`users/${this.userId}/boards/${boardId}`).set(true);
await database.ref(`users/${this.userId}/lastActiveBoard`).set(boardId);
```

1.3.4 Розробка процесу аутентифікації користувача

Для реалізації процесу аутентифікації користувача реалізовано окремий файл з назвою registermodal. Починає він своє виконання з процесу пошуку необхідних елементів за допомогою querySelector, з якими буде відбуватися взаємодія, серед них: кнопки авторизації та профілю, поля для заповнення інформацією, кнопки перемикання між вікном реєстрації та входу в акаунт.

Далі виконується ініціалізація стану користувача, зокрема оголошуються змінні для зберігання поточного користувача, даних користувача й модального вікна профілю:

```
let currentUser = null;
let userData = null;
let profileModal = null;
```

За допомогою onAuthStateChanged відбувається відстеження стану авторизації. Якщо користувач авторизований, то завантажуються його дані (loadUserData), оновлюється інтерфейс (updateAuthUI(true)) приховується вікно для реєстрації, і з’являється робочий інтерфейс. Якщо з’являється користувач у якого ще немає акаунту, або він вийшов з нього, то відбувається очищення даних (userData = null), оновлюється інтерфейс (updateAuthUI(false)), і здійснюється очищення форми (clearForms).

```
auth.onAuthStateChanged(async (user) => {
    currentUser = user;

    if (user) {
        userData = await loadUserData(user.uid);
        updateAuthUI(true);
        authModal.style.display = 'none';
    } else {
        userData = null;
        updateAuthUI(false);

        if (profileModal) {
            profileModal.style.display = 'none';
            profileModal = null;
        }

        clearForms();
    }
});
```

Рисунок 1.26 Код для onAuthStateChanged

Для кнопки входу в акаунт (loginBtn) додається обробник подій addEventListener, який реагує на натискання. Якщо відбулося натиснення на кнопку для входу до акаунту відбувається отримання полів у які вводяться електронна пошта та пароль. Далі йде перевірка на їх заповненість, якщо поля виявляться пустими, то виведеться відповідне повідомлення і виконання зупиниться. Якщо ж перевірка була виконана успішно, то виконується вхід до акаунту через auth.signInWithEmailAndPassword. Вкінці відбувається очищення форми, щоб при повторному відкритті поля були пустими

```
loginBtn.addEventListener('click', async (e) => {
    e.preventDefault();
    const email = loginForm.querySelector('.email').value;
    const password = loginForm.querySelector('.password').value;

    if (!email || !password) {
        alert('Будь ласка, заповніть всі поля');
        return;
    }

    await auth.signInWithEmailAndPassword(email, password);
    authModal.style.display = 'none';
    clearForms();
});
```

Рисунок 1.27 Код для loginBtn

Аналогічним чином відбувається обробка процесу реєстрації, з’являється лише одне додаткове поле для вводу імені користувача. Тобто отримуються поля у які потрібно ввести інформацію, після чого по натисненню на кнопку відбувається перевірка, чи не є одне з трьох полів пустим. Якщо одне з полів виявиться не заповненим, то на екран виведеться повідомлення з відповідною помилкою, а виконання функції в свою чергу припиниться. Далі створюється новий користувач і відбувається запит до бази даних для збереження введеної інформації, такої як: пошта, пароль та ім’я користувача. Після чого модальне вікно автоматично закривається, поля для заповнення інформації очищуються і оновлюється інтерфейс за допомогою функції updateAuthUI з подальшим виведенням на екран робочого простору, з яким користувач вже може взаємодіяти.

```
registerBtn.addEventListener('click', async (e) => {
    e.preventDefault();
    const name = registerForm.querySelector('.name').value;
    const email = registerForm.querySelector('.email').value;
    const password = registerForm.querySelector('.password').value;

    if (!name || !email || !password) {
        alert('Заповніь всі поля');
        return;
    }

    const userCredential = await auth.createUserWithEmailAndPassword(email, password);
    const user = userCredential.user;

    await database.ref('users/' + user.uid).set({
        username: name,
        email: email,
        createdAt: new Date().toISOString(),
        boards: {}
    });

    authModal.style.display = 'none';
    clearForms();
    updateAuthUI(true);
});
```

Рисунок 1.28 Код для обробки процесу реєстрації

Рисунок 1.29 Алгоритм процесу реєстрації

Функція яка відповідає за показ вікна профілю називається showProfile, вона починається з того, що виконується перевірка на те, чи існує вікно профілю, і якщо воно не було знайдено, то викликається функція createProfileModal, задача якої є створення цього самого вікна. Потім за допомогою querySelector відбувається отримання посилань на елементи вікна для подальшого заповнення інформацією, до них входять поля: імені користувача, електрона пошта, ідентифікатор користувача, та дата реєстрації. Наступним кроком є заповнення вище згаданих полів інформацією, у випадку, якщо потрібна інформація не знайдена, то буде виведено напис не знайдено. У кінці функції змінюється CSS стиль для відображення вікна на екрані.

```
function showProfile(user, data) {
    if (!profileModal) {
        profileModal = createProfileModal();
    }

    const usernameEl = profileModal.querySelector('.profile-username');
    const emailEl = profileModal.querySelector('.profile-email');
    const idEl = profileModal.querySelector('.profile-id');
    const createdAtEl = profileModal.querySelector('.profile-created-at');

    usernameEl.textContent = data?.username || user.displayName || 'Не вказано';
    emailEl.textContent = user.email || 'Не вказано';
    idEl.textContent = user.uid || 'Не вказано';
    createdAtEl.textContent = data?.createdAt ?
        new Date(data.createdAt).toLocaleDateString() :
        new Date(user.metadata.creationTime).toLocaleDateString() || 'Не вказано'

    profileModal.style.display = 'flex';
}
```

Рисунок 1.30 Код функції showProfile

Функція для створення вікна профілю, як уже згадувалось це createProfileModal. Тому далі мова піде про неї. На початку функція створює контейнер div, у якому буде відображення усіх необхідних елементів. Для цього контейнеру задається клас ‘modal modal-profile’, де modal це базовий клас для модальних вікон, який також використовувався у всіх попередніх випадках, а modal-profile це специфічний клас, саме для вікна профілю. Далі відбувається форматування HTML-розмітки.

```
modal.innerHTML = `
    <div class="modal-main">
        <div class="modal-header">
            <h2 class="modal-title">Профіль</h2>
            <button class="modal-close">X</button>
        </div>
        <div class="modal-content profile-content">
            <div class="modal-content">
                <div class="modal-content-title">
                    <label>Ім'я користувача:</label>
                    <span class="profile-username field-value">Не вказано</span>
                </div>
                <div class="modal-content-title">
                    <label>Email:</label>
                    <span class="profile-email field-value">Не вказано</span>
                </div>
                <div class="modal-content-title">
                    <label>ID користувача:</label>
                    <span class="profile-id field-value">Не вказано</span>
                </div>
                <div class="modal-content-title">
                    <label>Дата реєстрації:</label>
                    <span class="profile-created-at field-value">Не вказано</span>
                </div>
            </div>
            <div class="modal-button-container">
                <button class="modal-logout">Вийти</button>
            </div>
        </div>
    </div>
`;
```

Рисунок 1.31 HTML-розмітка, яка використовується для вікна профілю

Після додавання графічної частини за допомогою document.body.appendChild(modal) вікно додається до документу. Для кнопки, яка має клас ‘modal-logout’ через використання addEventListener викликається метод signOut(), що забезпечує вихід з акаунту при натисненні на кнопку. Далі аналогічним відбувається додавання обробників для закриття вікна.

1.3.5 Розгортання сайту на Firebase

Оскільки для реалізації бази даних був обраний сервіс Firebase, то і хостинг сайту буде відбуватись на ньому.

Спочатку за допомогою npm треба встановити набір інструментів Firebase, для цього використовується команда npm install -g firebase-tools. Де npm це пакетний менеджер, який працює на node.js і використовується для встановлення або оновлення бібліотек. Install це команда для завантаження пакету. -g вказує на те, що завантаження має відбутись глобально. Firebase-tools це назва необхідного пакету.

Далі вводиться команда firebase login, вона використовується для входу в акаунт, до якого під’єднаний Firebase.

Наступна команда – firebase init, відповідає за ініціалізацію проекту. Після того, як відбудеться її виконання, у консолі з’явиться перелік доступних дій, обрати потрібно обрати опцію з назвою Hosting. Далі відбувається вибір проекту, тобто визначається, це створюється новий проект, або це робиться для вже існуючого (варто зазначити, що мова йде про проект, який створюється саме у середовищі Firebase). У разі вибору опції для вже існуючого проекту потрібно обрати потрібний зі списку запропонованих. Далі задається директорія у якій знаходиться файли з кодом для сайту.

На останньому етапі вводиться команда firebase deploy, яка вже розгортає додаток на Firebase та видає посилання на нього, за яким можна перейти. Якщо до коду сайту вносяться якісь зміни, наприклад, треба виправити помилку, яка виникла, то у консоль треба знову вводити firebase deploy для того, щоб застосування у коді застосувались.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

В даному дипломному проекті було розроблено програмний продукт для відстеження ступеня готовності виконання проєкту.

Ефективність кожного програмного продукту визначається його якістю та ефективністю процесу розробки. Якість ПП визначається наступними складовими: з точки зору користувача; з позиції використання ресурсів; виконання вимог до програмного забезпечення.

2.2 Розрахунок ціни програмного продукту нормативним методом

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку. Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначається обсяг програмних засобів, у тисячах умовних машинних команд програми аналога.

У таблиці 2.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт.

Таблиця 2.1. Каталог аналогів

Найменування ПП	Обсяг функції ПП – V_o, усл. Машинних командах.
3. ПП введення інформації	1060 – 5750

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл.2.2

Таблиця 2.2. Vo в умовних машинних командах

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
3.00	262

На підставі отриманого значення визначається укрупнена норма часу на розробку аналога програмного забезпечення, яка коректується поправочним

коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8:

T^a = 262 х 0,8 = 209,6 (люд/годин)

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

- Розробка технічного завдання

  $T_{ТЗ} = T^a p * L_1 * K_H$

- Розробка технічного проекту

  $T_{ТП} = T^a p * L_2 * K_H$

- Розробка робочого проекту

  $T_{РП} = T^a p * L_3 * K_H * K_T$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 2.2.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.3.);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.4.).

Таблиця 2.3. Значення питомих коефіцієнтів трудомісткості

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

У зазначеному варіанті це ступінь новизни – Б, де L_1 = 00,12, L_2 = 00,15, L_3 = 00,58

Таблиця 2.4. Значення поправочного коефіцієнта

Код ступеня новизни	Ступінь новизни	Значення K_H
А	Принципово нові ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП маючий аналог	0,7

У зазначеному варіанті значення$K_Н$ = 1,0

Таблиця 2.5. Значення коефіцієнта ступеня використання

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	Значення $K_Т$
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

У зазначеному варіанту ступінь охоплення реалізованих функцій становить 20-40%, відповідно значення $K_Т$ = 0,8

Розрахунок трудомісткості по кожному етапу окремо:

- Трудомісткість технічного завдання

  $T_{ТЗ} = T^а р * L_1 * K_Н = 209,6 * 0,12 * 1 = 25,12$ (люд/годин)

- Трудомісткість розробки технічного проекту

  $T_{ТП} = T^а р * L_2 * K_Н = 209,6 * 0,15 * 1 = 34,44$ (люд/годин)

- Трудомісткість розробки робочого проект

  $T_{РП} = T^а р * L_3 * K_Н * K_Т = 209,6 * 0,58 * 1 * 0,8 = 97,25$ (люд/годин)

Для подальших розрахунків необхідно визначити кількість папера, витраченого на кожен етап. $N_{тз}$= 2 (стр), $N_{тп}$= 15 (стр), $N_{рп}$= 25 (стр), $N_{пз}$= 25 (стр) – технічне завдання, розробка технічного проекту, розробка робочого проекту, пояснювальна записка відповідно. Розрахунок зведений у таблицю 2.6

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, години.		
1	Розробка ПП	Контроль керівника	Нормоконтроль
1.ТЗ	$T_{Ртз}$=25,12	$T_{кк}$=0,7*$N_{тз}$=0,7*2=1,4	$T_{нк}$=0,15*$N_{тз}$=0,15*2=0,3
2.Розробка ТП	$T_{Ртп}$=34,44	$T_{кк}$=0,7 * $N_{тп}$=0,7*15=10,5	$T_{нк}$=0,15*$N_{тп}$=0,15*15=2,25
3.Розробка РП	$T_{Ррп}$=97,25	$T_{кк}$=0,7 * $N_{рп}$=0,7*25=17,5	$T_{нк}$=0,15*$N_{рп}$=0,15*25=3,75
4.Розробка пояснювальної	$T_{пз}$=1,5 $N_{пз}$=1,5*25=37,5	$T_{кк}$=0,7 * $N_{тз}$=0,7*25=17,5	$T_{нк}$=0,15*$N_{пз}$=0,15*25=3,75

Продовження таблиці 2.6.

записки			
Усього, в т.ч.:	$T_{пп}$=243,76		
на розробку	$\sum T_р$=186,81		
- контроль керівника		$\sum T_{хк}$=46,9	
нормоконтроль			$\sum T_{нк}$=10,05

2.3 Розрахунок ціни програмного продукту

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПО. Розрахунок основної заробітної плати виконавців приведений у таблиці 6.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2024» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2025 року – 8000 гривень; мінімальну погодинну тарифну ставку – 46 грн.

Таблиця 2.7. Розрахунок основної заробітної плати виконавців

Найменування робіт	Трудомісткість робіт, роб.години	Годинна тарифна ставка,грн..	Розрахунок, грн.
1.Розробка ПП	186,81	70,00	13076,7
2.Контроль керівника	46,9	80,00	3752
3.Нормоконт-роль	10,05	80,00	804
Усього (З о	-	-	∑Зо = 17632,7

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПО

Найменування матеріальних витрат	Тип, модель	Кількість, шт	Ціна одиниці, грн.	Вартість, грн.
Папір А1	-	-	-	-
Папір А4	арткуш	70	2,0	140

Продовження таблиці 2.8.

Разом	-	-	-	$В_{м\ і} = 140$
Транспортно заготівельні Витрати 10%				$В_{тр\_з} = 0,1 \times В_{м1} = 14$
Усього				$В_м = В_{м1} + В_{тр\_з} = 154$

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	154	В м (див. табл. 2.8)
2. Основна заробітна плата	17632,7	З о (див. табл. 2.7)
3.Додаткова заробітна плата	2644,9	*Зд = 0,15× Зо =0,15*17632,7*
4.Відрахування до єдиного фонду соціального внеску	4459,1	$В_{є.с.в.}=0,22 \times (З_о + З_д) =0,22*(17632,7+2644,9)$
5. Накладні витрати	5289,81	$В_{нак.} = 0,3*З_о = 0,3*17632,7$
6. Повна собівартість	30180,51	$С_{пов} = В_м + З_о + З_д + В_{є.с.в.} + В_{нак.} =$ 154+21615+3242,25+5468,59+6484,5

Розмір прибутку, що включається в ціну, визначається по наступній формулі:

$П=(С_{пов}*Р)/100 = (30180,51*10) / 100 = 3018,1$ (грн); (2.6)

Де Р– плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$Ц_о = С_{пов} + П = 30180,51 + 3018,1 = 33198,61$ (грн); (2.7)

Податок на додану вартість визначається по наступній формулі:

$ПДВ = 0.2 * Ц_о = 0,2 * 33198,61 = 6639,7$ (грн); (2.8)

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$Ц_р = Ц_о + ПДВ = 33179,61 + 6639,7 = 39819,31$ (грн); (2.9)

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА БЕЗПЕКИ

Організація охорони праці та безпеки на виробництві завжди була дуже важлива для збереження здоров'я працівників. З метою забезпечення безпечних умов праці спеціально створюються алгоритми дій та інструкції щодо виконання робочих завдань, які містять чітко визначені правила та послідовності дій. Кожен працівник повинен бути ознайомлений з цими інструкціями ще до початку виконання своїх обов'язків, проходити відповідні інструктажі.

3.1 Основні шкідливі чинники, що впливають на програміста

Одним із основних шкідливих чинників, що впливають на працівників, які здійснюють професійну діяльність із застосуванням комп'ютерної техніки, є статичне навантаження на опорно-руховий апарат. Тривале перебування у сидячому положенні протягом робочого дня, що у середньому становить від 7 до 10 годин, негативно позначається на функціональному стані м'язів, порушує кровообіг і спричиняє деформаційні зміни хребетного стовпа. Нерухоме положення тіла зумовлює надмірне навантаження на окремі групи м'язів, зокрема в ділянці шийного та грудного відділів хребта, плечового поясу, попереку. Хронічне м'язове перенапруження, відсутність регулярної зміни положення тіла та активного розслаблення м'язів можуть спричинити розвиток різноманітних захворювань.

Наступним фактором, що суттєво впливає на стан здоров'я працівника, є зорове навантаження. У процесі програмування, обробки графічної інформації та читання з екрана монітора працівник тривалий час фокусує зір на об'єктах, розташованих на близькій відстані. Найбільш розповсюдженим наслідком є синдром комп'ютерного зору, що проявляється у вигляді наступних симптомів: сухість очей, сльозотеча, біль в очах. Виникнення цих симптомів пов'язане зі зниженням частоти кліпання, надмірною яскравістю екрану. Для зменшення зорового навантаження рекомендується регулярно виконувати вправи для очей, наприклад, дуже ефективною є гімнастика для очей, яка стимулює кровообіг і знімає напруження.

3.2. Організація робочого місця.

Організація робочого місця працівника, що здійснює професійну діяльність за персональним комп'ютером, має відповідати вимогам ергономіки, техніки безпеки та санітарно-гігієнічним нормативам. Належно організоване робоче місце сприяє зниженню впливу шкідливих чинників, зменшенню ризику виникнення професійних захворювань і підвищенню продуктивності праці.

Робоче місце повинно бути укомплектоване меблями та обладнанням, конструкція яких забезпечує можливість індивідуального регулювання відповідно до антропометричних особливостей працівника. Висота робочого столу має становити від 680 до 760 мм, а глибина стільниці – не менше 800мм для розміщення монітора на рекомендованій від очей відстані. Офісне крісло має мати регульовану висоту сидіння, спинку з підтримкою поперекового відділу хребта, підлокітники, механізм зміни кута нахилу спинки та обертання. Конструкція крісла має забезпечувати підтримку ніг у положенні, за якого колінні суглоби утворюють кут 90-110 градусів, а ступні повністю спираються на підлогу або спеціальну підставку.

Розташування монітора має забезпечувати кут зору в межах 30-35 градусів нижче горизонтальної лінії погляду. Відстань від очей до екрана має бути не меншою за 600 мм. Центр екрана бажано розташувати трохи нижче рівня очей, щоб уникнути надмірного напруження м'язів шиї та очей. Монітор повинен мати чітке зображення без мерехтіння, з достатнім рівнем контрастності та яскравості, що не викликає зорового дискомфорту.

До обов'язкових умов також належить відповідне освітлення. Загальна освітленість у робочій зоні повинна бути в межах 300-500 лк. Робоче місце повинно бути оснащене комбінованим освітленням: загальним (стельовим), та місцевим (настільна лампа). Джерела світла не повинні створювати відблисків на поверхні екрана.

Параметри мікроклімату приміщення мають відповідати встановленим нормам, адже вони безпосередньо впливають на самопочуття, працездатність та здоров'я працівника. Оптимальною вважається температура повітря в межах 21

або 23 градуси С, що забезпечує комфорт під час тривалого перебування у приміщенні без перегріву чи переохолодження організму. Важливим показником також є відносна вологість повітря, яка має знаходитися в межах 40-60%. Надмірна сухість повітря може призводити до пересихання слизових оболонок, подразнення очей і проблем із диханням, тоді як надлишкова вологість створює сприятливе середовище для розвитку цвілі та бактерій.

Для підтримання стабільного та здорового мікроклімату рекомендується використовувати сучасні системи вентиляції, кондиціонування та зволоження повітря, які забезпечують постійний обмін повітря, фільтрацію від пилу й алергенів, а також підтримку температурного режиму на комфортному рівні. Регулярне провітрювання приміщення, контроль за рівнем вологості та використання побутових приладів для очищення повітря також позитивно впливають на загальний стан мікросередовища та сприяють підвищенню ефективності праці.

Електробезпека є критично важливим елементом забезпечення безпечних умов праці працівників, які здійснюють професійну діяльність із використанням комп'ютерної техніки. Оскільки більшість сучасних офісних та виробничих процесів пов'язана з використанням електроенергії, важливо усвідомлювати наявність потенційних ризиків, пов'язаних із порушенням правил експлуатації електрообладнання. Серед основних небезпечних і шкідливих факторів, що можуть виникати у таких умовах, слід виділити ризик ураження електричним струмом, виникнення коротких замикань, перегрів електронних компонентів, займання проводки, а також вихід з ладу мережевих пристроїв.

Особливу небезпеку становлять ситуації, пов'язані з використанням електромережі або порушенням ізоляції проводів. З метою запобігання виникненню нещасних випадків, травм, а також аварійних ситуацій, усі електричні пристрої, що використовуються в процесі роботи, повинні бути технічно справними та проходити регулярні перевірки відповідно до графіку технічного обслуговування. Обладнання має мати заводські маркування, технічну документацію, інструкції з експлуатації, сертифікати та відповідати вимогам

чинних нормативно-технічних документів, зокрема стандартів безпеки. Окрім цього, робочі місця повинні бути обладнані справними заземленнями та розетками, а також елементами захисту, такими як автоматичні вимикачі або пристрої захисного відключення.

Працівники повинні бути проінформовані щодо основних правил електробезпеки та проходити відповідні інструктажі, що дозволить своєчасно виявляти небезпечні ситуації та реагувати на них належним чином. Такий комплекс заходів сприяє створенню безпечного виробничого середовища та мінімізує ризики, пов’язані з експлуатацією електричного обладнання.

3.3 Пожежна безпека

Пожежна безпека є невід’ємною складовою загальної системи охорони праці на підприємстві, у тому числі в умовах організації праці програмістів. Незважаючи на відсутність безпосереднього контакту з відкрити джерелами вогню, діяльність пов’язана з експлуатацією значної кількості електронного обладнання, створює потенційно небезпечні умови щодо виникнення пожежонебезпечних ситуацій.

З метою запобігання виникненню пожежі всі електричні пристрої повинні бути розміщені з дотриманням встановлених нормативів на достатній відстані від легкозаймистих предметів. Всі експлуатовані електротехнічні засоби повинні проходити регулярну перевірку на технічну справність та відповідати чинним стандартам і вимогам електробезпеки.

У кожному робочому приміщенні необхідно передбачити наявність первинних засобів пожежогасіння, що відповідають типу можливих джерел займання. Зокрема, мають бути встановлені вогнегасники вуглекислого або порошкового типу, призначені для ефективної ліквідації загорянь електроустановок, які перебувають під напругою, без ризику пошкодження обладнання чи ураження електричним струмом. Вогнегасники повинні розміщуватись у доступних місцях, бути промаркованими, мати пломби та інструкції щодо використання, а також проходити регулярні перевірки та технічне обслуговування згідно з чинними нормативами.

Не менш важливою складовою пожежної безпеки є наявність актуальної та зрозумілої схеми евакуації, яка повинна містити чітке позначення основних і запасних маршрутів виходу з будівлі або приміщення. Крім того, на плані обов'язково зазначається місце розташування вогнегасників, пожежних щитів, тривожних кнопок або інших систем сповіщення про пожежу. Схема має бути розміщена на видимому місці, доступному для ознайомлення всім працівникам, а також відповідати актуальним стандартам пожежної безпеки.

Крім візуального плану , важливо забезпечити працівникам інструктажі з пожежної безпеки та проводити регулярні навчання з відпрацювання порядку дій у разі виявлення загоряння, задимлення чи іншої надзвичайної ситуації. Чітка організація таких заходів сприяє своєчасному реагуванню та ефективній евакуації персоналу у разі виникнення небезпечної ситуації, що у підсумку може врятувати життя та зменшити матеріальні збитки.

ВИСНОВКИ

Під час виконання дипломного проекту було реалізовано веб-застосунок під назвою «Task Manager». Для його реалізації були використані: HTML для розмітки, CSS для стилізації, JavaScript для надання функціональності та Firebase для збереження даних.

До функціоналу застосунку входять можливості по додаванню нових карток, у середину колонок, які також можна створювати. Реалізовано систему модальних вікон, завдяки яким можна проводити налаштування, зокрема для карток. Також для карток було реалізовано систему переміщення між колонками.

Для колонок також було реалізовано модальне вікно з можливостями налаштувань, розроблено можливість перенесення колонок між дошками із збереженням усього контенту, який є всередині, включно із налаштуваннями. Окрім можливості переміщувати колонки між дошками, також реалізовано можливість пересувати колонки всередині дошок, що дозволяє переміщувати їх на будь яку позицію.

Було реалізовано функціонал для меню, яке є частиною колонки та забезпечує ряд можливостей для налаштувань.

Реалізована система дошок, яка забезпечує функції додавання нових дошок, видалення старих, можливість перейменування вже існуючих, та здійснювати навігацію між ними.

Створено систему авторизації, яка дозволяє уникати плутанини при збереженні даних, оскільки для кожного користувача дані зберігаються окремо.

Загалом додаток демонструє гарну функціональну базу, яку можна надалі розширювати та масштабувати, додаючи новий функціонал. Як приклад це може бути реалізація можливості створення спільних дошок, до яких можна буде додавати користувачів, і працювати над планами у команді. Можна піти ще далі і реалізувати систему ролей для того, щоб розбивати команду на окремі групи, які працюватимуть над окремими списками задач. А завдяки використанню Firebase існує можливість для легкого впровадження чатів, як зручного інструменту для комунікації команди прямо у застосунку.

ПРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦФЙНИХ ДЖЕРЕЛ

1. Фрімен Е., Робсон Е. Вивчаємо JavaScript. Поглиблений курс. – Львів: Видавництво Старого Лева, 2020. – 640с.

2. Сучасний підручник з JavaScript. – [Електронний ресурс]. – Режим доступу: https://uk.javascript.info/ – Дата звернення: 05.05.2025

3. Firebase Official Documentation. – [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs – Дата звернення: 14.05.2025

4. Джейкобс К., Левінс А. JavaScript: від основ до просунутих технік. – Київ: Видавництво «Книжковий клуб», 2019. – 480с.

5. Сміт Джон. HTML та CSS: Розробка і дизайн веб-сторінок. – Київ: Видавництво «Буква», 2020. – 432с.

6. Сушко С. Веб-програмування з HTML, CSS та JavaScript: навчальний посібник. – Київ: Видавничий центр КНУ, 2022. – 276с.

7. Фелке М. CSS: секрети вебдизайну. – Київ: Видавництво «Наш формат», 2020. – 384с.

8. Шевко В. Сучасний JavaScript для розробників. – Київ: Видавництво «Ранок», 2022. – 416с.

9. Коваленко О. Веб-додатки на чистому JavaScript. – Харків: Видавництво «Фактор», 2021. – 320с.

10. Гриценко М. Інтерактивні веб-інтерфейси. – Київ: Видавництво «Основа», 2022. – 288с.

Додаток А. Лістинг коду мовою програмування JavaScript

```
// Лістинг файлу ініціалізації, main.js:
import { auth, database } from './firebase-config.js';
import { BoardManager } from './modules/board/boardmanager.js';

const componentInitializers = {
    menus: [
        () => import('./modules/menu/sidemenu.js').then(m => m.initSideMenu()),
        () => import('./modules/menu/menu.js').then(m => m.initColumnMenus()),
        () => import('./modules/menu/positionmenu.js').then(m =>
m.initMenusPosition())
    ],
    columns: [
        () => import('./modules/column/addcolumn.js').then(m =>
m.initNewColumnButton()),
        () => import('./modules/column/renamecolumn.js').then(m =>
m.initColumnRenaming()),
        () => import('./modules/column/dragdropcolumn.js').then(m =>
m.setupColumnDragFeature()),
        () => import('./modules/column/positioncolumn.js').then(m =>
m.initColumnsPosition())
    ],
    cards: [
        () => import('./modules/card/addcard.js').then(m => {
            m.initAddCardButtons();
            m.setupMenuAddCard();
        }),
        () => import('./modules/card/dragdropcard.js').then(m => {
            m.initDragAndDropForAllColumns();
            m.updateAllCardsDrag();
        })
    ],
    modals: [
        () => import('./modules/modal/cardmodal.js').then(m => m.initCardModal()),
        () => import('./modules/modal/columnmodal.js').then(m =>
m.initColumnModal()),
        () => import('./modules/modal/registermodal.js').then(m =>
m.initRegisterModal())
    ]
};

const appState = {
    initialized: false,
    user: null,
    boardManager: null
};

database.ref('.info/connected').on('value', (snapshot) => {
    console.log(snapshot.val() ? 'Firebase: Connected' : 'Firebase: Disconnected');
```

```
});

async function initializeApp(user) {
    if (appState.initialized) return;

    console.log('Starting app initialization...');

    try {
        appState.boardManager = new BoardManager();
        window.boardManager = appState.boardManager;

        if (user) {
            await appState.boardManager.loadBoards();
        }

        await initializeComponents();

        appState.initialized = true;
        console.log('App initialized successfully');

        setTimeout(() => {
            updateUI();
            setupMutationObserver();
        }, 300);

    } catch (error) {
        console.error('Initialization failed:', error);
    }
}

async function initializeComponents() {
    await executeInitializers(componentInitializers.menus);
    await executeInitializers(componentInitializers.columns);
    await executeInitializers(componentInitializers.cards);
    await executeInitializers(componentInitializers.modals);
}

async function executeInitializers(initializers) {
    for (const init of initializers) {
        try {
            await init();
        } catch (error) {
            console.error('Initializer failed:', error);
        }
    }
}

function updateUI() {
    Promise.all([
        import('./modules/column/positioncolumn.js').then(m =>
m.updateAllColumnsPosition()),
```

```
        import('./modules/card/dragdropcard.js').then(m => m.updateAllCardsDrag())
    ]).catch(console.error);
}

function setupMutationObserver() {
    const observer = new MutationObserver((mutations) => {
        if (document.querySelector('.column')) {
            updateUI();
            reinitializeColumnInteractivity();
        }
    });

    observer.observe(document.body, {
        childList: true,
        subtree: true,
        attributes: false,
        characterData: false
    });
}

function reinitializeColumnInteractivity() {
    Promise.all([
        import('./modules/column/renamecolumn.js').then(m =>
m.initColumnRenaming()),
        import('./modules/card/addcard.js').then(m => m.initAddCardButtons())
    ]).catch(console.error);
}

function resetApp() {
    const appContent = document.getElementById('app-content');
    if (appContent) appContent.style.display = 'none';

    if (appState.boardManager) {
        appState.boardManager.clear();
    }
    appState.initialized = false;
    appState.user = null;
}

auth.onAuthStateChanged((user) => {
    appState.user = user;

    if (user) {
        console.log('User authenticated:', user.uid);
        document.getElementById('app-content').style.display = 'block';
        document.querySelector('.modal-auth').style.display = 'none';
    } else {
        console.log('User not authenticated');
        resetApp();
        showLoginScreen();
    }
```

```
});

function showLoginScreen() {
    const authModal = document.querySelector('.modal-auth');
    if (authModal) {
        authModal.style.display = 'flex';
        const loginForm = authModal.querySelector('.login-form');
        const registerForm = authModal.querySelector('.register-form');
        if (loginForm) loginForm.style.display = 'block';
        if (registerForm) registerForm.style.display = 'none';
    }
}

document.addEventListener('DOMContentLoaded', () => {
    console.log('DOM fully loaded');

    showLoginScreen();

    initializeApp(null);
});

window.addEventListener('load', () => {
    console.log('Window fully loaded');
    setTimeout(() => {
        if (appState.boardManager && auth.currentUser) {
            appState.boardManager.renderCurrentBoard();
        }
    }, 300);
});

export function initApp() {
    console.log('App initialization requested');
}
```

Додаток Б. Слайди презентації

Дипломний проєкт на тему: «Розробка програмного забезпечення для відстеження ступеня готовності виконання проєкту»

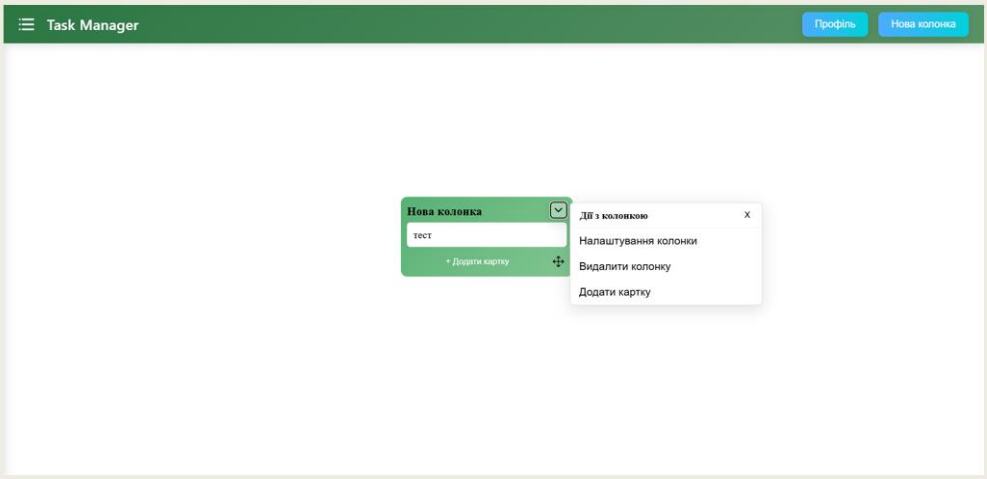
Воробйов Сергій, 4РП-08

Процес обміну даними на платформі Firebase

```
task-manager
├── firebase
│   └── hosting
├── public
│   ├── CSS
│   │   ├── board.css
│   │   ├── cards.css
│   │   ├── columns.css
│   │   ├── header.css
│   │   ├── main.css
│   │   ├── menu.css
│   │   ├── modals.css
│   │   └── sidebar.css
│   ├── js
│   │   ├── modules
│   │   │   ├── board
│   │   │   │   └── boardmanager.js
│   │   │   ├── card
│   │   │   │   ├── addcard.js
│   │   │   │   └── dragdropcard.js
│   │   │   ├── column
│   │   │   │   ├── addcolumn.js
│   │   │   │   ├── dragdropcolumn.js
│   │   │   │   ├── positioncolumn.js
│   │   │   │   └── renamecolumn.js
│   │   │   ├── menu
│   │   │   │   ├── menu.js
│   │   │   │   ├── positionmenu.js
│   │   │   │   └── sidemenu.js
│   │   │   └── modal
│   │   │       ├── cardmodal.js
│   │   │       ├── columnmodal.js
│   │   │       └── registermodal.js
│   │   ├── firebase-config.js
│   │   └── main.js
│   ├── index.html
│   └── png
├── firebaseserc
├── gitignore
└── firebase.json
```

Файлова структура застосунку

Зовнішній вигляд інтерфейсу



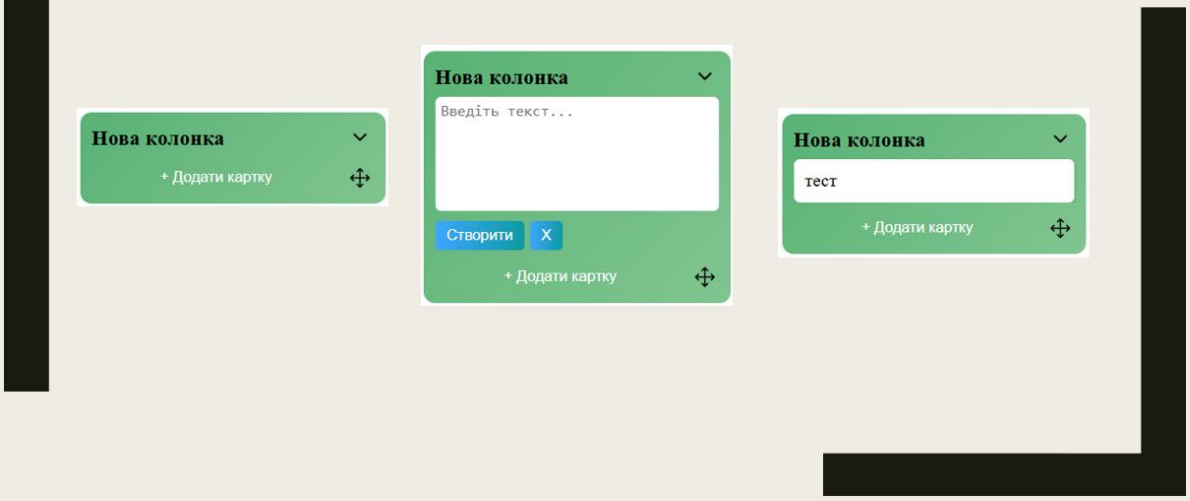
```
export async function createCard(column, cardText = "") {
    const boardId = window.boardManager?.currentBoardId;
    const columnId = column.dataset.columnId;

    if (!boardId || !columnId) {
        return null;
    }

    const cardRef = database.ref(`boards/${boardId}/columns/${columnId}/cards`).push();
    const cardId = cardRef.key;

    await cardRef.set({
        text: cardText.trim(),
        color: "#ffffff",
        tags: "",
        deadline: "",
        status: "not_done",
        createdAt: new Date().toISOString()
    });

    const card = document.createElement("div");
    card.classList.add("card");
    card.dataset.cardId = cardId;
    card.innerHTML = `
        <p class="card-text">${cardText}</p>
    `;

    initDragForCard(card);

    if (window.boardManager) {
        await window.boardManager.saveCurrentBoard();
    }

    return card;
}
```

Скріншот коду для створення карток

Ключовими моментами є:

- виклик await cardRef.set;
- створення нового div елемента.
- виклик window.boardManager

Демонстрація процесу створення карток



Код для створення нових колонок

Перша частина:

```
export async function createNewColumn() {
    const boardId = window.boardManager?.currentBoardId;
    if (!boardId) {
        return null;
    }

    const columnRef = database.ref(`boards/${boardId}/columns`).push();
    const columnId = columnRef.key;

    const columnsSnapshot = await database.ref(`boards/${boardId}/columns`).once('value');
    const columnCount = columnsSnapshot.val() ? Object.keys(columnsSnapshot.val()).length : 0;

    const columnData = {
        name: "Нова колонка",
        position: columnCount,
        cards: {},
        createdAt: new Date().toISOString()
    };

    await columnRef.set(columnData);

    const board = document.querySelector(".board");
    const newColumn = document.createElement("div");
    newColumn.classList.add("column");
    newColumn.dataset.columnId = columnId;
```

Друга частина:

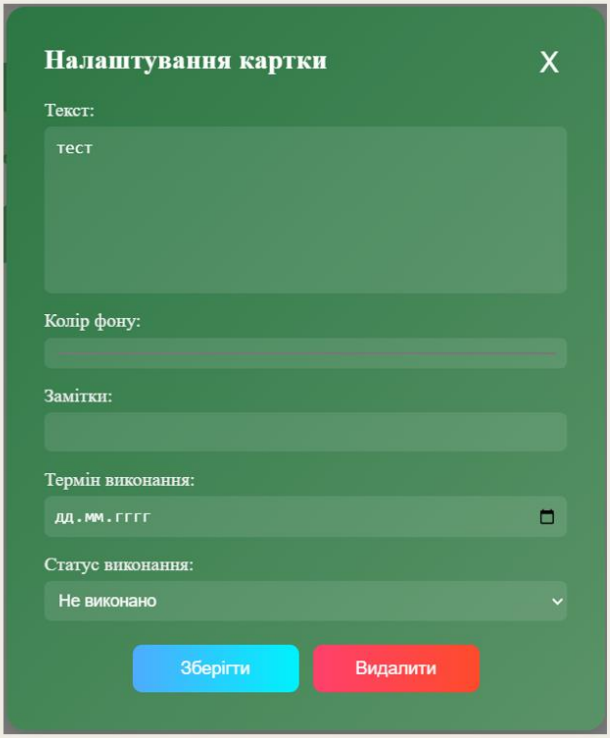
```
newColumn.innerHTML = `
    <div class="column-header">
        <h3 class="column-title" contenteditable="true">${columnData.name}</h3>
        <button class="menu-button" aria-expanded="false" aria-controls="list">
            <img class="menu-png" src="./png/menu.png" alt="">
        </button>
        <ul class="menu" id="list">
            <div class="menu-header">
                <h3 class="menu-title">Дії з колонкою</h3>
                <button class="menu-clouse">X</button>
            </div>
            <button class="menu-item item1">Налаштування колонки</button>
            <button class="menu-item item3">Видалити колонку</button>
            <button class="menu-item item4">Додати картку</button>
        </ul>
    </div>
    <div class="card-list"></div>
    <div class="button-container">
        <button class="add-card">+ Додати картку</button>
        <button class="drag-button">
            <img class="drag-button-png" src="./png/movecolumn.png" alt="" draggable="false">
        </button>
    </div>
`;
```

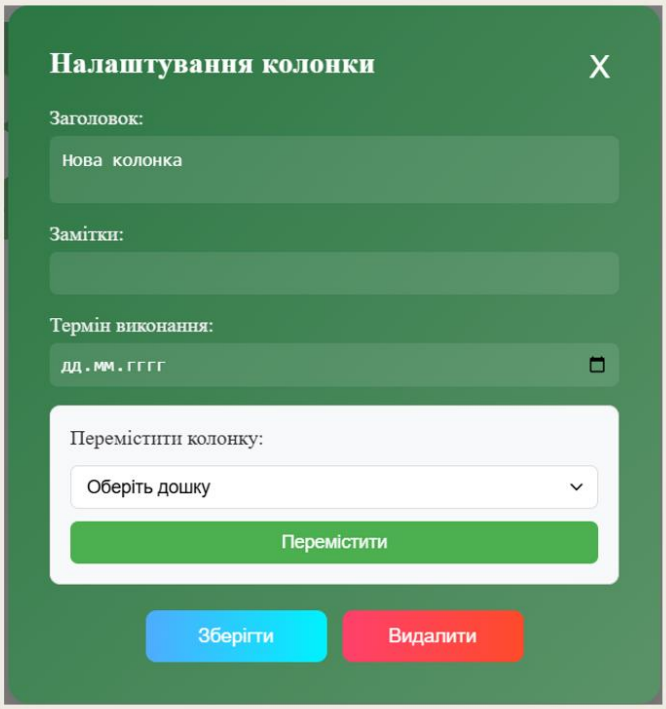
Третя частина:

```
    board.appendChild(newColumn);
    initColumnEventHandlers(newColumn);

    if (window.boardManager) {
        await window.boardManager.saveCurrentBoard();
    }

    return newColumn;
}
```

Модальне вікно для картки

Використовується для налаштування зовнішнього виду картки та редагування змісту





Модальне вікно для колонки

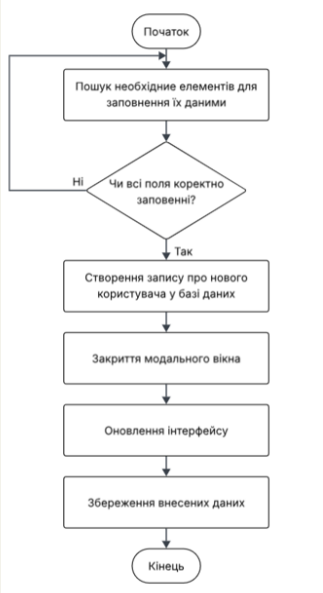
Використовується для налаштування колонок

Зовнішній вигляд бічного меню та його функціонал

```
async  switchBoard(boardId) {
    if (!this.boards[boardId]) return;

    if (this.currentBoardId) {
        await this.saveCurrentBoard();
    }

    const boardSnapshot = await database.ref(`boards/${boardId}`).once('value');
    if (!boardSnapshot.exists()) return;

    this.boards[boardId] = boardSnapshot.val();
    this.currentBoardId = boardId;

    await database.ref(`users/${this.userId}/lastActiveBoard`).set(boardId);

    this.renderCurrentBoard();
    this.renderBoardList();

    await this.reinitializeHandlers();
    if (typeof initApp === 'function') initApp();
}
```

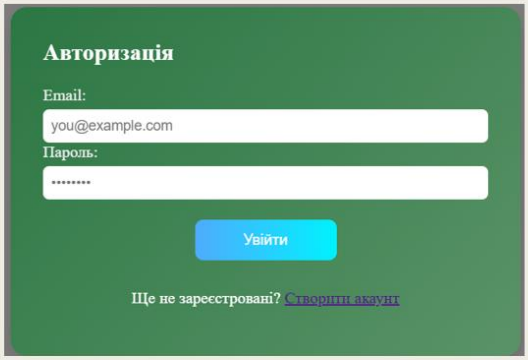
Ключовим моментом коду є взаємодія з базою даних

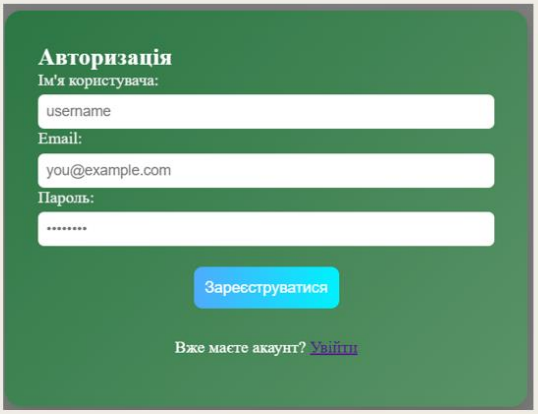
Реєстрація нового користувача алгоритм та код




```javascript
registerBtn.addEventListener('click', async (e) => {
    e.preventDefault();
    const name = registerForm.querySelector('.name').value;
    const email = registerForm.querySelector('.email').value;
    const password = registerForm.querySelector('.password').value;

    if (!name || !email || !password) {
        alert('Заповніь всі поля');
        return;
    }

    const userCredential = await auth.createUserWithEmailAndPassword(email, password);
    const user = userCredential.user;

    await database.ref('users/' + user.uid).set({
        username: name,
        email: email,
        createdAt: new Date().toISOString(),
        boards: {}
    });

    authModal.style.display = 'none';
    clearForms();
    updateAuthUI(true);
});
```

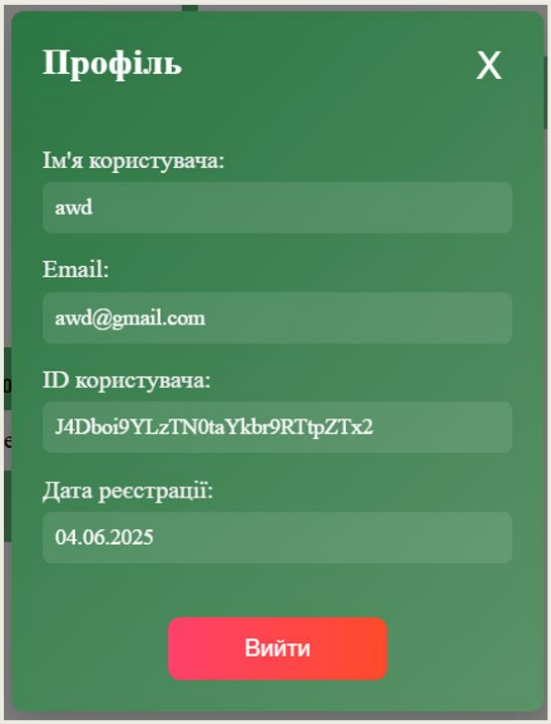
Вікна для входу та створення нового акаунту





Вікно профілю користувача

Використовується для відображення інформації про акаунт



Структура збереження даних

Для дошок

Для користувачів

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Воробйова Сергія Сергійовича

(прізвище, ім`я та по батькові)

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітня програма *«Розробка програмного забезпечення»*

Керівник дипломного проекту (роботи) *Залапін Олексій Ігорович*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка програмного забезпечення для відстеження ступеня готовності виконання проєкту*

Обсяг розрахунково-пояснювальної записки *63* сторінок

Обсяг графічної (презентаційної) частини *15* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений на рецензію дипломний проект відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячений розробці програмного забезпечення для відстеження ступеня готовності виконання проєкту та складається з пояснювальної записки, додатку з програмним кодом та мультимедійної презентації

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу, економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора КТ. Економічний розділ проекту містить розрахунок витрат на НДР та реалізацію проекту.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 15 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять ілюстративні схеми, скріншоти роботи програмного застосунку, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання графічної частини проекту та пояснювальної записки добра, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту *Використання Firebase як рішення для бази даних і хостингу, а також застосування модульної архітектури JavaScript з підтримкою ES6-функцій (async/await, import/export) демонструє обізнаність з актуальними підходами у веброзробці. У проєкті чітко визначені модулі за функціональними блоками, кожен із яких має власні файли та відповідні функції. Це спрощує масштабування і підтримку.*

д) основні недоліки дипломного проекту *Деякі описи форматування містять зайву вербозність (наприклад, поетапний опис кожного селектора або класу з CSS), що не завжди має практичну цінність. Хоча логіка додатку реалізована детально, відсутній опис автоматизованих або юніт-тестів. У багатьох розділах зустрічається повтор одного й того ж функціоналу або структурної логіки.*

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Добре*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові рецензента *к.т.н. Рудніченко Микола Дмитрович*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис: _____

«23» 06 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Воробйова Сергія Сергійовича

(прізвище, ім`я та по батькові)

Спеціальність: *121 "Інженерія програмного забезпечення"*

Освітньо-професійна програма: *«Розробка програмного забезпечення»*

Тема дипломного проекту: *Розробка програмного забезпечення для відстеження ступеня готовності виконання проекту*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка містить 76 сторінки. У пояснювальній записці наведено етапи розробки програмного забезпечення для відстеження ступеня готовності виконання проекту. Графічна частина складається з 15 слайдів мультимедійної презентації, які також містять схеми, діаграми та алгоритми, що передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.*

б) самостійність роботи над проектом: *Протягом всього строку дипломного проектування та переддипломної практики здобувач освіти Воробйов С.С. поступово та послідовно виконував всі етапи розробки. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти Воробйов С.С. під час роботи над дипломним проектом вивчив достатню кількість літературних джерел та матеріалів за даною тематикою. Вважаю, що теоретична підготовка дипломника добра і він готовий до захисту дипломного проекту*

г) вміння розв'язувати виробничі та конструкторські питання *Під час дипломного проектування здобувач освіти Воробйов С.С. мав змогу самостійно приймати окремі рішення з реалізації програмної частини для відстеження ступеня готовності виконання проекту та показав вміння організовано працювати над поставленим завданням, розробляти структурні схеми та програмні зв'язки із застосуванням сучасних комп'ютерних програмних засобів, таких як Visual Studio code, HTML, CSS, JavaScript, а також готувати презентаційні та звітні матеріали.*

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові керівника дипломного проекту *Залапін Олексій Ігорович*

Місце роботи і посада керівника дипломного проекту *ВСП "Одеський технічний фаховий коледж ОНТУ", викладач спецдисциплін комісії комп'ютерних технологій та програмної інженерії.*

Підпис

«16» червня 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Воробйов Сергій Сергійович
здобувач освіти гр. 4РП-08, та

Залапін Олексій Ігорович,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка програмного забезпечення для відстеження ступеня готовності виконання проекту» (автор роботи – Воробйов С.С., керівник роботи – Залапін О.І.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Воробйов С.С. /

Керівник _____ / Залапін О.І. /

«16» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4РП-08

Воробйова Сергія Сергійовича

на тему *Розробка програмного забезпечення для відстеження ступеня готовності виконання проекту*

Висновок відповідальної особи за проведення нормоконтролю:

пояснювальна записка до дипломного проєкту виконана з некритичними порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *20.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 18.06.2025 р. значення коефіцієнту подібності в роботі становить 8,85%, коефіцієнт цитування – 0,61%.*

(підпис) *20.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Воробйова С.С.* (П.І.Б.)

проведена « *20* » *червня* 2025 р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/18/2025
Дата редагування 6/18/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
Odesa Technical Professional College of Odesa National University of Technology

Заголовок
Розробка програмного забезпечення для відстеження ступеня готовності виконання проєкту

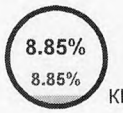
Автор
Воробйов Сергій Сергійович

Науковий керівник / Експерт
Залапін Олексій Ігорович

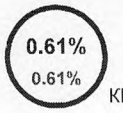
підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КЦ

25
Довжина фрази для коефіцієнта подібності 2

12233
Кількість слів

96126
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	19
Інтервали	A→	6
Мікропробіли	·	177
Білі знаки	ß	0
Парафрази (SmartMarks)	a	38

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	81 0.66 %
2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	66 0.54 %
3	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	64 0.52 %
4	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	60 0.49 %
5	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	50 0.41 %

6	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	46 0.38 %
7	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	37 0.30 %
8	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	34 0.28 %
9	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	32 0.26 %
10	https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download	28 0.23 %

з домашньої бази даних (0.37 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	29 (3) 0.24 %
2	Розробка web-застосунку для генерації повідомлень із використанням технологій штучного інтелекту [redacted] 6/14/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	16 (2) 0.13 %

з програми обміну базами даних (0.17 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Онлайн курс вивчення Python [redacted] 5/26/2025 Separate structural unit "Technical Vocational College of Lutsk National Technical University" (Separate structural unit "Technical Vocational College of Lutsk National Technical University")	11 (1) 0.09 %
2	ФКНТ_2023_123_Попович_М.В [redacted] 7/11/2024 Ukrainian national aviation university (Ukrainian national aviation university)	10 (1) 0.08 %

з Інтернету (8.31 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content**	**515 (18) 4.21 %**
2	**https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download**	**108 (5) 0.88 %**
3	**https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download**	**57 (2) 0.47 %**
4	https://card-file.ontu.edu.ua/bitstreams/e4afae26-0a7e-4a4d-afc2-94341838de2a/download	53 (5) 0.43 %
5	**https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download**	**37 (2) 0.30 %**
6	**https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download**	**33 (2) 0.27 %**
7	https://card-file.ontu.edu.ua/bitstreams/21173711-5b67-4b87-b17f-6302c25e7a31/download	31 (2) 0.25 %
8	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	29 (2) 0.24 %
9	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	29 (3) 0.24 %

10	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	27 (2) 0.22 %
11	https://card-file.ontu.edu.ua/bitstreams/dfa57ac3-98fa-4c22-86e7-0549d1254d89/download	25 (2) 0.20 %
12	https://card-file.ontu.edu.ua/server/api/core/bitstreams/c63b91ba-d04f-4715-890d-b16277695c7e/content	22 (2) 0.18 %
13	https://card-file.ontu.edu.ua/bitstreams/12d5c0ab-e979-48f2-a8ec-d5fc31f71fd5/download	18 (2) 0.15 %
14	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	13 (2) 0.11 %
15	https://card-file.ontu.edu.ua/bitstreams/8f088d70-9465-490c-8fa6-2eb74516c620/download	10 (1) 0.08 %
16	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	5 (1) 0.04 %
17	https://card-file.ontu.edu.ua/bitstreams/55e2b8f2-7d3c-4235-99fc-2be51199b96d/download	5 (1) 0.04 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма: **«Розробка програмного забезпечення»**
Група: 4РП- 08

Дипломний проєкт
здобувача освіти денної форми навчання
РП.08.05.000.ДП

ВОРОБЙОВА
СЕРГІЯ СЕРГІЙОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма: **«Розробка програмного забезпечення»**
Група: 4 **РП- 08**

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проєкту на тему:

_____

_____

Проєктний матеріал складається з пояснювальної записки на _____ сторінках та графічного (презентаційного) матеріалу на _____ аркушах (слайдах)

Дипломник _____ (Воробйов С.С.)
Керівник _____ (Залапін О.І.)

Консультанти: з економічного розділу_____ (Канський М.Ю.)
з розділу **охорони праці та техніки безпеки**_____ (**Чорновол Н.І.**) з нормоконтролю
_____(Петрашова **В.І.**) **старший** консультант _____(Кривченко Ю. В.)
До захисту допущений Голова циклової комісії _____ (**Кривченко Ю.В.**)
Завідувач відділення _____ (Краснокутська К.Г.)

Захист «____» _____ 2025 **р. Протокол** ЕК No_____ **Оцінка** ЕК _____

Секретар ЕК _____