Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Розробка веб-додатків з використанням JavaScript фреймворків Vue та React»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1811»

_____ /Анастасія БІБЕ/
(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /доц. Вадим АНДРЮЩЕНКО/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ /доц. Олена КУРОП'ЯТНИК/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Development of web applications using JavaScript Vue and React frameworks»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ1811: /Anastasiia BIBE/

Scientific Supervisor: /Vadym ANDRIUSHCHENKO/

Normative controller: /Olena KUROPIATNYK/

Dnipro – 2022

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)

Дата _____

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра
студенту Бібе Анастасії Євгеніївни
1. Тема роботи: «Розробка веб-додатків з використанням JavaScript фреймворків Vue та React»
Керівник роботи: Андрющенко Вадим Олександрович, доцент
затверджені наказом № 77 ст від 08.12.2021
2. Строк подання студентом роботи: __.__.202_ р.
3. Вихідні дані до роботи: _____

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
Вступ
Збір вимог до програмного забезпечення
Огляд програмних аналогів
Огляд програмного продукту «Trello»
Огляд платформи «Jira»
Зовнішнє проєктування
Внутрішнє проєктування
Тестування методом «білої скриньки»
Налагодження програми
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Презентація
Відео роботи програми

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі	05.01.2022	
2	Огляд літератури та аналіз аналогів	24.01.2022	
3	Розробка структур вхідних і вихідних даних	02.02.2022	
4	Визначення вимог до програми. Вибір та обгрунтування мови програмування	10.02.2022	
5	Узгодження та затвердження ТЗ	18.02.2022	
6	Розробка та програмування логіки програми	01.03.2022	
7	Розробка і реалізація інтерфейсу користувача	20.03.2022	
8	Відлагодження програми	24.03.2022	
9	Подання кваліфікаційної роботи до кафедри	10.06.2022	
10	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	21.06.2022	

Студент _____ Анастасія БІБЕ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ доц. Вадим АНДРЮЩЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка складається з 7 розділів:

– вступ – в даному розділі описується сутність розробки, її актуальність.

Складається з 1 сторінки;

– збір вимог до програмного забезпечення – у цьому розділі описуються
аналоги програми та література по даній предметній області.

Складається з 15 сторінок;

– зовнішнє і внутрішнє проектування – у цьому розділі проведений огляд
вхідних і вихідних даних, формалізація задачі, розробка фізичного
проекту, наводиться опис об’єктно-орієнтованого проектування,
проектування інтерфейсу користувача, ескізи форм, аналіз проекту,
проектування динаміки системи, вибір мови програмування. Складається
з 8 сторінок;

– тестування та налагодження – включає в себе вибір стратегії тестування,
опис тестів методами «чорного» та «білого» ящика. Також аналіз
помилок, їх вплив на систему та вирішення проблеми.

Складається з 11 сторінок;

– висновки.

Складається з 1 сторінки;

– список літератури – включає в себе бібліографічний список використаної літератури.

Складає 1 сторінку;

– додатки – містить технічне завдання і код програми.

Складається з 33 сторінок.

Кількість таблиць: 4 штуки.

Кількість рисунків: 19 штук.

Ключові слова: проєктування, Kanban, Scrum, “TodoDashboard”.

ЗМІСТ

ВСТУП

В нашому часі, коли комп’ютерні технології розвиваються дуже стрімко, однією з перших задач людини є розвиток відповідно до оточуючого світу. Кожен з нас намагається максимально «діджиталізуватися», тобто перевести своє життя у цифровий вимір. Більшість з нас вже майже не уявляють своє життя без онлайн послуг, ми замовляємо їжу онлайн, купуємо одяг та товари для дому і навіть платимо рахунки за комунальні послуги. У всьому цьому нам допомагають веб-сайти та веб-додатки.

Наразі існує багато конструкторів для веб-сайтів, такі як Wordpress, використання яких є більш дешевим та не потребує кваліфікованих програмістів. Проте використання таких конструкторів, на відміну від написання сайту на мові програмування JavaScript, не дозволяє створити якісний веб-сайт або веб-додаток. JavaScript дає змогу прописувату у коді скадну логіку для веб-додатків, а також якісно кастомізувати об’єкти на сайті або у додатку. Ще одним аргументом на користь JavaScript є її популярність серед програмістів: JavaScript вже перевершує C# за рейтингом, і поступається тільки Java. Також згідно з SitePoint, JavaScript є більш оплачуваною навіть за C#, що також є сильним аргументом на користь JavaScript.

Задачею даного дипломного проекту є написання веб-додатку на мові JavaScript з використанням фреймворку Vue.js та бібліотеки React.js., що буде відображати задачі, які треба зробити, їх стан та час на виконання. Такий додаток допоможе команді відслідковувати прогрес роботи та оцінити продуктивність команди. Все що потребується від користувача – це створити певну задачу та встановити час на виконання. Кінцевим користувачем програмного засобу може бути як компанія так і проста особа.

1. ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Огляд програмних аналогів

Серед програмних аналогів для ведення обліку задач можна виділити такі як «Trello» та «Jira», що використовуються для керування проєктами.

1.1.1. Огляд програмного продукту «Trello»

Trello – хмарна програма для керування проєктами невеликих груп, розроблена Fog Creek Software. Trello використовує парадигму для управління проєктами, відому як канбан.

Trello дозволяє організувати персональну роботу або роботу невеликої команди без додаткових зусиль і за мінімальний час. Основною перевагою Trello є можливість бачити кілька одночасно запущених проектів та їх стан у поточний момент часу. Якщо є потреба керувати групою розробників або інших виконавців, які працюють над проектами з кінцевою датою виконання або фіксованою метою, то дана система може дати уявлення про перебіг проектів у будь-який час.Картки мають багато можливостей: підтримують коментарі, вкладення, терміни виконання та контрольні списки. Додаток дозволяє проводити обговорення, голосування, завантажувати файли даних, задавати дедлайни, призначати текстові і колірні мітки.

Важливо зауважити, що всі члени робочої групи бачать у реальному часі зміни, що вносяться в проєкт, і можуть спостерігати стани один одного так само в реальному часі – онлайн або офлайн.

Дана система не підходить для масштабних проектів, але для стартапів, домашніх проектів і просто наборів ідей, які вимагають перевірки - це гарне рішення. Trello доступно як у Інтернеті, так і у вигляді програми для iPhone. Використання всіх версій зараз безкоштовне.

1.1.2. Огляд платформи «Jira»

«Jira» – комерційна система відстеження помилок, призначена для організації взаємодії з користувачами, хоча в деяких випадках використовується і для управління проектами. Розроблена компанією Atlassian.

«Jira» є інструментом управління проєктами, який допомагає оптимізувати роботу команди. Принцип роботи сервісу нагадує диспетчер задач у комп'ютері: з його допомогою відстежують запущені процеси (проєкти) і контролюють кількість ресурсів (співробітників).Цей інструмент створювали для відстеження статусу завдань та помилок, але згодом його функціонал розширився. Сьогодні у «Jira» можна керувати процесом розробки від ідеї до запуску готового продукту. Окрім IT-команд, її використовують маркетологи, аналітики, тестувальники та інші спеціалісти.

Для чого може допомогти Jira:

1. Управління вимогами. Вимоги – це вступні дані для роботи над проєктом. Їх пишуть в окремому документі разом із замовником, щоб не виникало розбіжностей у процесі роботи, а розробники могли на них орієнтуватися. Щоб змінити або скласти вимоги для команди, використовують «Jira» у поєднанні з «Confluence» – інструментом спільної роботи. У ньому можна створювати, обговорювати та редагувати документи.
2. Управління товарами. Команди складають у «Jira» дорожні карти – покрокові плани масштабних проектів. Такі карти допомагають налагодити взаємодію між відділами. Наприклад, при грамотному управлінні маркетологи можуть планувати промокампанію паралельно з розробкою, а не чекати на готовий продукт. У дорожніх картах не прописуються докладні завдання та методи виконання, у них розставляються цілі, пріоритети та позначаються залежності роботи одного відділу від роботи іншого.
3. Управління проєктами. «Jira» налаштовується під проекти, тому такий інструмент є корисним для Project-менеджерів. Можна візуально відстежити шлях кожного завдання від створення до результату: генерація ідей та гіпотез, створення прототипу, дизайн, узгодження дизайн-концепту, розробка, створення контенту, тестування.

1.2. Огляд літератури

У кожного програмного забезпечення є життєвий цикл – етапи, через які воно проходить із початку створення до кінця розробки та впровадження. Методологія розробки включає набір методів з управління розробкою: правила, техніки та принципи, які роблять її більш ефективною. Останнім часом набирають популярності так звані гнучкі моделі та методи розробки, що передбачають розбиття проекту на невеликі робочі частини, які мають назву історії [1].

1.2.1. Scrum методологія

Scrum – це методика, яка допомагає командам вести спільну роботу. Як спортивна команда готується до вирішальної гри (до речі, scrum — англ. «битва», елемент гри в регбі), так і команда співробітників компанії повинна отримувати уроки з набутого досвіду, освоювати принципи самоорганізації, працюючи над вирішенням проблеми, та аналізувати свої успіхи та провали, щоб постійно вдосконалюватися. Scrum сприяє цьому.

Методику Scrum найчастіше застосовують команди розробників додатків, але принципи та досвід її використання можна застосувати до командної роботи будь-якого роду. Це одна із причин такої популярності методики. Scrum часто представляють як платформу управління проєктами за методикою Agile. Учасники команди Scrum проводять збори, використовують

спеціальні інструменти та беруть на себе особливі ролі, щоб організувати роботу та керувати нею.

Платформа.

Поняття Scrum та Agile часто плутають, тому що Scrum будується навколо ідеї про постійне вдосконалення, яке є головним принципом Agile. І все ж таки Scrum — це методика роботи, а Agile — це спосіб мислення. Перейти на Agile не так просто: вся команда має прагнути змінити свій підхід до створення цінності клієнтам. Але можна просто почати використовувати методику, таку як Scrum. Це спрямує мислення у потрібне русло і допоможе практикувати принципи Agile у повсякденному спілкуванні та роботі.

Методика Scrum за своєю сутністю є евристичною. В її основі лежить постійне навчання та адаптація до мінливих факторів. Згідно з Scrum, команда не знає всього на початку проекту, але розвиватиметься, вивчаючи уроки з досвіду. У структурі Scrum закладена та свобода, з якою команди пристосовуються до змінних умов та вимог користувачів. Робочий процес передбачає зміну пріоритетів та короткі цикли релізу, що сприяє постійному навчанню та вдосконаленню команди.

Структурованість не заважає методології Scrum бути гнучкою. Її можна адаптувати до потреб організації. Існує безліч теорій про те, як слід застосовувати Scrum, щоб досягти успіху. Однак більш ніж десятирічний досвід сприяння agile-командам у виконанні роботи за допомогою продуктів Atlassian підказує, що незалежно від того, яку методологію ви виберете, її головними принципами має бути ясність комунікації, прозорість та прагнення постійного вдосконалення. Решта ви можете вибирати самі.

Артефакти Scrum.

Спочатку визначимо три артефакти Scrum. Артефакт — це те, що ми створюємо, наприклад, інструмент для вирішення проблеми. У Scrum існує три артефакти: беклог продукту, беклог спринту та інкремент із вашими критеріями готовності. Ці три постійні присутні у кожній команді Scrum, ми постійно до них звертаємось та приділяємо їм час (рис.1).

Беклог продукту – це головний перелік завдань, які потрібно виконати. Його веде власник чи менеджер товару. Це перелік функціональних можливостей, вимог, поліпшень і виправлень, з якого складаються завдання для беклога спринту. Загалом це список завдань команди. Власник продукту постійно звертається до беклогу продукту, змінює в ньому пріоритети і підтримує його актуальність, тому що може з'явитися нова інформація або можуть статися зміни на ринку, через що зникне сенс виконувати наявні завдання або з'являться нові способи вирішення проблем.

Беклог спринту – це список робочих завдань, історій або виправлень багів, відібраних командою розробників для реалізації в поточному циклі спринту. Перед кожним спринтом проводиться збори з планування спринту, де команда вибирає, які завдання з бэклога продукту потрібно виконати у межах спринту. Беклог спринту може бути фіксованим і може змінюватися по ходу спринту.

Однак ніщо не повинно заважати досягненню основної мети спринту - того, чого команда хоче досягти за поточний спринт.

Ікремент (або мета спринту) – це готовий для використання кінцевий продукт за підсумками спринту. У компанії Atlassian прийнято представляти інкремент на демонстрації наприкінці спринту, де команда показує, що вона зробила за спринт. Слово «інкремент» не часто зустрічається у повсякденному житті. Його часто визначають як прийняті в команді критерії готовності продукту, контрольну точку, ціль спринту або навіть повну версію. Все залежить від того, якими критеріями готовності керується ваша команда та як вибираються цілі спринту. Наприклад, деякі команди вважають за краще випускати щось для своїх клієнтів наприкінці кожного спринту. Однак для інших команд це може бути непрактичним. Уявіть, що ви працюєте над серверним продуктом, який можна постачати клієнтам лише раз на три місяці. Ви, як і раніше, можете розбивати роботу на двотижневі спринти, але для вас продукт буде готовий, коли ви завершите роботу над частиною більшої версії, яку плануєте поставити повністю.

Допустимі різні варіанти, навіть коли йдеться про артефакти, яким ваша команда може надавати ту чи іншу форму. Це показує, чому важливо залишатися відкритими для вдосконалення, зокрема для вдосконалення способу ведення артефактів.

Збори чи заходи Scrum.

Трохи більш відомі такі складові методики Scrum, як послідовні заходи, церемонії чи збори, які регулярно проводять команди Scrum. Саме у підході до зборів найпомітніше виявляються різницю між командами. Деяким командам в тягар проводити одноманітні збори; в інших робочі зустрічі є обов'язковими. Якщо ви починаєте знайомство зі Scrum, рекомендується протягом перших двох спринтів провести всі збори, щоб зрозуміти своє ставлення до них. Після цього можна організувати коротку ретроспективу (практика в розробці програмного забезпечення, що спрямована на поліпшення процесу. Популярність ретроспективи пов'язана з впровадженням гнучких методологій. Є різновидом мозкового штурму), щоби вирішити, що потрібно скоригувати.

Перерахуємо основні збори, у яких може взяти участь команда Scrum.

1) Організація беклогу. За цей захід, також відомий як ведення беклогу, несе відповідальність власник продукту. До його основних обов'язків входять приведення продукту у відповідність до його концепції та постійне відстеження настроїв на ринку та потреб клієнта. Для цього власник продукту і веде список, змінюючи в ньому пріоритети та підтримуючи його в актуальному вигляді на основі інформації від користувачів та команди розробників, щоб у будь-який час можна було розпочати роботу над внесеними до нього завданнями.
2) Планування спринту. На зборах команда розробників під керівництвом Scrum-майстра планує роботу (обсяг спринту), яку необхідно виконати протягом поточного спринту. На ньому вибирається

ціль спринту. Потім спринт додаються конкретні користувальницькі історії з бэклога продукту. Ці історії завжди співвідносяться з метою. При цьому команда Scrum узгоджує такі історії, які можна буде реалізувати практично під час спринту.

Наприкінці зборів із планування кожен член команди Scrum повинен чітко уявляти, які завдання можна виконати за спринт та як поставити інкремент.

3) Спринт. Спринт - це фактичний проміжок часу, протягом якого команда Scrum спільно працює над створенням готового інкременту. Як правило, спринт триває два тижні, хоча деяким командам простіше спланувати обсяг спринту на один тиждень або поставити інкремент, який має достатню цінність, за місяць. Дейв Вест із Scrum.org рекомендує планувати спринт тим коротшим, чим складніша робота і чим більше у ній невідомих. Але останнє слово завжди за командою. Протягом цього періоду власник продукту та команда розробників можуть переглянути обсяг спринту, якщо це необхідно. Це і є ключем до розуміння емпіричної суті Scrum.

Усі заходи від планування до ретроспективи проводяться протягом спринту. Після того, як тимчасовий проміжок для спринту визначений, він повинен залишатися незмінним, доки ведеться розробка. Так команда отримуватиме цінні уроки з минулого досвіду та застосовуватиме висновки до майбутніх спринтів.

4) Щоденна Scrum-нарад. Це дуже короткі щоденні збори, які для зручності проводяться в один і той же час (звичайно вранці) і в тому самому місці. Багато команд намагаються вкластися в 15 хвилин, проте це лише рекомендація. Щоденна Scrum-нарада проводиться, щоб кожен учасник команди був у курсі того, що відбувається, не відхилявся від мети та отримував план роботи на найближчі 24 години.

Найчастіше в рамках Scrum-наради кожному учаснику команди пропонується відповісти на такі три питання, пов'язані з досягненням мети спринту:

• «Що мені вдалося зробити вчора?»
• «Що я планую зробити сьогодні?»
• «Чи може мені щось завадити?»

Втім, такі збори можуть перетворитися на зачитування людьми записів із щоденника. Scrum-нарад потрібна, щоб усі обговорення залишалися в рамках щоденних зборів, а решта часу команда могла зосередитися на роботі.

5) Огляд результатів спринту. Наприкінці спринту команда збирається перегляду демонстрації інкременту (чи його вивчення) у неформальній обстановці. Команда розробників представляє зацікавленим сторонам та колегам завершені робочі завдання з беклогу, щоб зібрати відгуки. Власник продукту вирішує, чи варто випускати інкремент.

На зборах з огляду підсумків власник продукту також змінює беклог продукту виходячи з результатів останнього завершеного спринту. Цей процес може перейти у планування наступного спринту.

6) Ретроспектива спринту. Ретроспектива проводиться, щоб команда зафіксувала та обговорила всі успіхи та невдачі спринту, проекту, учасників та їхніх взаємин, інструментів чи навіть певних зборів. Мета ретроспективи — створити умови, щоб команда могла приділити увагу всьому, що вдалося і що потрібно покращити наступного разу, і не зациклювалася на невдачах.

Три найважливіші ролі, від яких залежить успіх застосування Scrum.

Склад scrum-команди передбачає три окремі ролі: власник продукту, scrum-майстер та команда розробників. Оскільки scrum-команди поєднують безліч функцій, в команду розробників також входять тестувальники, дизайнери, UX-фахівці та інженери з операцій.

1) Власник продукту Scrum

Власники борються за свій продукт. Їхнє завдання — розуміти вимоги бізнесу, клієнта та ринку. За підсумками цього розуміння вони розставляють пріоритети між робочими завданнями, які технічна команда виконуватиме у порядку. Про ефективні власники продукту можна сказати таке:

- вони становлять беклог продукту та керують ним;
- вони тісно співпрацюють з керівництвом компанії та командою, повідомляючи кожному учаснику значення робочих завдань у белог продукту;
- вони дають команді зрозумілі вказівки, щоб її учасники знали, які можливості надати такими;
- вони вирішують, коли поставити продукт, прагнучи робити це якнайчастіше.

Роль власника товару не завжди поєднана з роллю менеджера товару. Власники прагнуть створити всі умови, аби команда розробників створювала максимальну цінність для бізнесу. Важливо, щоб власником продукту була одна людина. Навряд команда розробників захоче отримувати різні вказівки від різних власників одночасно.

2) Scrum-майстер

Scrum-майстри стежать за застосуванням принципів Scrum у своїх командах. Вони навчають команди, власників товарів та інших співробітників компанії тонкощам Scrum-процесу і намагаються оптимізувати застосування цієї практики.

Успіх Scrum-майстра залежить від того, наскільки добре він розуміється на роботі, яку виконує команда, і може допомогти команді підвищити прозорість роботи та оптимізувати процес постачання продукту. Це головний координатор, який складає перелік необхідних ресурсів (кадрових та матеріально-технічних) для зборів із планування спринту та огляду його підсумків та ретроспективи спринту.

3) Команда розробників Scrum

На команди Scrum лягає вся основна робота. Вони спеціалісти за принципами збалансованої розробки. Найуспішніші команди згуртовані, знаходяться в одному місці і складаються з 5–7 учасників. Щоб визначити розмір команди, можна звернутися до відомого правила двох піц, яке сформулював глава Amazon Джефф Безос.

Кожен учасник команди має свої компетенції. Учасники навчають один одного виконанню різних завдань, щоб жоден із них не став перешкодою на шляху до мети. Успішні Scrum-команди здатні до самоорганізації, і їхній підхід до проектів пронизаний командним духом. Всі учасники команди допомагають один одному успішно завершити спринт.

Scrum-команди становлять план на кожен спринт. Вони прогнозують обсяг роботи, який здатні виконати за ітерацію, використовуючи як орієнтир показники своєї швидкості в минулих спринтах. Завдяки фіксованій тривалості ітерації команда розробників може проаналізувати правильність оцінки складності та процесу постачання продукту, що значно підвищує точність її прогнозів з часом.

Scrum, Kanban та Agile.

Scrum — настільки популярна agile-методика, що слова Scrum та Agile часто помилково використовують як синоніми. Але є й інші популярні методики, наприклад, Kanban. Деякі компанії навіть віддають перевагу гібридній моделі, що поєднує в собі елементи Scrum і Kanban. Її називають Scrumban або Kanplan — по суті, Kanban із беклогом.

І в Scrum, і Kanban використовуються візуальні засоби, такі як дошка Scrum або Kanban, для відстеження ходу виконання роботи. В основі обох методик лежить ефективність та розподіл складних завдань на дрібні робочі завдання, що піддаються виконанню, проте їх підходи до досягнення мети різняться.

Scrum будується навколо невеликих ітерацій із фіксованою тривалістю. Спочатку потрібно визначити тривалість спринту, після чого вибрати власні історії або елементи бэклога продукту, які можна реалізувати протягом цього циклу спринту. У Kanban кількість завдань, які потрібно виконати в поточному циклі, задається спочатку. Потім ведеться зворотний відлік часу, який піде на реалізацію цих можливостей.

У методики Kanban немає структури, властивої Scrum. У Scrum присутні кілька суворо певних понять: огляд підсумків спринту, ретроспектива, щоденна Scrum-нарада і т. д. Scrum також вимагає формування багатофункціональної команди, щоб можливість досягнення мети не залежала від сторонніх учасників. Зібрати міжпосадову команду – завдання не з легких. У цьому плані Kanban простіше адаптувати, а застосування Scrum тягне за собою докорінну зміну способу мислення та особливостей діяльності команди розробників.

Чому варто обрати Scrum?

Сама собою методика Scrum проста. Зрозуміти правила, артефакти, заходи та ролі нескладно. Вона задає структуру, але в ній є свобода вибору, яка виключає білі плями в процесі розробки та дозволяє належним чином врахувати специфіку різних компаній. Складні завдання можна впорядковувати в легко здійсненні історії користувача, а значить, Scrum ідеально підійде для складних проектів. Завдяки тому, що ролі та планові заходи чітко розмежовані, протягом усього циклу розробки зберігається прозорість та колективна відповідальність. Частий випуск продуктів мотивує команду та гарантує задоволення користувачів, адже вони бачать, як продукт розвивається протягом короткого відрізка часу.

І все-таки, щоб освоїти Scrum, може знадобитися якийсь час, особливо якщо команда розробників звикла до стандартної каскадної моделі. Новій команді належить вибрати scrum-майстра, освоїтися у світі коротких ітерацій, щоденних scrum-зборів та оглядів підсумків спринту. Це може стати справжнім струсом основ. Але переваги у довгостроковій перспективі переважають усі складнощі, пов'язані з освоєнням нових принципів. Scrum успішно застосовується в розробці складного апаратного та програмного забезпечення в різних. Це хороший аргумент на користь застосування методики в рамках організації.

1.2.2. Kanban методологія

Використання Kanban у розробці програмного забезпечення щороку набирає популярності. Гнучкість методу дозволяє пристосовуватися до постійних змін у процесі роботи та швидко переключатися між різними етапами розробки. У роботі проведено аналіз різновидів Kanban-дошок, які використовуються у розробці програмного забезпечення, та наведено процес організації роботи команди за основними принципами Kanban-методу. Метою роботи є проведення досліджень і реалізація методу роботи команди з розробки програмного забезпечення за принципами Kanban-методу. Об’єктом дослідження є процес організації роботи команди у розробці програмного забезпечення. Предметом є моделі, методи, алгоритми та інструменти реалізації Kanban-методу. Авторами розглянуто реалізацію загальних Kanban-підходів agile-методології та lean-мислення, без жорстких правил, але із принципами, на які можна посилатися. Для Kanban існують тільки спеціальні методи управління, але вони не є стандартизованими. Отже, удосконалення процесу lean-розробки є достатньо актуальним за його використання для реалізації програмного забезпечення. Kanban передбачає послідовні дії для удосконалення процесу організації розробки й отримання гарного результату. Система Kanban передбачає максимальне скорочення Lead Time, а саме часу роботи над завданням на всіх етапах. Тому підхід до виконання задач реалізується з орієнтацією на дошку, потік завдань і виявлення проблемних місць. Основним завданням є вдосконалення і реалізація процесу роботи таким чином, щоб команда була готова до того, що вимоги до системи та пріоритети задач можуть змінюватися декілька разів на

день. Процес підтримки комунікацій між різними командами також потребує чіткої схеми функціонування для швидкого обміну інформацією про виконану роботу. У роботі реалізована дошка візуалізації завдань з оптимальним набором статусів для ефективної роботи.

У кожного програмного забезпечення є життєвий цикл – етапи, через які воно проходить із початку створення до кінця розробки та впровадження. Методологія розробки включає набір методів з управління розробкою: правила, техніки та принципи, які роблять її більш ефективною. Останнім часом набирають популярності так звані гнучкі моделі та методи розробки, що передбачають розбиття проекту на невеликі робочі частини, які мають назву історії [1]. Одним із найпопулярніших прикладів гнучких підходів є Kanban-метод. Однак для ефективної роботи з використанням цієї системи потрібно реалізувати чітку схему циклів розробки програмного забезпечення. Аналіз останніх досліджень і публікацій. Сьогодні існує велика кількість методів і способів розробки [2;3]. Одними з найпопулярніших є Scrum і Waterfall. Scrum передбачає розробку через невеликі ітерації (два-чотири тижні), за які команда виконує чітко визначений об’єм задач [2]. Scrum забезпечує високий рівень взаємодії між членами команди та мотивування до самоорганізованої плідної роботи. В основі методу Waterfall лежить послідовний перехід від одного етапу на другий без пропусків і можливості повернутися назад. Це забезпечує зрозумілу і просту структуру процесу розробки та стабільність завдань, що виконуються [4]. Система Kanban передбачає використання дошки: фізичної чи електронної. Дошка є обов’язковим елементом для гнучкої методології. Кожен член команди отримує до неї доступ в будь-який час і бачить, на якому етапі перебуває те чи інше завдання. Наявні приклади дошок передбачають розподіл на основні категорії: планування, розробку, тестування та реліз. Такий підхід дозволяє чітко розуміти, на якому етапі знаходиться команда, але у процесі розробки дуже часто виникає необхідність повернутися назад на будь-який етап [5].

У роботі представлено цикл розробки програмного забезпечення, що передбачає вільний перехід між основними етапами розробки, а також описано процедури, які потрібно виконувати команді для ефективної роботи за цим методом. Також авторами визначені основні ролі спеціалістів та задачі, які повинен виконувати кожен із них [6]. Задачі постійно передаються від однієї команди розробників до іншої. Крім цього, Kanban реалізує ідею потоку як виробничого процесу, де немає простою незавершених завдань [7]. Тобто над однією задачею можуть працювати послідовно декілька спеціалістів. У цьому разі попередні помилки очевидні та виявляються миттєво. Це дозволяє уникнути зайвих витрат, підвищує якість розробки та скорочує терміни її виконання [8].

Kanban-метод передбачає обговорення продуктивності в режимі реального часу і повну прозорість робочих процесів. Етапи роботи візуально представлені на Kanban-дошці, що дозволяє членам команди бачити стан кожного завдання у реальному часі. Зазвичай команда складається з таких

спеціалістів: власника продукту (Product Owner), програмістів і головного розробника (Developer Lead), спеціалістів із тестування (QA) і головного QA (QA Lead), скрам-майстра (Scrum Master), бізнес-аналітика (Business Analyst) і продукт-архітектора (Product Architect).

Після затвердження проектних рішень, прийнятих на етапі проектування, команда починає розробку. Весь функціонал розподіляться між програмістами, котрі реалізують алгоритми, пишуть вихідний код, виконують компіляцію і налагодження коду. У проекті є план процесу робіт. Спочатку він аналізується і поділяє дошку на об’єкти, які відображають етапи [9]. Виділимо статуси для завдань: To do, In Progress, Code Review, Ready QA Testing, QA Under Test, Ready UAT Testing, UAT Under Test, UAT Done, Expected та Done (рис. 1.1).

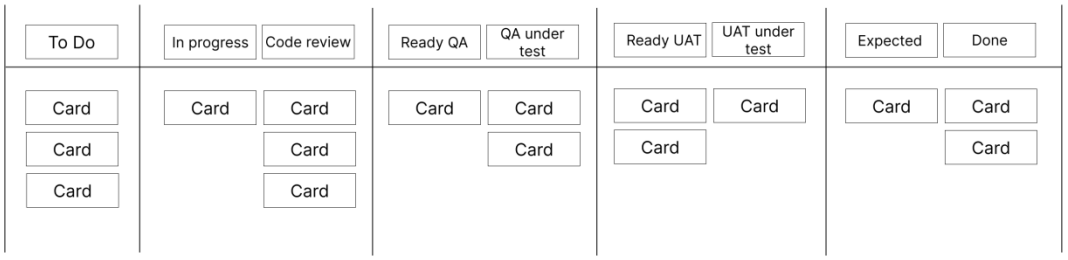

Рисунок 1.1 – Схема дошки за методом Kanban

Система Kanban передбачає максимальне скорочення Lead Time, тобто часу роботи над завданням на всіх етапах. У зв’язку з цим підхід до виконання задач реалізується з орієнтацією на дошку, фокусуванням на потоці завдань і виявленням проблемних місць. Тобто команда пересувається по об’єктах із завданнями справа наліво й обговорює, яким чином можна швидше перевести задачу на наступний етап. Kanban-дошка з візуалізацією ситуації на проекті дозволяє обговорювати не конкретні дії членів команди, а й поточний стан завдань і їх статус у потоці виконання.

Виділимо п’ять основних типів завдань, які використовуються під час роботи над проектом: завдання (task), історію (story), епік (epic), поліпшення (improvement) і баг (bug). На початку циклу розробки створюється проектна документація із вимогами до системи. Вона повинна чітко та детально відображати весь функціонал системи та після погодження бути оформлена бізнес-аналітиками у вигляді задач. Спочатку створюється епік – велике завдання, яке повністю описує окремий новий функціонал. Кожен епік складається з історій (описів вимог). Кожна історія реалізує окремий функціонал і може не входити до складу епіків, а бути окремим самостійним об’єктом, що описує новий функціонал або зміни поточного.

Крім того, створюються покращення (improvements) – вимога до оновлення або невеликої зміни частини функціоналу. Вони можуть додаватися на етапі розробки або тестування. Тестувальники вносять свої пропозиції, які можуть спростити роботу із системою та зробити її більш

комфортною для кінцевого користувача. Також часто баги, що знаходяться на етапі тестування, перекваліфіковують у поліпшення. Баг є помилкою в системі, через яку програма видає неправильну поведінку і, як наслідок, результат. Більшість багів виникають через помилки, допущені розробниками у вихідному коді або при реалізації дизайна. Також деякі баги виникають через некоректну роботу компілятора, що виробляє некоректний код.

Баги розрізняють за ступенем критичності та пріоритетом. За ступенем критичності виділяють: блокуючі (blocker), які роблять неможливою подальшу роботу з додатком; важливі (major), через які система не функціонує належним чином; нормальні (normal), що призводять до некоректної роботи окремих компонентів системи; малозначущі (minor) або невеликі баги. За пріоритетністю виділяють баги: fix in release – виправити в новій версії продукту; must fix – терміново виправити (блокуючі, які усувають до виходу нової версії); fix if time – виправити, якщо дозволяє час; never fix – ніколи не виправляти (є неактуальними або прийнятні для системи). Оскільки Kanban не передбачає жорстких, обмежених певних часом етапів, під час яких потрібно реалізувати ту чи іншу задачу, рішення про початок роботи із завданням приймається на командних мітингах (основа зворотного зв’язку). Регулярність задає ритм, за яким проходить потік роботи. Виділимо основні типи мітингів, такі як: − щоденна зустріч, де обговорюється статус поточних завдань; − зустріч по плану робіт (раз на два тижні); − зустріч по покращенню сервісу (раз на два тижні) – обговорюється якість системи та її покращення; − зустріч із організації процесу роботи (раз на два тижні) – обговорюються проблеми, що виникли під час організації процесу роботи (займається Agile-мастер, котрий координує роботу команди); − зустріч по обзору стратегії (раз на місяць, квартал) – обговорюються зміни у стратегії. Взаємодія команди відбувається за схемою, зображеною на рис. 1.2.

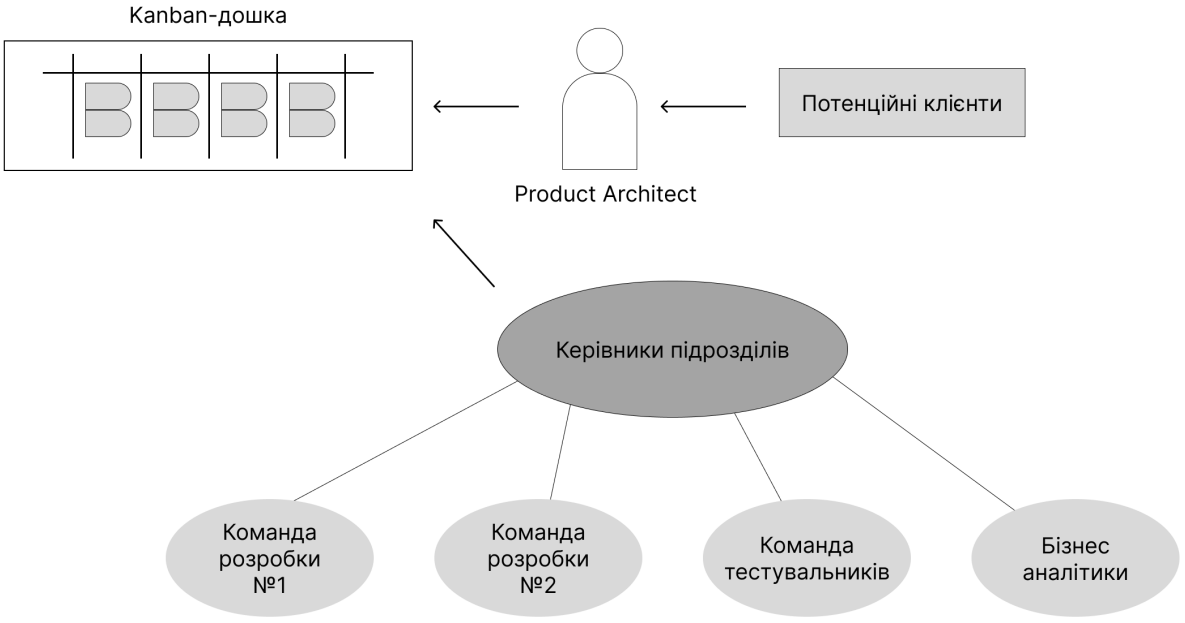

Рисунок 1.2 – Взаємодія Kanban-команди

Така модель забезпечує особисте спілкування членів команд кожного напряму з керівником підрозділу. Члени команди самостійно організовують свою роботу, використовуючи Kanban-дошку. Керівник займається пріоритезацією завдань і вирішенням виникаючих проблем.

Визначимо процес розробки на прикладі невеликої частини додатку, що виконує просте математичне ділення двох чисел із плаваючою крапкою.

```
int main()
{ float x = 0;
float y = 0;
std::scanf(“%f %f”, &x, &y);
float z = -x / y;
std::printf(“Result: %f \n”, z); }
```

Product Architect аналізує побажання потенційних клієнтів щодо функціоналу та інтерфейсу додатку. На основі отриманих даних створюється проектна документація з описом вимог. Далі оформлюється Kanban-дошка, створюються задачі та додається команда.

Для нових створених задач автоматично призначається статус To do. Цей статус означає, що задача створена, але поки не розпочата робота над нею. Цикл змін статусів відображено на рис. 3.

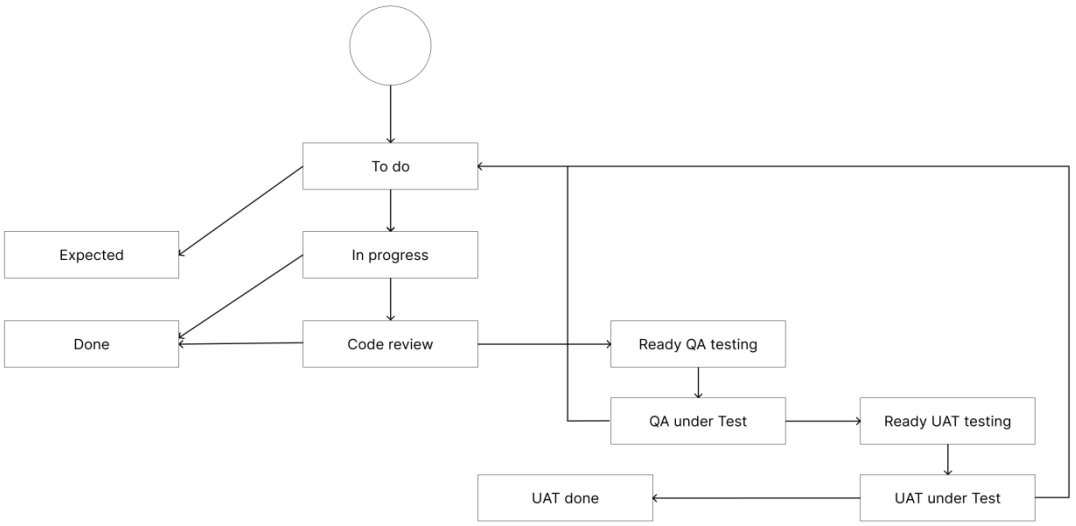

Рисунок 1.3 – Цикл змін статусів для задач

Спеціаліст, на якого призначена задача, змінює статус на In Progress, коли починає виконання. На етапі розробки програміст, який розробляє або виправляє функціонал, описаний у завданні, вносить зміни на dev-оточенні та залишає свої коментарі про результати виконання.

Розробники розроблюють окремо невеликі частини функціоналу та проводять юніт-тестування, тобто переконуються, що окремо взята частина додатку працює. Після цього статус завдання потрібно змінити на Code Review [9]. Після встановлення цього статусу на задачу автоматично призначається головний розробник, який перевіряє розроблений код. Якщо

текст коду задовольняє вимоги, зміни додаються до тестового оточення. Задачі призначається статус Ready QA Testing.

Розробник може призначити задачу на тестувальника, котрий її створював або вже працював із нею [10]. Також задачу можна залишити непризначеною (Unassigned), і вільний тестувальник зможе почати роботу над нею. Коли починається етап тестування, QA-спеціаліст змінює статус на QA Under Test. Головна задача тестувальника – перевірити, що нова частина додатку працює та коректно взаємодіє з усією системою. Якщо під час перевірки виникають питання або тестувальник знаходить баги, вони оформлюються в окремих задачах зі статусом To do.

Далі продукт-архітектори та розробники вивчають проблему та приймають рішення про доцільність її вирішення. Коли тестування задачі закінчено, проблеми вирішені та функціонал відповідає поставленим до нього вимогам, тестувальник переміщає задачу у статус Ready UAT Testing.

Коли починається цикл UAT тестування, задачі переводяться спеціалістами у статус UAT Under Test, і призначаються відповідні тестувальники. UAT тестування – це приймальне, призначене для користувача тестування. Будь-яка розробка або доопрацювання програмного забезпечення проходить завершальну стадію UAT-тестування. Тестування проводиться бізнес-користувачами прийнятої системи, тобто потенційними клієнтами, але на практиці часто його проводять QA-спеціалісти. Перевірки проводять на спеціальному оточенні, яке містить у собі всі зміни та доповнення, що відбувалися на етапах розробки та QA-циклу. Якщо знаходяться проблеми на етапі UAT-тестування, приймається рішення про їх виправлення у поточному релізі, перенесення на наступний або визнання проблеми особливою можливістю системи та внесення відповідних правки в релізну документацію. Після успішного завершення перевірки задачі переводяться до UAT Done.

Також передбачені статуси Expected і Done. Expected призначаються створеному багу, якщо він визнається не багом, а особливою можливістю (feature). Тобто мається на увазі поведінка системи, що явно не відображена у вимогах, але є правильною з погляду бізнес-логіки.

Статус Done використовується для багів, які вирішено не виправляти. Окрім цього, цей статус виставляється для задач типу Task, коли вони виконані тестувальником або розробником. Основний результат ефективності методу можна відстежити на діаграмі, наведеній на рис. 1.4.

На діаграмі відображена динаміка задач залежно від часу та кількості задач. Зі збільшенням часу та кількості задач відповідно змінюється кількість задач у процесі та закритих задач.

На ці показники насамперед впливають такі фактори: час на формування вимог, розробку, тестування та виправлення багів, зміна пріоритетів по задачам, зміна вимог до функціоналу. Саме від цих факторів залежить час на виконання задачі та перебування задачі в черзі. Завдяки орієнтованості методу на зменшення кількості одночасно виконуваних

завдань час на закриття задачі відповідно зменшується. Дуже часто виникають зависання задач. Виділимо основні причини:

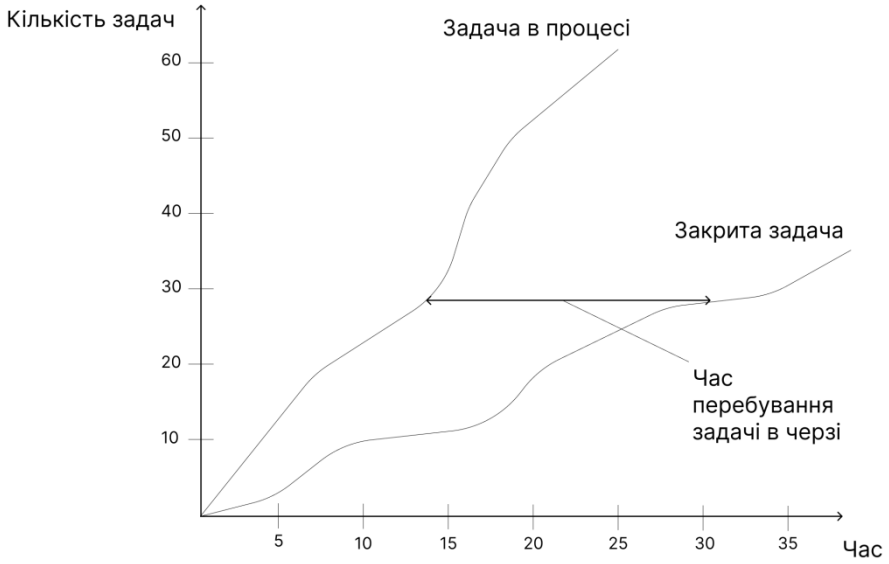

Рисунок 1.4 – Діаграма залежності.

− велику кількість заблокованих завдань. Така ситуація може виникати через баги, які блокують функціонал, затримки на етапі підтвердження від продукт-архітектор, затримки у процесі виправлення багів;

− постійну зміну пріоритетів задач. Завдання доходять до останніх стадій, раптом стають не дуже важливими, команда перемикається на нові завдання. Виходить, що змінилися пріоритети, і робота не завершена;

− додавання нових поліпшень і змін у вимогах до системи. Kanban передбачає можливість постійних змін вимог до системи, тому часто виникають ситуації, коли розроблений функціонал доповнюють або навіть змінюють.

Висновки: підбиваючи підсумки, хочу відзначити, що Scrum - гнучка методика розробки, а Kanban - ще більше. Усьому свій час та місце. Якщо це розробка нового продукту, то на старті розробки і до релізу краще використовувати Scrum, тому що він робить розробку контрольованішою за термінами. Також у Scrum багато комунікацій у команді: робітники обговорюють весь беклог спринту перед стартом, ставлять питання авторам завдання (UX, продакт-менеджерам, бізнес-аналітикам), оцінюють завдання спільно за допомогою Planning poker. Scrum допомагає детально занурити команду у суть продукту.

2. ЗОВНІШНЄ І ВНУТРІШНЄ ПРОЄКТУВАННЯ

2.1. Зовнішнє проектування

На цьому етапі висвітлимо основні стадії розробки веб-додатку.

2.1.1. Вхідні дані.

Вхідними даними для веб-додатку “TodoDashboard” є:

- назва задачі;
- опис задачі;
- час на виконання;
- виконавець;
- статус задачі;

2.1.2.Вихідні дані

Вихідними даними для веб-додатку “TodoDashboard” є:

- список задач у вигляді карток, поділених на типи;
- кругова діаграма аналізу виконання задач.

2.1.3.Функціональне призначення

Програмний продукт використовує браузер для доступу до інтернету та синхронізації задач на всіх пристроях користувачів.

Операції існує 5 видів:

- «Add ToDo»;
- «Delete ToDo»;
- «Edit ToDo»;
- «Change ToDo status.
- «Add Performer;
- «Delete Performer;
- «Edit Performer;

2.1.4.Експлуатаційне призначення

За допомогою програмного продукту виконується спрощення процесу планування та комунікації між членами команди та дає змогу більш детально відстежувати робочі процеси та їх виконання. Це несе за собою чималу вигоду для бізнесу, оскільки зменшується час на розробку програмного продукту.

2.1.5.Функціональні вимоги.

Програма представляє собою дошку, поділену на наступні сегменти: «Задачі до реалізації», «У роботі», «На тестуванні», «Виконано».

У секції «Задачі до реалізації» зберігаються усі задачі, які команда має реалізувати у поточному спринті, для кожної задачі вже обрані розробник та час на виконання.

В секції «У роботі» зберігаються задачі, які почали реалізовуватися.

У секції «На тестуванні» відображаються задачі, які завершені в розробці і наразі прядені тестувальникам для перевірки. Якщо задача не пройшла етап тестування, то повертається до секції «У роботі».

У секції «Виконано» відображаються виконані задачі, які пройшли тестування.

Опис картки задачі.

Кожна картка створюється за допомогою конструктора, у якому заповнюються поля «Найменування задачі», «Короткий опис», «Статус» (секції на дошці), «Виконавець», «Час на реалізацію». Кожну створену картку можна редагувати та змінювати її статус та виконавця.

Виконавець задачі обирається зі списку робітників, які зберігаються у базі даних. Для кожного співробітника є поля з основною інформацією: «Ім’я Прізвище По-батькові», «Дата народження», «Посада».

2.1.6.Специфікація функціональних вимог.

Специфікація функціональних вимог представлена у вигляді діаграми прецедентів (рис. 2.1).

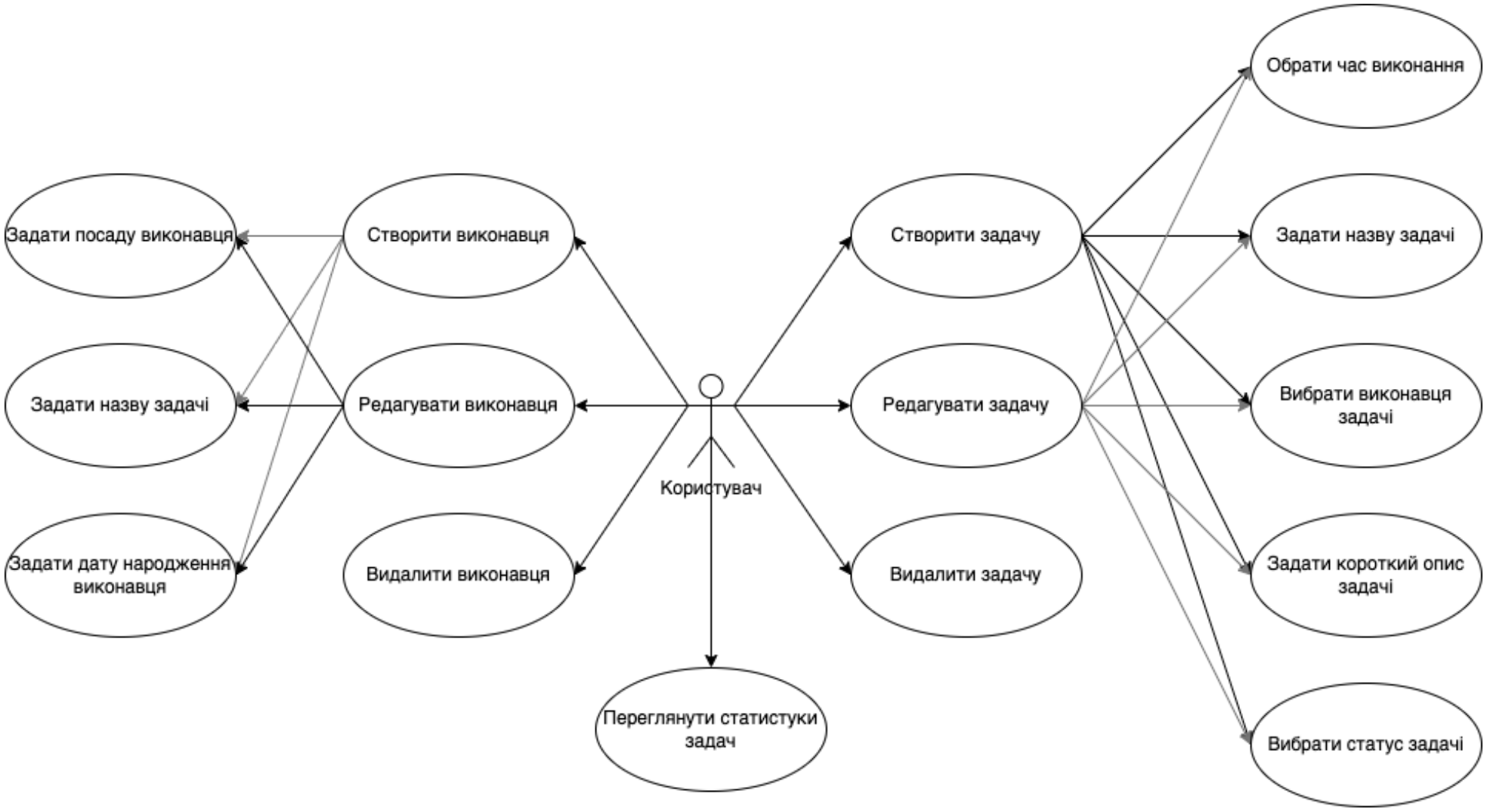

Рисунок 2.1 – Діаграма прецедентів.

2.2. Внутрішнє проектування

2.2.1. Проєктування архітектури системи

2.2.1.1. Моделювання словника системи

Ідентифіковані сутності: дошка, картка задачі, список співробітників, робітник, графік.

Ідентифіковані обов’язки:

Дошка – відображення чотирьох секцій з картками задач, а саме: «Задачі до реалізації», «У роботі», «На тестуванні», «Виконано».

Картка задачі – зберігання та відображення даних задачі, а саме: «Найменування задачі», «Короткий опис», «Статус» (секції на дошці), «Виконавець», «Час на реалізацію».

Список співробітників – зберігання та відображення даних кожного співробітника компанії у вигляді списку, а саме: «Ім’я Прізвище По-батькові», «Дата народження», «Посада».

Співробітник – введення та редагування даних співробітника.

Графік – будування та відображення графіку аналітики задач та їх успішності.

Атрибути та операції, необхідні для виконання обов'язків кожної сутності-класу наведено у табл. 2.1.

Таблиця 2.1 – Атрибути та операції, необхідні для виконання обов'язків кожної сутності-класу

Сутність	Атрибути	Методи
дошка	секції картки	відобразити дані
картка задачі	назва опис статус виконавець час	створити видалити редагувати
список співробітників	масив елементи масиву	відобразити дані
співробітник	ім’я дата народження посада	створити видалити редагувати
графік	картки	відобразити дані

2.2.1.2. Моделювання примітивних типів, простих залежностей, наслідування та структурних зв’язків.

Визначення примітивних типів виконаємо на етапі побудови діаграми компонентів. Результат моделювання різних видів зв’язків подано у табл. 2.

Таблиця 2.2 – Моделювання залежностей

Компонент, який зв'язується	Клас, з яким зв’язуються	Тип зв’язку
App	AppRouter	Асоціація
AppRouter	AdminPanel	Асоціація
	Vacancies	Залежність
	Team	Залежність
	Projects	Залежність
AdminPanel	AdminVacancyCard	Залежність
	AdminProjectCard	Залежність
	AdminTeamCard	Залежність

Відношення між компонентами (функціями), представлені у вигляді діаграми компонентів (рис.2.2) для React та у (рис.2.3) для Vue.

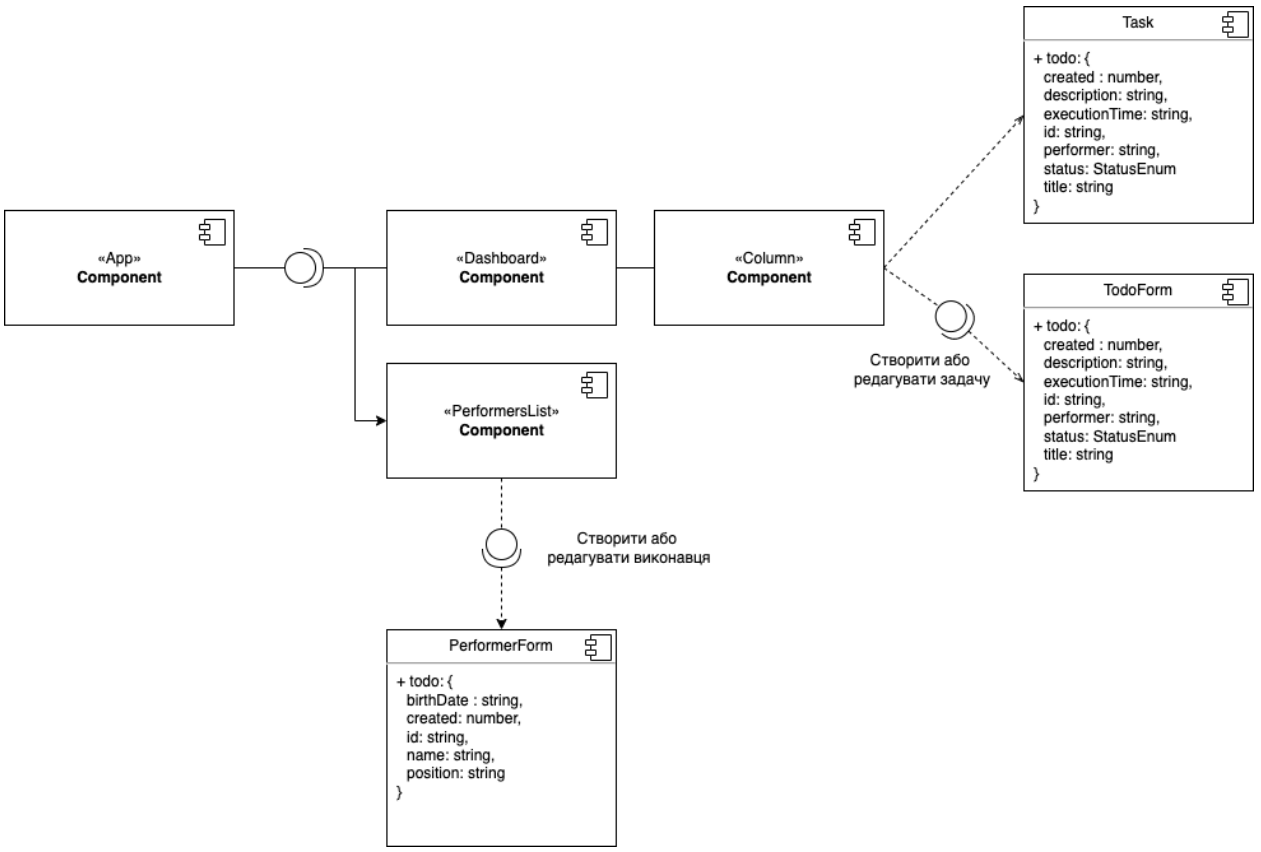

Рисунок 2.2 – Діаграма компонетнів для веб-додатку “DashBoard” для React.js

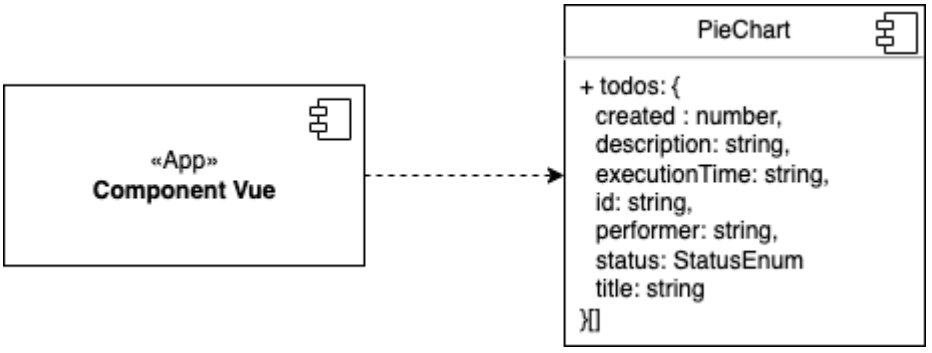

Рисунок 2.3 – Діаграма компонетнів для веб-додатку “DashBoard” для Vue.js.

2.2.2. Проєктування інтерфейсу користувача

Інтерфейс користувача буде відображатися у формі, поділеній на секції з картками. Основні кольори інтерфейсу нейтральні – білий, чорній та сірий. Картки матимуть кольорові позначки, залежну від того, у якому стані вони знаходяться:

«Задачі до реалізації» – червоний,

«У роботі» – жовтий,

«На тестуванні» – синій,

«Виконано» – зелений.

Розміри карток – 00x00 pt.

Розмір тексту на сторінці – 16 pt.

Розміри кнопок – 00x00 pt.

Сценарій діалогу для системи «ToDo Dashboard» наведено на діаграмі станів (рис.2.4).

2.2.3. Проєктування бази даних

У даному проєкті використовується платформа Firebase для доступу до бази даних.

Firebase — це платформа розробки мобільних додатків Google, яка допомагає створювати, покращувати та розвивати вашу програму.

Firebase — це набір інструментів для «створення, покращення та розвитку вашої програми», і інструменти, які він надає, охоплюють більшу частину сервісів, які розробники зазвичай повинні створювати самі, але насправді не хочуть створювати, тому що вони краще зосередитись на самому додаток. Це включає такі речі, як аналітика, автентифікація, бази даних, конфігурація, сховище файлів, push-повідомлення, і цей список можна продовжити. Сервіси розміщуються у хмарі та масштабуються практично без зусиль з боку розробника. Продукти мають серверні компоненти, які повністю обслуговуються і керуються Google. Клієнтські SDK, що надаються Firebase,

взаємодіють з цими серверними службами безпосередньо без необхідності встановлювати проміжне ПЗ між вашим додатком і службою. Отже, якщо ви використовуєте один із варіантів бази даних Firebase, ви зазвичай пишете код для запиту бази даних у своєму клієнтському додатку.

У Firebase було створено Realtime Database, яка працює через WebSocket. Створено сутності Todos, Performers (рис. 2.5).

Рисунок 2.4 – Діаграма станів

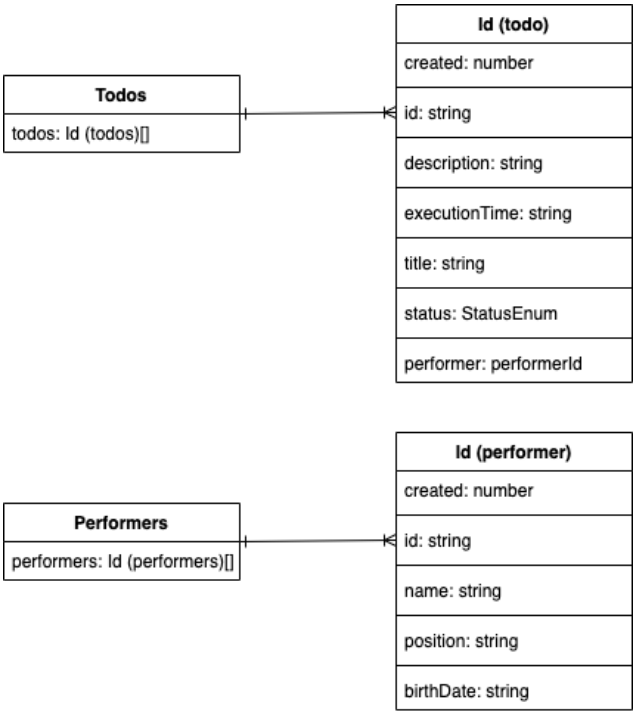

Рисунок 2.5 – Діаграма структури бази даних

Висновки: на етапі проєктування веб-додатку був розроблений інтерфейс користувача та спроєктовані основні алгоритми та база даних для сайту. На основі цих розробок був написаний код програми, який надалі буде проходити тестування та налагодження.

3. ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

3.1. Тестування методом «білої скриньки»

Метод “handleSubmit (saveTodo)”.
Метод створює або змінює вибрану задачу.

```
const handleSubmit = async (values, helpers) => {
   if (!executionTime) {
     notification.error({ message: 'Треба вибрати час виконання!' });
     return;
   }

   if (isEditMode) {
     try {
       await updateTodo({ ...values, executionTime });
       notification.success({ message: 'Задача успішно змінена' });
       handleCloseModal(helpers);
     } catch (error) {
       notification.error({
         message: 'Не вдалося змінити задачу, спробуйте пізніше',
         description: error.message,
       });
     }
   } else {
     try {
       await addTodo({ ...values, executionTime })
       notification.success({ message: 'Задача успішно додана' });
       handleCloseModal(helpers);
     } catch (error) {
       notification.error({
         message: 'Не вдалося додати задачу, спробуйте пізніше',
         description: error.message,
       });
     }
   }
 }
```

Спочатку користувачу потрібно натиснути на кнопку “Створити задачу” або редагувати вибрану задачу. Заповнити форму валідними даними, вибрати виконавця, вибрати статус, дату на реалізацію, заповнити найменування та опис задачі та натиснути на кнопку “ОК”. Якщо збереження елемента у базу даних пройшло успішно, буде виведене повідомлення "Задача успішно додана" або “Задача успішно змінена” у випадку помилки, користувач

отримає повідомлення “Не вдалося додати задачу, спробуйте пізніше” або “Не вдалося змінити задачу, спробуйте пізніше”.

Вхідні дані:
Змінна title типу string;
Змінна description типу string;
Змінна status типу string приймає значення:
'backlog' або 'in progress' або 'testing' або 'done'.
Змінна performer типу string приймає значення: performer_id.
Змінна executionTime типу string приймає значення: дата.

Вихідні дані:
Повідомлення про успішне змінення або створення задачі "Задача успішно додана" або “Задача успішно змінена”, або повідомлення про помилку “Не вдалося додати задачу, спробуйте пізніше” або “Не вдалося змінити задачу, спробуйте пізніше”.

Тестування метода “handleSubmit (saveTodo)”
Приклад на введення валідних даних крім title (рис. 3.1).
Вхідні дані:
title = пусте значення;
description = “test description”;
status = 'backlog';
performer = ”1653213057266”;
executionTime = “21.07.2022”;
Вихідні дані: повідомлення про невалідний title, “Обов'язково”;

Приклад на введення валідних даних крім description (рис. 3.2).
Вхідні дані:
title = “test title”;
description = пусте значення;
status = 'backlog';
performer = ”1653213057266”;
executionTime = “21.07.2022”;
Вихідні дані: повідомлення про невалідний description, “Обов'язково”;

Приклад на введення валідних даних крім performer (рис. 3.3).
Вхідні дані:
title = “test title”;
description = “test description”;
status = 'backlog';
performer = пусте значення;
executionTime = “21.07.2022”;
Вихідні дані: повідомлення про невалідний performer, “Обов'язково”;

Приклад на введення валідних даних крім executionTime (рис. 3.4).
Вхідні дані:
title = “test title”;
description = “test description”;
status = 'backlog';
performer = ”1653213057266”;
executionTime = пусте значення;
Вихідні дані: повідомлення про невалідний executionTime, “Треба вибрати час виконання!”;

Приклад на введення усіх валідних даних (рис. 3.5).
Вхідні дані:
title = “test title”;
description = “test description”;
status = 'backlog';
performer = ”1653213057266”;
executionTime = “21.07.2022”;
Вихідні дані: повідомлення про успішне створення або редагування задачі, "Задача успішно додана" або “Задача успішно змінена”;
Покриття умов для метода handleSubmit (saveTodo) наведена у табл. 3.1.

Таблиця 3.1 – Покриття умов (метода handleSubmit (saveTodo)).

	isValidTitle		isValidDescription		isValidPerformer		isValidExecutionTime		isValidForm	
	+	-	+	-	+	-	+	-	+	-
1		*	*		*		*			*
2	*			*	*		*			*
3	*		*			*	*			*
4	*		*		*			*		*
5	*		*		*		*		*	

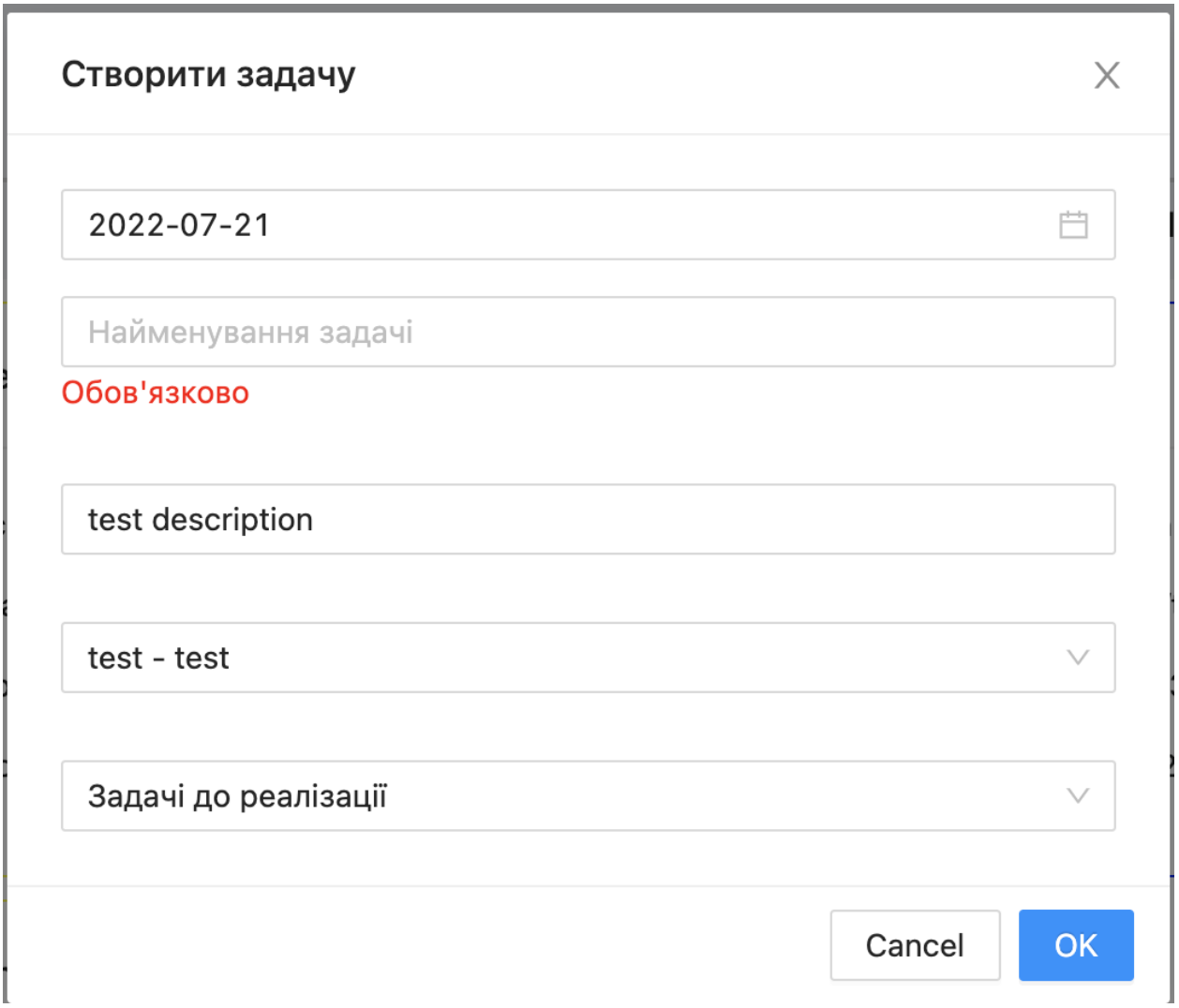

Рисунок 3.1 – Тестування методу “handleSubmit (saveTodo)”.

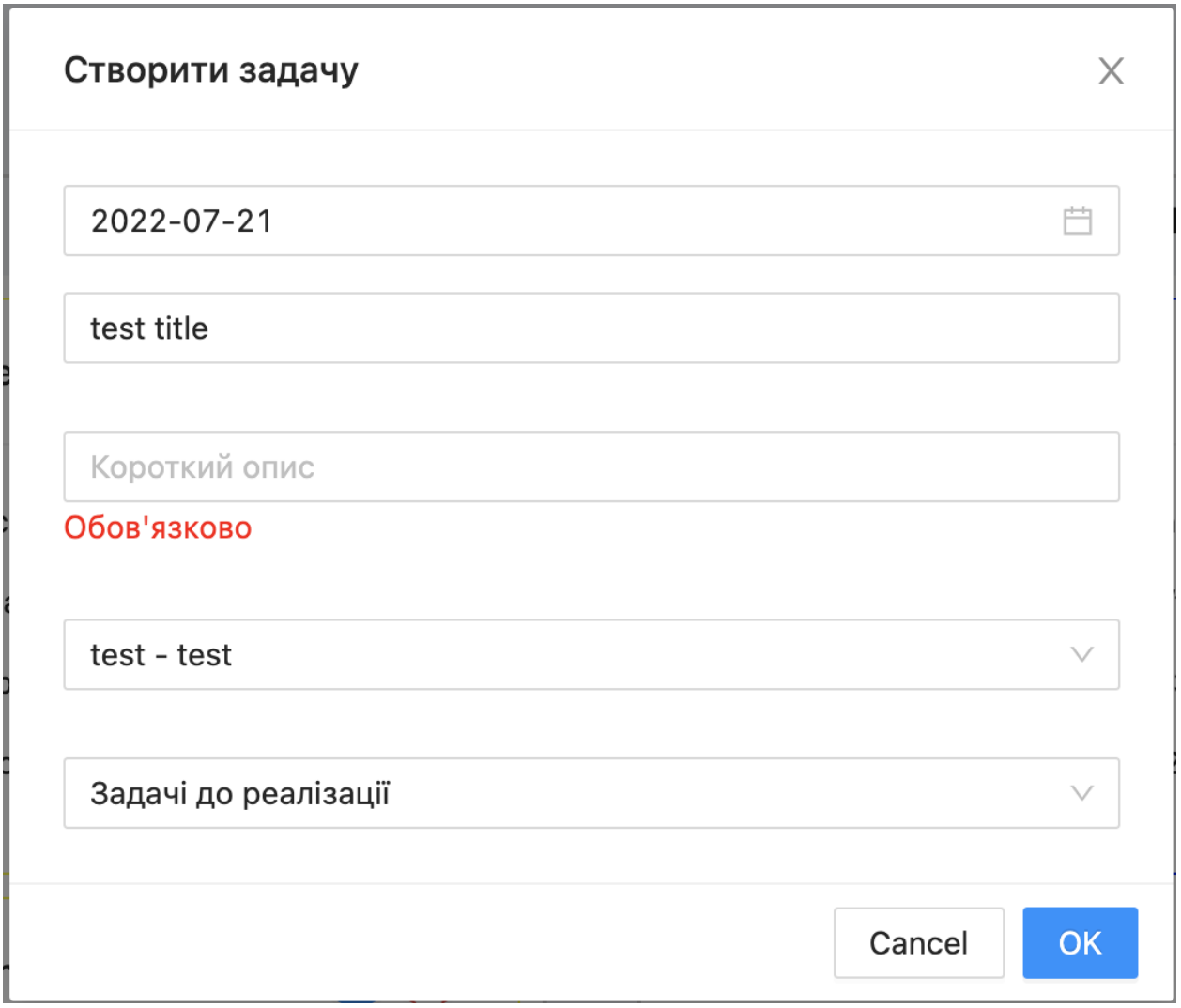

Рисунок 3.2 – Тестування методу “handleSubmit (saveTodo)”.

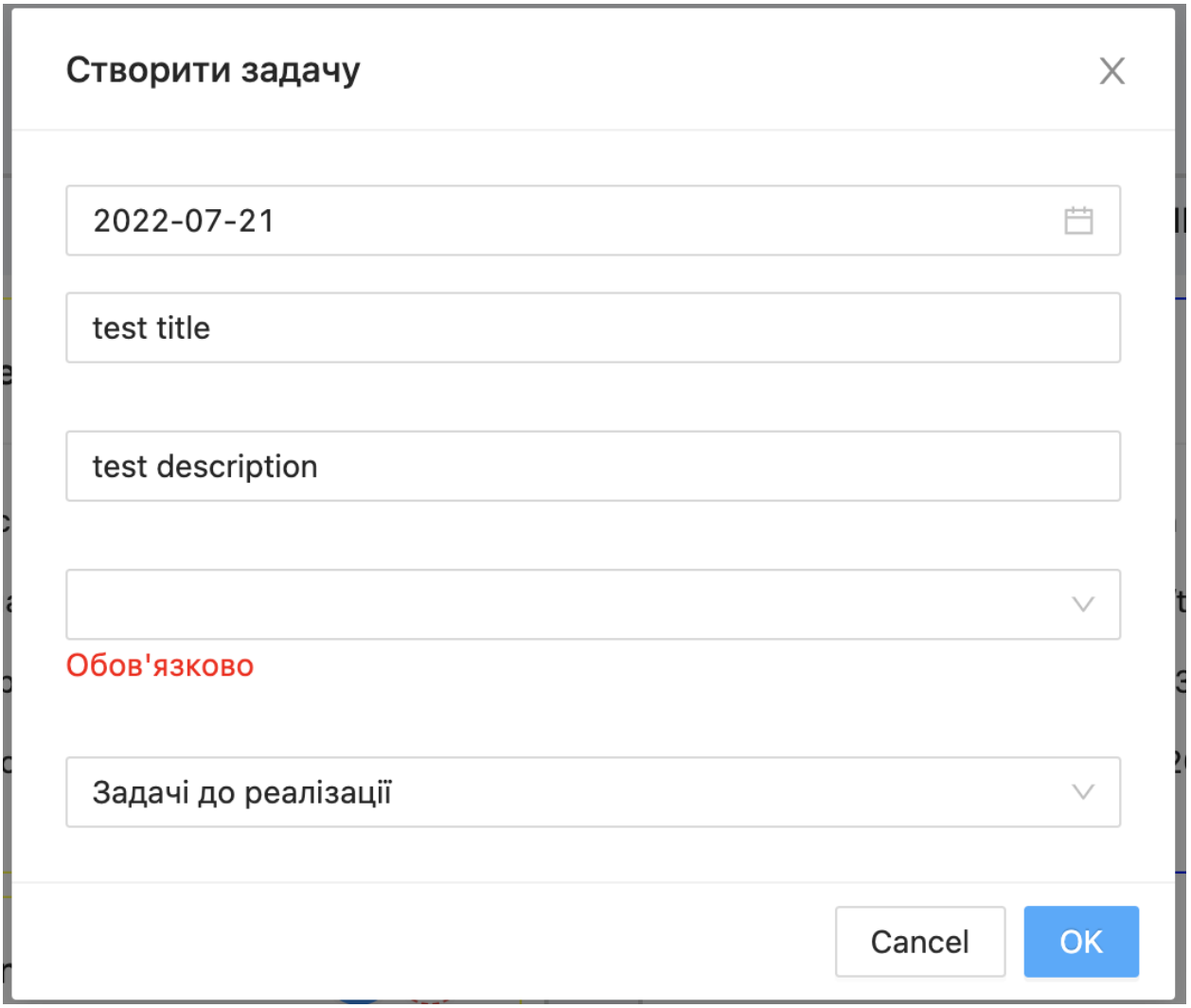

Рисунок 3.3 – Тестування методу “handleSubmit (saveTodo)”.

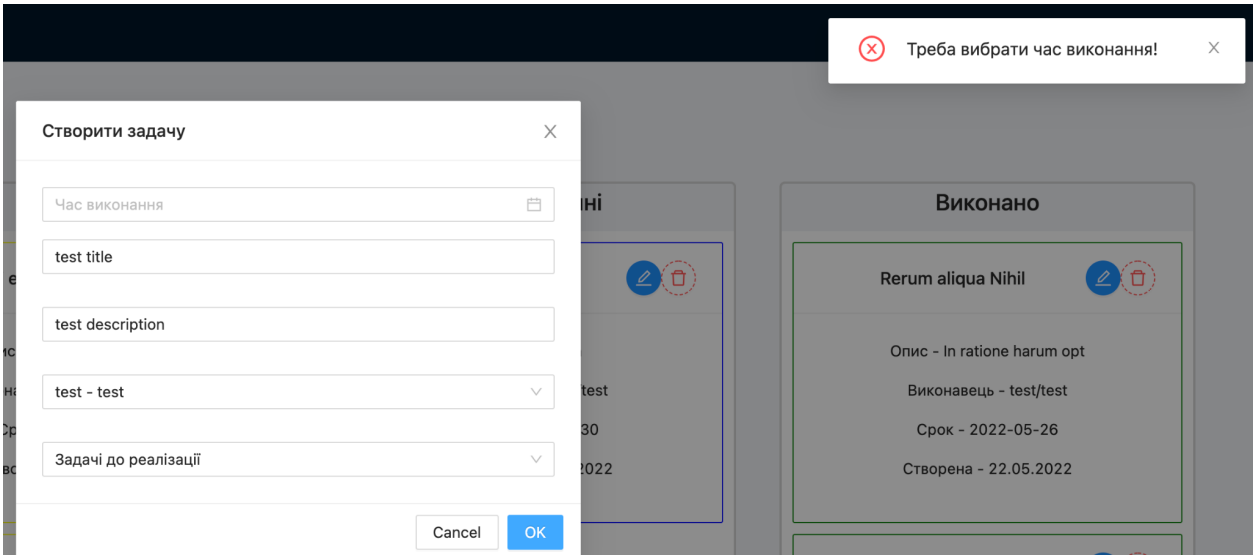

Рисунок 3.4 – Тестування методу “handleSubmit (saveTodo)”.

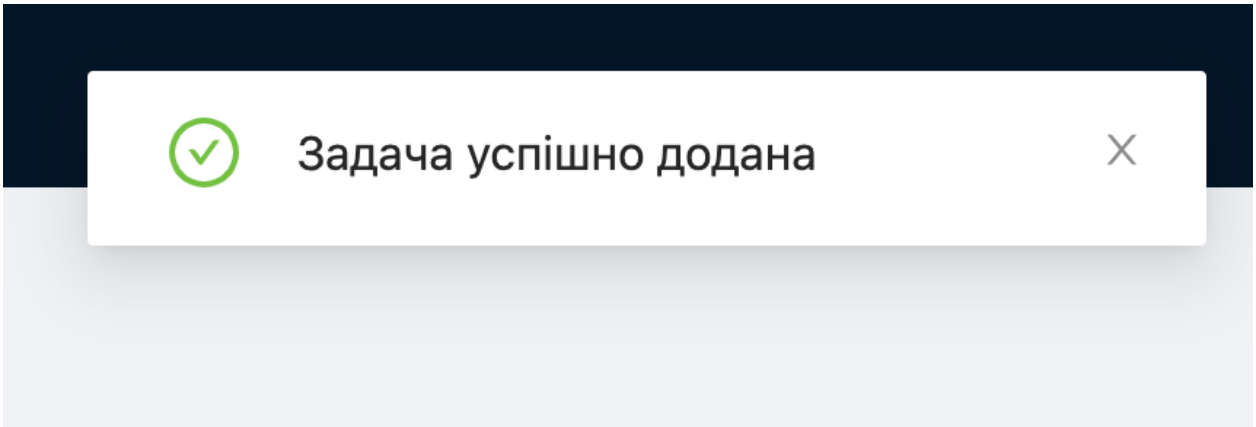

Рисунок 3.5 – Тестування методу “handleSubmit (saveTodo)”.

Метод “handleSubmit (savePerformer)”.
Метод створює або змінює вибрану задачу.

```
const handleSubmit = async (values, helpers) => {
   if (isEditMode) {
     try {
       await updatePerformer(values);
       notification.success({ message: 'Виконавець успішно змінений' });
       handleCloseModal(helpers);
     } catch (error) {
       notification.error({
         message: 'Не вдалося змінити виконавця, спробуйте пізніше',
         description: error.message,
       });
     }
   } else {
     try {
       await addPerformer(values)
       notification.success({ message: 'Виконавець успішно доданий' });
       handleCloseModal(helpers);
     } catch (error) {
       notification.error({
         message: 'Не вдалося добавити виконавця, спробуйте пізніше',
         description: error.message,
       });
     }
   }
 }
```

Спочатку користувачу потрібно натиснути на кнопку “Створити виконавця” або редагувати вибраного виконавця. Заповнити форму валідними даними, ввести ПІБ, ввести посаду, вибрати дату народження та натиснути на кнопку “ОК”. Якщо збереження елемента у базу даних пройшло успішно, буде виведене повідомлення "Виконавець успішно доданий" або “Виконавець успішно змінений” у випадку помилки, користувач отримає повідомлення “Не вдалося змінити виконавця, спробуйте пізніше” або “Не вдалося добавити виконавця, спробуйте пізніше”.

Вхідні дані:
Змінна name типу string;
Змінна position типу string;
Змінна birthDate типу string приймає значення: дата.

Вихідні дані:

Повідомлення про успішне змінення або створення виконавця "Виконавець успішно доданий" або “Виконавець успішно змінений”, або повідомлення про помилку “Не вдалося змінити виконавця, спробуйте пізніше” або “Не вдалося добавити виконавця, спробуйте пізніше”.

Тестування метода “handleSubmit (savePerformer)”
Приклад на введення валідних даних крім name (рис. 3.6).
Вхідні дані:
name = пусте значення;
position = “test position”;
birthDate = “21.07.2001”;
Вихідні дані: повідомлення про невалідний name, “Обов'язково”;

Приклад на введення валідних даних крім position (рис. 3.7).
Вхідні дані:
name = “test name”;
position = пусте значення;
birthDate = “21.07.2001”;
Вихідні дані: повідомлення про невалідний position, “Обов'язково”;

Приклад на введення валідних даних крім birthDate (рис. 3.8).
Вхідні дані:
name = “test name”;
position = “test position”;
birthDate = пусте значення;
Вихідні дані: повідомлення про невалідний birthDate, “Обов'язково”;

Приклад на введення усіх валідних даних (рис. 3.9).
Вхідні дані:

name = “test name”;
position = “test position”;
birthDate = “21.07.2001”;
Вихідні дані: повідомлення про успішне створення або редагування виконавця, "Виконавець успішно доданий" або “Виконавець успішно змінений”;
Покриття умов для метода handleSubmit (savePerformer) наведена у табл. 3.2.

Таблиця 3.2 – Покриття умов (метода handleSubmit (savePerformer)).

	isValidName		isValidPosition		isValidBirthDate		isValidForm	
	+	-	+	-	+	-	+	-
1		*	*		*			*
2	*			*	*			*
3	*		*			*		*
4	*		*		*		*	

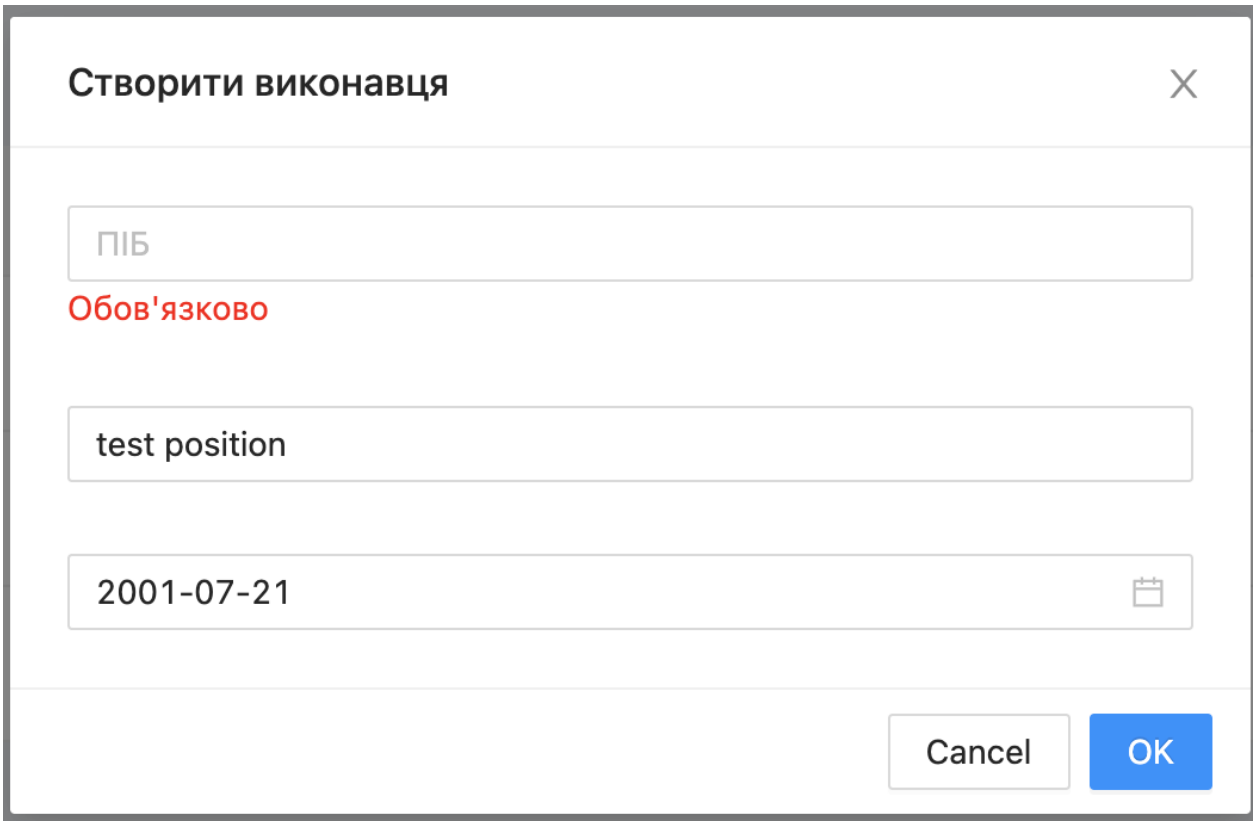

Рисунок 3.6 – Тестування методу “handleSubmit (savePerformer)”.

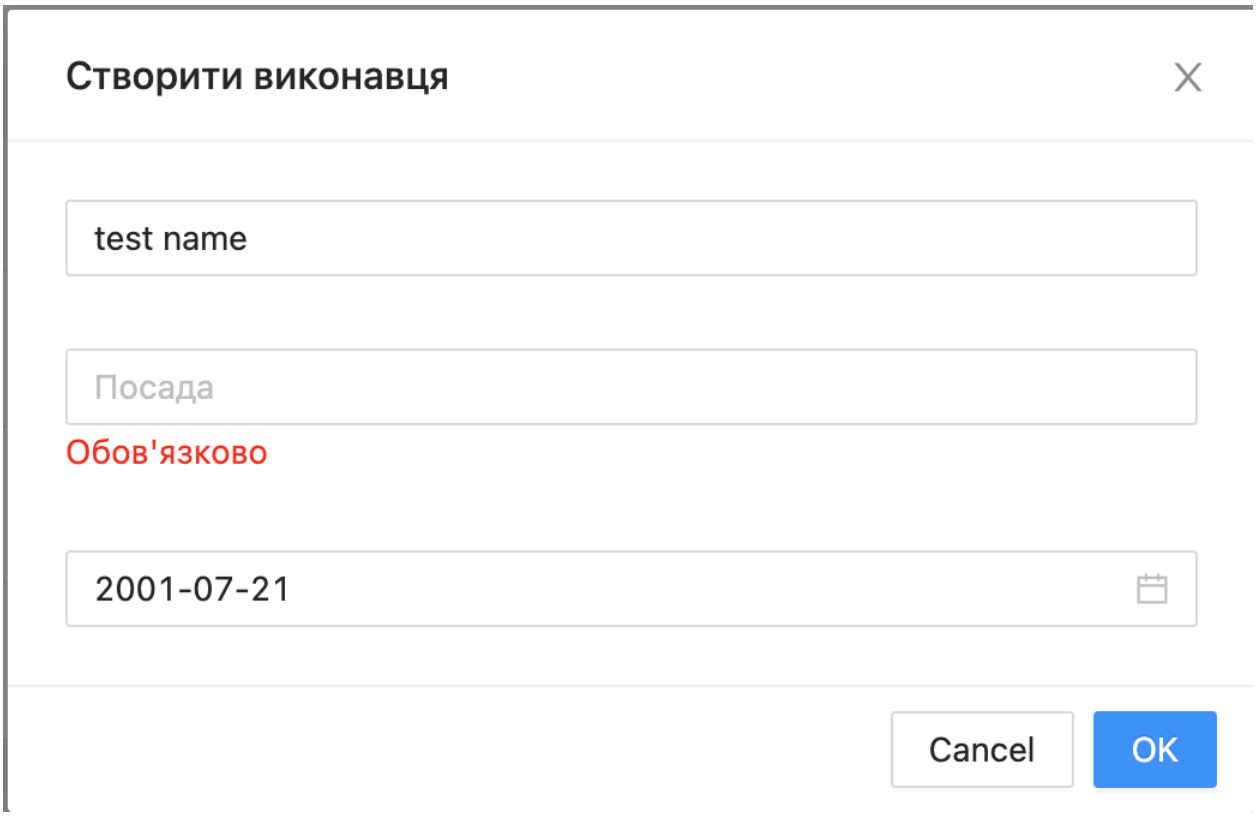

Рисунок 3.7 – Тестування методу “handleSubmit (savePerformer)”.

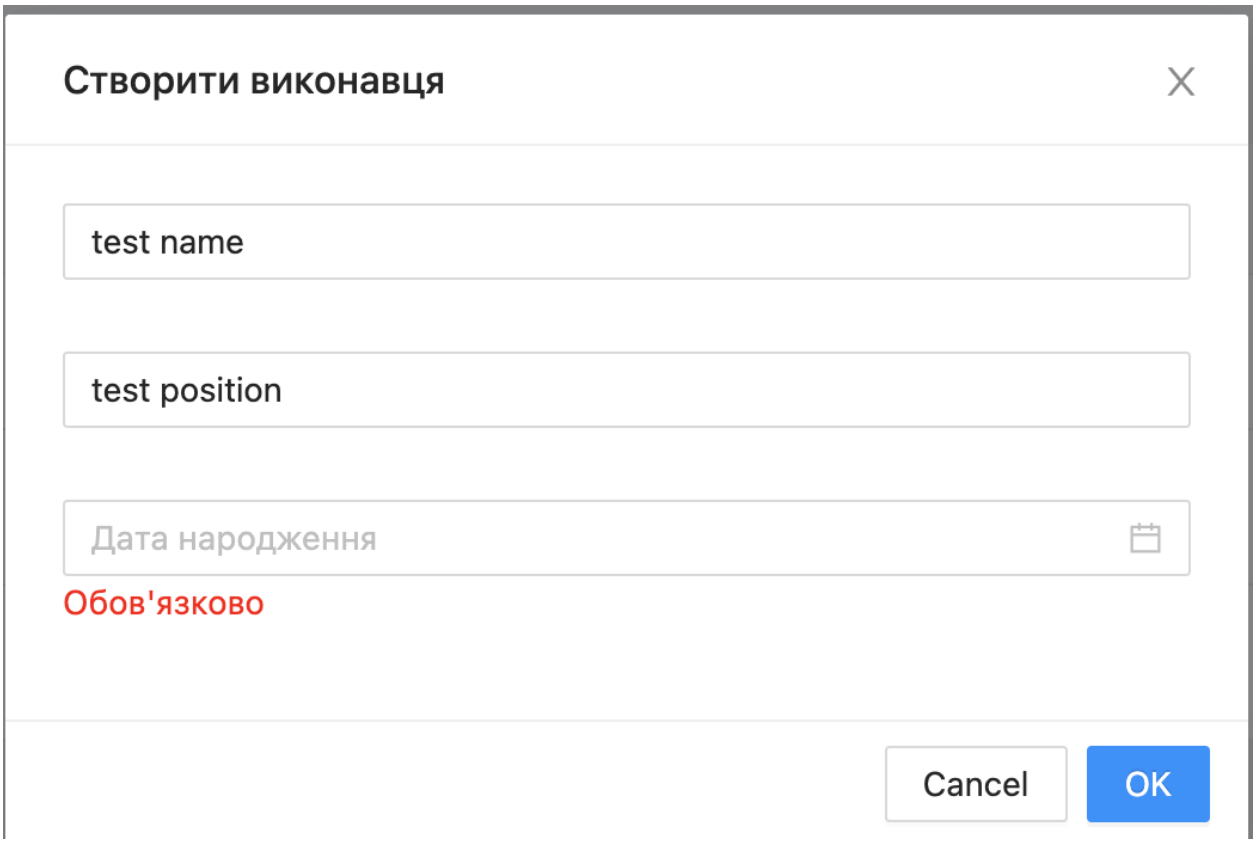

Рисунок 3.8 – Тестування методу “handleSubmit (savePerformer)”.

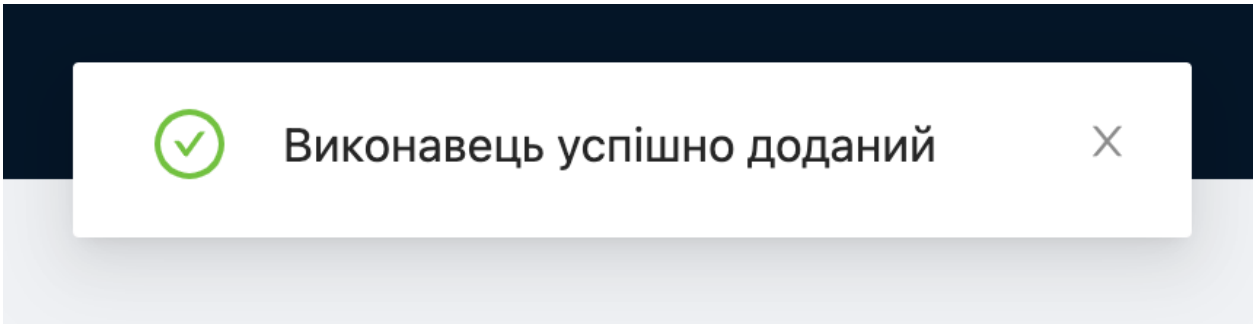

Рисунок 3.9 – Тестування методу “handleSubmit (savePerformer)”.

3.2. Налагодження програми

В ході тестування написаного веб-додатку була виявлена помилка із відображенням помилки валідації при створення виконавця у полі “Дата народження”. Помилка не відображалася, але валідація працювала коректно, про це свідчила вимкнена кнопка “OK”. Помилку було усунено (рис. 3.10), через людський фактор був пропущений обов’язковий атрибут для компоненту ErrorMessage.

Рисунок 3.10. Виправлена помилка.

Висновки: завдяки проведеному тестуванню вдалося перевірити валідацію полів при створенні та редагуванні задачі і виконавця та виправити знайдені помилки. Завдяки тестуванню валідація та відправка даних тепер працюють коректно.

ВИСНОВКИ

У роботі вдосконалено процес організації роботи команди з розробки програмного забезпечення за системою Kanban. Реалізована дошка для візуалізації завдань з оптимальним набором статусів для ефективної роботи, приведені принципи роботи з нею та цикл змін статусів для задач. Розглянуто на практичному прикладі процес розробки та взаємодії команди. Розробка за системою Kanban є однією з найпопулярніших як для невеликих, так і для малих проектів завдяки орієнтованості на задачі, а саме: на зменшення кількості одночасно виконуваних задач і зменшення часу виконання кожної задачі, гнучкість у прийняті рішень і частоту змін пріоритетів, тісну взаємодію між усіма членами команди. Наукова цінність наведеного методу полягає у підвищенні ефективності та покращенні роботи наявних методів організації розробки програмного забезпечення. У ході подальших досліджень планується вдосконалення розробленої моделі з метою підвищення ефективності системи.

Література

1. Corona E., Pani F.E. A Review of Lean-Kanban Approaches in the Software Development. 2013. URL: https:// pdfs.semanticscholar.org/0725/ 482b6ced393863440f7e063c268e3790d18c.pdf. (дата звернення: 07.11.2020)
2. Reddy A. The Scrumban [R] Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban. AddisonWesley Professional, 2015.
3. Стелман Е., Грін. Д. Осягаючі Agile: Цінності, принципи, методології. Москва : Манн, Іванов і Фербер, 2019. 448 с.
4. Киричек Г.Г., Киричек О.О. Модель оцінки плагіату програмного коду на основі системи контролю версій. Восточно-Европейский журнал передовых технологий. 2012. № 2 (2). С. 25–28.
5. Кон М. Scrum: гибкая разработка ПО. Москва : Вільямс, 2011. 576 с.
6. Kirichek, G., Tymoshenko, V., Rudkovskyi, O., Hrushko, S.: Decentralized System for Run Services. In: CEUR Workshop Proceedings 2353, 2019. P. 860–872.
7. Murino T., Naviglio G., Romano E. Optimal size of Kanban board in a single stage multi product system. 2010. URL: https://www.researchgate.net/ publication/234790654_Optimal_size_of_Kanban_board_in_a_ single_stage_multi_product_system (дата звернення: 07.11.2020)
8. Kirichek, G., Skrupsky, S., Tiahunova, M., Timenko, A.: Implementation of web system optimization method. In: CEUR Workshop Proceedings 2608, 2020. P. 199–210.
9. Ahmad, M.O., Markkula, J., Oivo, M. Kanban in software development: A systematic literature review. In Software Engineering and Advanced Applications (SEAA). 2013. URL: https://www.researchgate.net/publication/260739586_ Kanban_in_Software_Development_A_Systematic_Literature_Review (дата звернення: 07.11.2020)
10. Мартин Р., Ньюкирк Дж, Косс Р. Быстрая разработка программ. Принципы, примеры, практика. Москва : Вильямс, 2004. 752 с.

Додатки

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету науки і
технологій
Анатолій РАДКЕВИЧ
18.02.22

ВЕБ-ДОДАТОК "TODO DASHBOARD"

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01227-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
18.02.22

Керівник розробки
Вадим АНДРЮЩЕНКО
18.02.22

Виконавець
Анастасія Бібе
18.02.22

Нормоконтролер
Олена КУРОП'ЯТНИК
18.02.22

2022

ЗАТВЕРДЖЕНО
44165850.01227-01-ЛЗ

ВЕБ-ДОДАТОК “TODO DASHBOARD”

Технічне завдання

44165850.01227-01-ЛЗ

Листів 13

2022

МІСТ

ВСТУП

Веб-додаток «ToDo Dashboard», що розробляється, призначений для використання з браузера. За допомогою цього додатку будуть створюватися задачі (ToDo) для учасників команди. Додаток дозволяють слідкувати за часом виконання задач та статистикою їх успішності.

Додаток буде заповнений прикладами, які поділені на операції:

- «Add ToDo»;
- «Delete ToDo»;
- «Edit ToDo»;
- «Save ToDo»;
- «Change ToDo order».

Приклади можуть бути виконані в режимі реального часу

Користувач, використовуючи цей продукт, може вносити свої задачі, редагувати, видаляти, змінювати та змінювати їх статус, також даний додаток зберігає усі нотатки на сервері і може отримати доступ до них з будь-якого пристрою на котрому є браузер та доступ до інтернету.

Програмний продукт є безкоштовним та легким у використанні, що робить його більш цікавим на фоні інших аналогів.

Програмний продукт призначений для людей віком від 18 до 65 років.

1 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 08.12.21 №77ст ректора Українського державного університету науки і технологій “Про призначення наукових керівників та затвердження тем бакалаврських робіт” за спеціальністю 121 “Інженерія програмного забезпечення» факультету “Комп’ютерних технологій і систем” по кафедрі “Комп’ютерні інформаційні технології”.

Тема дипломної роботи: “Розробка веб-додатків з використанням JavaScript фреймворків Vue і React””.

Керівник: доцент Андрющенко В. О.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення – програмний продукт використовує браузер для доступу до інтернету та синхронізації задач на всіх пристроях користувачів.

Операції існує 5 видів:

- «Add ToDo»;
- «Delete ToDo»;
- «Edit ToDo»;
- «Save ToDo»;
- «Change ToDo order».

Експлуатаційне призначення – за допомогою програмного продукту виконується спрощення процесу планування та комунікації між членами команди та дає змогу більш детально відстежувати робочі процеси та їх виконання.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

4.1 Вимоги до функціональних характеристик

Програма повинна забезпечити можливість керування задачами, а саме, створення задач, отримання текстової інформації та статусу задачі.

До неї належать такі основні сутності: задача;

В програмі необхідно реалізувати формування списку задач

Вхідні дані:

- назва задачі;
- опис задачі;
- час на виконання;
- виконавець;

Дані вводяться з клавіатури.

Вихідні дані: список задач;

Самі задачі зберігаються у базі даних Firebase.

Вихідні дані виводяться у вигляді інтерфейсу.

4.2 Вимоги до надійності

Вимоги до надійності наступні:

- для полів ведення необхідно забезпечити контроль ведених даних;
- при оновленні даних забезпечити показ відповідних повідомлень про стан роботи програми та результати виконання операцій;
- наявність архівної копії тексту програми на зовнішньому носії;
- наявність резервної копії бази даних на зовнішньому носії (сервері де розміщена база даних).

4.3 Умови експлуатації

Програмний продукт повинен використовуватись у приміщеннях, які відповідають умовам роботи ЕОМ, а саме мають такі кліматичні, санітарні та гігієнічні умови, які відповідають ДНАОП 0.00-1.31-99 (див. табл. 4.1).

Таблиця 4.1 − Кліматичні умови

Пора року	Категорія робіт згідно з ГОСТ 12.1-005-88	Температура повітря, град.С оптимальна	Відносна вологість повітря, % оптимальна	Швидкість руху повітря, м/с оптимальна
Холодна	легка-1 а	22 - 24	40 - 60	0,1
	легка-1 б	21 - 23	40 - 60	0,1
Тепла	легка-1 а	23 - 25	40 - 60	0,1
	легка-1 б	22 - 24	40 - 60	0,2

Працювати з програмою може людина, що має навички роботи з персональним комп’ютером та ознайомлена з керівництвом користувача.

4.4 Вимоги до складу та параметрів технічних засобів

Розроблюваний програмний продукт повинен використовуватись на ЕОМ, що мають:

- 4 ГБ оперативної пам’яті;
- 69 ГБ простору на жорсткому диску;
- клавіатуру;
- мишу;
- процесор 32- або 64-разрядный процесор з тактовою частотою 1 ГГц або вище с набором інструкцій SSE2;
- монітор;
- USB-порт;
- доступ до мережі інтернет;

4.5 Вимоги до інформаційної та програмної сумісності

Програмний продукт розробляється для всіх видів операційних систем.

Обов’язковим до встановлення є бібліотека “.Net framework 4.5” або вище. Середовище розробки: “Opera”, “Google Chrome”, “Firefox”, “Safari”.

4.6 Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних). На

упаковці повинно бути вказана назва продукту, номер версії (якщо вона змінювалась), мінімальні системні вимоги. На зворотній стороні упаковки вказується розробник та його юридична адреса.

4.7 Вимоги до транспортування та зберігання

Транспортування повинне забезпечувати збереження програмного продукту, його цілісність і запобігання несанкціонованого доступу до нього. Програмний виріб міститься на хмарному носії, переданий на флешці.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

- специфікація;
- текст програми;
- опис програми.

Вся документація до програмного додатку повинна задовольняти вимоги до програмної документації.

5 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Основні стадії та етапи розробки наведені у таблиці 5.1.

Таблиця 5.1 – Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	31.01.22 - 18.02.22
Робочий проект	Програмування та відладка програми.	19.02.22 - 01.05.22
	Тестування програми	02.05.22 - 24.05.22
	Розробка, узгодження і затвердження програмної документації.	25.05.22 - 12.06.22

7 ПОРЯДОК І КОНТРОЛЬ ПРИЙМАННЯ

Контроль за виконанням роботи здійснює керівник розробки доц. Андрющенко В. О.

Прийом здійснюється уповноваженою комісією.

6 БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.:

Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. − 38

Додаток Б

Текст програми

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './assets/index.css';
import { store } from './store';
import { Provider } from 'react-redux';
import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
 <Provider store={store}>
  <BrowserRouter>
   <App />
  </BrowserRouter>
 </Provider>,
 document.getElementById('root')
);
```

styles/index.css

```
@import 'antd/dist/antd.css';
```

App.js

```
import { Layout, Menu } from 'antd';
import styles from './assets/styles.module.css';

// Components
import Dashboard from './components/Dashboard';
import PerformersList from './components/PerformersList';
import { Route, Link, Navigate, Routes } from
'react-router-dom';

function App() {
 return (
  <div className="App">
   <Layout>
    <Layout.Header>
     <Menu mode="horizontal" theme="dark">
      <Menu.Item key="dashboard"><Link
to="dashboard">Дашборд задач</Link></Menu.Item>
      <Menu.Item key="performers"><Link
to="performers">Список робітників</Link></Menu.Item>
      <Menu.Item key="static"><a
href="https://dashboard-vue-five.vercel.app/">Статистика</
a></Menu.Item>
     </Menu>
    </Layout.Header>
    <Layout.Content className={styles.content}>
     <Routes>
      <Route index path="dashboard" element={<Dashboard
/>} />
      <Route path="performers" element={<PerformersList />}
/>
      <Route path="*" element={<Navigate to="dashboard"
replace />} />
     </Routes>
    </Layout.Content>
   </Layout>
  </div>
 );
}

export default App;
```

assets/styles.module.css

```
.content {
  padding: 50px;
  text-align: center;
  width: 100%;
  margin: 0 auto;
}

.todo {
  width: 100%;
}
```

```
.form {
  width: 60%;
  margin: 0 auto 50px;
}

.todo__text {
  text-align: left;
  overflow: hidden;
}

.todo__completed .todo__text {
  text-decoration: line-through;
}

.todo__completed {
  box-shadow: 0px 0px 8px 2px rgba(38, 255, 0, 0.2);
}

.todo__over {
  box-shadow: 0px 0px 8px 2px rgba(153, 24, 24, 0.2);
}

.dashboard {
  display: flex;
}

.error {
  color: red;
}

.task {
  margin-bottom: 10px;
  border-width: 3px;
}

.task__list {
  flex-grow: 1;
  min-height: 100px;
  padding: 10px;
  list-style: none;
  transition: background-color 0.3s ease;
}

.column {
  flex: 1;
  display: flex;
  flex-direction: column;
  margin: 20px;
  border: 2px solid lightgray;
  border-radius: 5px;
}
```

components/Dashboard

```
import React, { useState, useEffect } from "react";
import { DragDropContext } from "react-beautiful-dnd";
import Column from "./Column";
import styles from '../assets/styles.module.css';
import { useListVals } from 'react-firebase-hooks/database';
import { getTodosRef, changeTodoStatus, getPerformersRef }
from "../api"
import TodoForm from "./TodoForm"
import { Button } from "antd";

export const columns = {
 'backlog': {
  id: 'backlog',
  title: 'Задачі до реалізації',
  todos: [],
 },
 'in progress': {
  id: 'in progress',
  title: 'У роботі',
  todos: [],
 },
 'testing': {
  id: 'testing',
  title: 'На тестуванні',
  todos: [],
 },
 'done': {
  id: 'done',
```

```
  title: 'Виконано',
  todos: [],
 },
};

const Dashboard = () => {
 const [todos, todosLoading] = useListVals(getTodosRef());
 const [performers, performersLoading] =
useListVals(getPerformersRef());

 const [state, setState] = useState({ ...columns });
 const [isOpenModal, setIsOpenModal] = useState(false);

 useEffect(() => {
  if (!todosLoading && !performersLoading) {
   const result = todos.reduce(function (r, a) {
    const currentPerformer = performers.find(item => item.id
=== a.performer);
    r[a.status] = r[a.status] || [];
    r[a.status].push({
     ...a,
     performer: currentPerformer,
    });
    return r;
   }, Object.create(null));

   const newState = { ...state };
   Object.keys(newState).forEach(key => {
    newState[key].todos = result[key] || []
   })
   setState(newState);
  }
 }, [todos, performers])

 const handleCloseModal = () => {
  setIsOpenModal(false);
 }

 const onDragEnd = (result) => {
  const { source, destination, draggableId } = result;

  if (!destination) {
   return;
  }

  if (
   source.droppableId === destination.droppableId &&
   source.index === destination.index
  ) {
   return;
  }

  const startColumn = state[source.droppableId];
  const finishColumn = state[destination.droppableId];

  // Move in the same column
  if (startColumn === finishColumn) {
   const newTodos = [...startColumn.todos];
   newTodos.splice(source.index, 1);
   newTodos.splice(destination.index, 0, draggableId);

   const newColumn = {
    ...startColumn,
    todos: newTodos
   };

   const newState = {
    ...state,
    [newColumn.id]: newColumn,
   };

   setState(newState);
   return;
  }

  changeTodoStatus(draggableId, destination.droppableId)
};

if (todosLoading || performersLoading || !state) {
 return <p>Loading...</p>
}
```

```
 return (
  <>
   <Button
    type="primary"
    onClick={() => setIsOpenModal(true)}
    size="large"
   >
    Створити задачу
   </Button>
   <DragDropContext onDragEnd={onDragEnd}>
    <div className={styles.dashboard}>
     {Object.keys(state).map((columnId) => {
      const column = state[columnId];
      return <Column key={column.id} column={column}
todos={column.todos} />;
     })}
    </div>
   </DragDropContext>
   <TodoForm
    visible={isOpenModal}
    onCancel={handleCloseModal}
   />
  </>
 );
}

export default Dashboard;
```

components/TodoForm.js

```
import { useState, useRef, useEffect } from 'react';
import { Input, Space, DatePicker, Select, notification, Modal }
from 'antd';
import styles from '../assets/styles.module.css';
import { Formik, Form, Field, ErrorMessage } from 'formik';
import * as Yup from "yup";
import { addTodo, updateTodo, getPerformersRef } from
"../api";
import moment from "moment";
import { useListVals } from 'react-firebase-hooks/database';

const statusOptions = [
 {
  id: 'backlog',
  title: 'Задачі до реалізації',
 },
 {
  id: 'in progress',
  title: 'У роботі',
 },
 {
  id: 'testing',
  title: 'На тестуванні',
 },
 {
  id: 'done',
  title: 'Виконано',
 }
];

const emptyValues = {
 title: "",
 description: "",
 status: "backlog",
 performer: "",
}

const validationSchema = Yup.object({
 title: Yup.string().required("Обов'язково"),
 description: Yup.string().required("Обов'язково"),
 status: Yup.string().required("Обов'язково"),
 performer: Yup.string().required("Обов'язково"),
});

const CustomInput = ({ props, name }) => (
 <>
  <Input {...props} />
  <p className={styles.error}>
   <ErrorMessage name={name} />
  </p>
 </>
)
```

```
export default function TodoForm({
 onCancel,
 visible,
 isEditMode,
 todo
}) {
 const [performers, performersLoading] =
useListVals(getPerformersRef());
 const [executionTime, setExecutionTime] = useState(null);
 const formikRef = useRef();
 const [initialValues, setInitialValues] = useState(emptyValues);


 useEffect(() => {
  if (!isEditMode) return;
  setInitialValues({
   ...todo,
   performer: todo.performer.id,
  });
  const momentDate = moment(todo.executionTime,
'YYYY-MM-DD');
  setExecutionTime(momentDate);
 }, [visible])


 const handleCloseModal = (helpers) => {
  onCancel();
  setExecutionTime(null)
  helpers.resetForm();
 }


 const handleSubmit = async (values, helpers) => {
  if (!executionTime) {
   notification.error({ message: 'Треба вибрати час
виконання!' });
   return;
  }


  if (isEditMode) {
   try {
    await updateTodo({ ...values, executionTime });
    notification.success({ message: 'Задача успішно
змінена' });
    handleCloseModal(helpers);
   } catch (error) {
    notification.error({
     message: 'Не вдалося змінити задачу, спробуйте
пізніше',
     description: error.message,
    });
   }
  } else {
   try {
    await addTodo({ ...values, executionTime })
    notification.success({ message: 'Задача успішно додана'
});
    handleCloseModal(helpers);
   } catch (error) {
    notification.error({
     message: 'Не вдалося додати задачу, спробуйте
пізніше',
     description: error.message,
    });
   }
  }
 }


 const disabledDate = (current) => {
  return current && current < moment().endOf('day');
 }


 if (performersLoading) {
  <p>Loading...</p>
 }


 return (
  <Modal
   title={isEditMode ? "Редагування задачі" : "Створити
задачу"}
   visible={visible}
   onOk={() => formikRef.current.handleSubmit()}
   onCancel={onCancel}
  >
   <Formik
    enableReinitialize
    initialValues={initialValues}
```

```
      onSubmit={handleSubmit}
      validationSchema={validationSchema}
      innerRef={formikRef}
    >
      <Form className="w-100 px-4">
        <Space direction="vertical" size="middle" style={{ display:
'flex' }}>
          <DatePicker
            onChange={(e) => setExecutionTime(e)}
            value={executionTime}
            placeholder="Час виконання"
            style={{ width: '100%' }}
            disabledDate={disabledDate}
            showToday={false}
          />
          <Field
            name="title"
            placeholder="Найменування задачі"
            component={({ field, form, ...props }) =>
              <CustomInput props={{ ...field, ...props }}
name={field.name} />
            }
          />
          <Field
            name="description"
            placeholder="Короткий опис"
            component={({ field, form, ...props }) =>
              <CustomInput props={{ ...field, ...props }}
name={field.name} />
            }
          />
          <Field
            name="performer"
            placeholder="Виконавець"
            component={({ field, form, ...props }) => (
              <>
                <Select
                  {...props}
                  value={field.value}
                  onChange={e => form.setFieldValue(field.name, e)}
                  style={{ width: '100%' }}
                  name={field.name}
                >
                  {performers.map(option => (
                    <Select.Option
                      key={option.id}
                      value={option.id}
                    >
                      {`${option.name} - ${option.position}`}
                    </Select.Option>
                  ))}
                </Select>
                <p className={styles.error}>
                  <ErrorMessage name={field.name} />
                </p>
              </>
            )}
          />
          <Field
            name="status"
            component={({ field, form }) => (
              <Select
                value={field.value}
                defaultValue="backlog"
                onChange={e => form.setFieldValue(field.name, e)}
                style={{ width: '100%' }}
                name={field.name}
              >
                {statusOptions.map(option => (
                  <Select.Option key={option.id}
value={option.id}>{option.title}</Select.Option>
                ))}
              </Select>
            )}
          >
          </Field>
        </Space>
      </Form>
    </Formik>
  </Modal>
 )
}
```

api.js

```
import { initializeApp } from "firebase/app";

import { getDatabase, ref, update, set } from
"firebase/database";


const firebaseConfig = {

 apiKey: process.env.REACT_APP_FIREBASE_API_KEY,

 authDomain:
process.env.REACT_APP_FIREBASE_AUTH_DOMAIN,

 projectId: process.env.REACT_APP_FIREBASE_PROJECT_ID,

 storageBucket:
process.env.REACT_APP_FIREBASE_STORAGE_BUCKET,

 messagingSenderId:
process.env.REACT_APP_FIREBASE_SENDER_ID,

 appId: process.env.REACT_APP_FIREBASE_APP_ID,

 databaseURL:
process.env.REACT_APP_FIREBASE_DATABASE_URL,

};


export const firebaseApp = initializeApp(firebaseConfig);

const db = getDatabase();

export const getTodosRef = () => ref(getDatabase(firebaseApp),
'todos');


export const addTodo = (data) => {

 const timeNow = Date.now();

 set(ref(db, 'todos/' + timeNow), {

   ...data,

   created: timeNow,

   id: timeNow.toString(),

   executionTime: data.executionTime.format('YYYY-MM-DD'),

 });

}

export const updateTodo = (data) =>

 update(ref(db), {

   [`/todos/${data.id}`]: {

     ...data,

     executionTime: data.executionTime.format('YYYY-MM-DD'),

   }

 });


export const changeTodoStatus = (id, status) =>

 update(ref(db), {

   [`/todos/${id}/status`]: status,

 });


export const deleteTodo = (id) =>

 update(ref(db), {

   [`/todos/${id}`]: null,

 });


export const getPerformersRef = () =>
ref(getDatabase(firebaseApp), 'performers');


export const deletePerformer = (id) =>

 update(ref(db), {

   [`/performers/${id}`]: null,

 });


export const addPerformer = (data) => {

 const timeNow = Date.now();

 set(ref(db, 'performers/' + timeNow), {

   ...data,

   created: timeNow,

   id: timeNow.toString(),

   birthDate: data.birthDate.format('YYYY-MM-DD'),

 });

}

export const updatePerformer = (data) =>

 update(ref(db), {

   [`/performers/${data.id}`]: {

     ...data,

     birthDate: data.birthDate.format('YYYY-MM-DD'),

   }

 });



components/Column.js

import React from 'react';

import { Droppable } from 'react-beautiful-dnd';

import Task from './Task';

import styles from '../assets/styles.module.css';
```

```
const Column = ({ column, todos }) => (
 <div className={styles.column}>
  <h2>{column.title}</h2>
  <Droppable droppableId={column.id}>
   {(provided, snapshot) => (
    <ul
     className={styles.task__list}
     style={{
      backgroundColor: snapshot.isDraggingOver ? 'lightblue' :
'white',
     }}
     ref={provided.innerRef}
     {...provided.droppableProps}
    >
     {todos.map((todo, index) => (
      <Task key={todo.id} index={index} todo={todo} />
     ))}
     {provided.placeholder}
    </ul>
   )}
  </Droppable>
 </div>
);

export default Column;
```

components/Task.js

```
import React, { useState } from 'react';
import { Draggable } from 'react-beautiful-dnd';
import styles from '../assets/styles.module.css';
import { Card, Button, notification } from "antd";
import { EditOutlined, DeleteOutlined } from
'@ant-design/icons';
import TodoForm from "./TodoForm"
import { deleteTodo } from "../api";

const Task = ({ index, todo }) => {
 const [isOpenModal, setIsOpenModal] = useState(false);

 const handleCloseModal = () => {
  setIsOpenModal(false);
 }

 const handleDeleteTodo = () => {
  deleteTodo(todo.id);
  notification.success({ message: 'Задача успішно
видалена' });
 }

 const getTodoColor = () => {
  switch (todo.status) {
   case "backlog":
    return "#ff0000"

   case "in progress":
    return "#FFFF00"

   case "testing":
    return "#0000ff"

   case "done":
    return "#008000"

   default:
    break;
  }
 }

 return (
  <>
   <Draggable
    draggableId={todo.id}
    index={index}
   >
    {(provided, snapshot) => (
     <li
      className={styles.task}
      ref={provided.innerRef}
      {...provided.draggableProps}
      {...provided.dragHandleProps}
     >
```

```
        <Card
          title={todo.title}
          style={{
            backgroundColor: snapshot.isDragging ? 'lightsalmon' :
'white',
            borderColor: getTodoColor()
          }}
          extra={
            <>
              <Button
                type="primary"
                shape="circle"
                icon={<EditOutlined />}
                onClick={() => setIsOpenModal(true)}
              />
              <Button
                type="dashed"
                shape="circle"
                danger
                icon={<DeleteOutlined />}
                onClick={handleDeleteTodo}
              />
            </>
          }
        >
          <p>Опис - {todo.description}</p>
          <p>Виконавець -
{todo.performer.name}/{todo.performer.position}</p>
          <p>Срок - {todo.executionTime}</p>
          <p>Створена - {new
Date(todo.created).toLocaleDateString("ru-RU")}</p>
        </Card>
      </li>
    )}
  </Draggable>
  <TodoForm
    isEditMode
    visible={isOpenModal}
    todo={todo}
    onCancel={handleCloseModal}
  />
 </>
 );
};

export default Task;
```

components/TodoForm.js

```
import { useState, useRef, useEffect } from 'react';
import { Input, Space, DatePicker, Select, notification, Modal }
from 'antd';
import styles from '../assets/styles.module.css';
import { Formik, Form, Field, ErrorMessage } from 'formik';
import * as Yup from "yup";
import { addTodo, updateTodo, getPerformersRef } from
"../api";
import moment from "moment";
import { useListVals } from 'react-firebase-hooks/database';

const statusOptions = [
 {
  id: 'backlog',
  title: 'Задачі до реалізації',
 },
 {
  id: 'in progress',
  title: 'У роботі',
 },
 {
  id: 'testing',
  title: 'На тестуванні',
 },
 {
  id: 'done',
  title: 'Виконано',
 }
];

const emptyValues = {
 title: "",
 description: "",
 status: "backlog",
```

```
 performer: "",
}

const validationSchema = Yup.object({
 title: Yup.string().required("Обов'язково"),
 description: Yup.string().required("Обов'язково"),
 status: Yup.string().required("Обов'язково"),
 performer: Yup.string().required("Обов'язково"),
});

const CustomInput = ({ props, name }) => (
 <>
  <Input {...props} />
  <p className={styles.error}>
   <ErrorMessage name={name} />
  </p>
 </>
)

export default function TodoForm({
 onCancel,
 visible,
 isEditMode,
 todo
}) {
 const [performers, performersLoading] =
useListVals(getPerformersRef());
 const [executionTime, setExecutionTime] = useState(null);
 const formikRef = useRef();
 const [initialValues, setInitialValues] = useState(emptyValues);

 useEffect(() => {
  if (!isEditMode) return;
  setInitialValues({
   ...todo,
   performer: todo.performer.id,
  });
  const momentDate = moment(todo.executionTime,
'YYYY-MM-DD');
  setExecutionTime(momentDate);
 }, [visible])

 const handleCloseModal = (helpers) => {
  onCancel();
  setExecutionTime(null)
  helpers.resetForm();
 }

 const handleSubmit = async (values, helpers) => {
  if (!executionTime) {
   notification.error({ message: 'Треба вибрати час
виконання!' });
   return;
  }

  if (isEditMode) {
   try {
    await updateTodo({ ...values, executionTime });
    notification.success({ message: 'Задача успішно
змінена' });
    handleCloseModal(helpers);
   } catch (error) {
    notification.error({
     message: 'Не вдалося змінити задачу, спробуйте
пізніше',
     description: error.message,
    });
   }
  } else {
   try {
    await addTodo({ ...values, executionTime })
    notification.success({ message: 'Задача успішно додана'
});
    handleCloseModal(helpers);
   } catch (error) {
    notification.error({
     message: 'Не вдалося додати задачу, спробуйте
пізніше',
     description: error.message,
    });
   }
  }
 }
```

```
const disabledDate = (current) => {
  return current && current < moment().endOf('day');
}

if (performersLoading) {
  <p>Loading...</p>
}

return (
  <Modal
    title={isEditMode ? "Редагування задачі" : "Створити задачу"}
    visible={visible}
    onOk={() => formikRef.current.handleSubmit()}
    onCancel={onCancel}
  >
    <Formik
      enableReinitialize
      initialValues={initialValues}
      onSubmit={handleSubmit}
      validationSchema={validationSchema}
      innerRef={formikRef}
    >
      <Form className="w-100 px-4">
        <Space direction="vertical" size="middle" style={{ display: 'flex' }}>
          <DatePicker
            onChange={(e) => setExecutionTime(e)}
            value={executionTime}
            placeholder="Час виконання"
            style={{ width: '100%' }}
            disabledDate={disabledDate}
            showToday={false}
          />
          <Field
            name="title"
            placeholder="Найменування задачі"
            component={({ field, form, ...props }) =>
              <CustomInput props={{ ...field, ...props }} name={field.name} />
            }
          />
          <Field
            name="description"
            placeholder="Короткий опис"
            component={({ field, form, ...props }) =>
              <CustomInput props={{ ...field, ...props }} name={field.name} />
            }
          />
          <Field
            name="performer"
            placeholder="Виконавець"
            component={({ field, form, ...props }) => (
              <>
                <Select
                  {...props}
                  value={field.value}
                  onChange={e => form.setFieldValue(field.name, e)}
                  style={{ width: '100%' }}
                  name={field.name}
                >
                  {performers.map(option => (
                    <Select.Option
                      key={option.id}
                      value={option.id}
                    >
                      {`${option.name} - ${option.position}`}
                    </Select.Option>
                  ))}
                </Select>
                <p className={styles.error}>
                  <ErrorMessage name={field.name} />
                </p>
              </>
            )}
          />
          <Field
            name="status"
            component={({ field, form }) => (
              <Select
                value={field.value}
                defaultValue="backlog"
```

```
            onChange={e => form.setFieldValue(field.name, e)}
            style={{ width: '100%' }}
            name={field.name}
           >
            {statusOptions.map(option => (
              <Select.Option key={option.id}
value={option.id}>{option.title}</Select.Option>
            ))}
           </Select>
          )}
         >
         </Field>
       </Space>
     </Form>
   </Formik>
  </Modal>
 )
}
```

components/PerformersList.js

```
import { useState } from "react";
import { getPerformersRef, deletePerformer } from "../api";
import { Button, notification, Table, Space } from "antd";
import { EditOutlined, DeleteOutlined } from
'@ant-design/icons';
import { useListVals } from 'react-firebase-hooks/database';
import PerformerForm from "./PerformerForm";

const PerformersList = () => {
 const [performers, performersLoading] =
useListVals(getPerformersRef());
 const [modalMode, setModalMode] = useState(null);
 const [selectedPerformer, setSelectedPerformer] =
useState(null);

 const columns = [
  {
   title: 'ПІБ',
   dataIndex: 'name',
   key: 'name',
  },
  {
   title: 'Дата народження',
   dataIndex: 'birthDate',
   key: 'birthDate',
  },
  {
   title: 'Посада',
   dataIndex: 'position',
   key: 'position',
  },
  {
   title: 'Дії',
   key: 'action',
   render: (_, record) => (
    <Space size="middle">
     <Button
      type="primary"
      shape="circle"
      icon={<EditOutlined />}
      onClick={() => {
       setModalMode("edit");
       setSelectedPerformer(record);
      }}
     />
     <Button
      danger
      type="dashed"
      shape="circle"
      icon={<DeleteOutlined />}
      onClick={() => handleDeleteTodo(record)}
     />
    </Space>
   ),
  },
 ]

 const handleDeleteTodo = (record) => {
  deletePerformer(record.id);
  notification.success({ message: 'Виконавець успішно
видалений' });
 }
```

```
 if (performersLoading) {
  return <p>Loading...</p>
 }

 return (
  <>
   <Button
    type="primary"
    onClick={() => setModalMode("create")}
    size="large"
   >
    Створити виконавця
   </Button>
   <Table
    columns={columns}
    rowKey={record => record.id}
    dataSource={performers}
   />
   <PerformerForm
    isEditMode={modalMode === "edit"}
    visible={modalMode}
    data={selectedPerformer}
    onCancel={() => setModalMode(null)}
   />
  </>
 );
};

export default PerformersList;
```

components/PerformerForm.js

```
import { useState, useRef, useEffect } from 'react';
import { Input, Space, DatePicker, notification, Modal } from
'antd';
import styles from '../assets/styles.module.css';
import { Formik, Form, Field, ErrorMessage } from 'formik';
import * as Yup from "yup";
import { addPerformer, updatePerformer } from "../api";
import moment from "moment";

const emptyValues = {
 name: "",
 position: "",
 birthDate: null,
}

const validationSchema = Yup.object({
 name: Yup.string().required("Обов'язково"),
 position: Yup.string().required("Обов'язково"),
 birthDate: Yup.object().typeError("Обов'язково"),
});

const CustomInput = ({ props, name }) => (
 <>
  <Input {...props} />
  <p className={styles.error}>
   <ErrorMessage name={name} />
  </p>
 </>
)

export default function PerformerForm({
 onCancel,
 visible,
 isEditMode,
 data
}) {
 const formikRef = useRef();
 const [initialValues, setInitialValues] = useState(emptyValues);

 useEffect(() => {
  if (!isEditMode) return;
  const momentDate = moment(data.birthDate,
'YYYY-MM-DD');
  setInitialValues({
   ...data,
   birthDate: momentDate,
  });
 }, [visible])
```

```
 const handleCloseModal = (helpers) => {
  onCancel();
  helpers.resetForm();
 }

 const handleSubmit = async (values, helpers) => {
  if (isEditMode) {
   try {
    await updatePerformer(values);
    notification.success({ message: 'Виконавець успішно змінений' });
    handleCloseModal(helpers);
   } catch (error) {
    notification.error({
     message: 'Не вдалося змінити виконавця, спробуйте пізніше',
     description: error.message,
    });
   }
  } else {
   try {
    await addPerformer(values)
    notification.success({ message: 'Виконавець успішно доданий' });
    handleCloseModal(helpers);
   } catch (error) {
    notification.error({
     message: 'Не вдалося добавити виконавця, спробуйте пізніше',
     description: error.message,
    });
   }
  }
 }

 const disabledDate = (current) => {
  return current && current > moment().endOf('day');
 }

 return (
  <Modal
   title={isEditMode ? "Редагування виконавця" : "Створити виконавця"}
   visible={visible}
   onOk={() => formikRef.current.handleSubmit()}
   onCancel={onCancel}
  >
   <Formik
    enableReinitialize
    initialValues={initialValues}
    onSubmit={handleSubmit}
    validationSchema={validationSchema}
    innerRef={formikRef}
   >
    <Form className="w-100 px-4">
     <Space direction="vertical" size="middle" style={{ display: 'flex' }}>
      <Field
       name="name"
       placeholder="ПІБ"
       component={({ field, form, ...props }) =>
        <CustomInput props={{ ...field, ...props }} name={field.name} />
       }
      />
      <Field
       name="position"
       placeholder="Посада"
       component={({ field, form, ...props }) =>
        <CustomInput props={{ ...field, ...props }} name={field.name} />
       }
      />
      <Field
       name="birthDate"
       component={({ field, form, ...props }) =>
        <>
         <DatePicker
          {...props}
          onChange={(e) => form.setFieldValue('birthDate', e)}
          value={field.value}
          placeholder="Дата народження"
          style={{ width: '100%' }}
          disabledDate={disabledDate}
          showToday={false}
```

```
            />
            <ErrorMessage name="birthDate" />
           </>
          }
         />
        </Space>
      </Form>
    </Formik>
  </Modal>
 )
}
```

.env

```
REACT_APP_FIREBASE_API_KEY=AIzaSyBG27q3zmrroodm4YyI28dG6Eij7q_tr_A
REACT_APP_FIREBASE_AUTH_DOMAIN=dashboard-781fc.firebaseapp.com
REACT_APP_FIREBASE_PROJECT_ID=dashboard-781fc
REACT_APP_FIREBASE_STORAGE_BUCKET=dashboard-781fc.appspot.com
REACT_APP_FIREBASE_SENDER_ID=665601050724
REACT_APP_FIREBASE_APP_ID=1:665601050724:web:4162fa3727d35ac43b7771
REACT_APP_FIREBASE_DATABASE_URL=https://dashboard-781fc-default-rtdb.europe-west1.firebasedatabase.app
```

database.rules.json

```
{
 "rules": {
  ".read": true,
  ".write": true
 }
}
```

firebase.json

```
{
 "database": {
  "rules": "database.rules.json"
 },
 "hosting": {
  "public": "build",
  "ignore": [
   "firebase.json",
   "**/.*",
   "**/node_modules/**"
  ],
  "rewrites": [
   {
    "source": "**",
    "destination": "/index.html"
   }
  ]
 }
}
```

package.json

```
{
 "name": "dashboard",
 "version": "0.1.0",
 "private": true,
 "dependencies": {
  "@ant-design/icons": "^4.7.0",
  "@testing-library/jest-dom": "^5.11.4",
  "@testing-library/react": "^11.1.0",
  "@testing-library/user-event": "^12.1.10",
  "antd": "^4.16.13",
  "firebase": "^9.8.1",
  "formik": "^2.2.9",
  "moment": "^2.29.3",
  "react": "^17.0.2",
  "react-beautiful-dnd": "^13.1.0",
  "react-countdown": "^2.3.2",
  "react-dom": "^17.0.2",
  "react-firebase-hooks": "^5.0.3",
  "react-redux": "^7.2.5",
  "react-router-dom": "^6.3.0",
  "react-scripts": "4.0.3",
  "redux": "^4.1.1",
  "redux-devtools-extension": "^2.13.9",
  "redux-saga": "^1.1.3",
  "web-vitals": "^1.0.1",
```

```
    "yup": "^0.32.11"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <title>Dashboard</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this
app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

dashboard/vue

App.vue

```
<template>
  <a-layout>
    <a-layout-header>
      <a-menu mode="horizontal" theme="dark">
        <a-menu-item key="dashboard">
          <a
href="https://dashboard-781fc.web.app/dashboard">Дашбор
д задач</a>
        </a-menu-item>
        <a-menu-item key="performers">
          <a
href="https://dashboard-781fc.web.app/performers">Список
робітників</a>
        </a-menu-item>
        <a-menu-item key="static">
          <a>Статистика</a>
        </a-menu-item>
      </a-menu>
    </a-layout-header>
    <a-layout-content>
      <PieChart v-if="todos" :data="todos" />
    </a-layout-content>
  </a-layout>
</template>

<script>
import "ant-design-vue/dist/antd.css";
import { Layout, Menu } from "ant-design-vue/lib";
import PieChart from "./components/Chart";
import { dbRef } from "./firebase";
```

```
import { child, get } from "firebase/database";

import {
 Chart as ChartJS,
 Title,
 Tooltip,
 Legend,
 BarElement,
 CategoryScale,
 LinearScale,
} from "chart.js";

ChartJS.register(
 Title,
 Tooltip,
 Legend,
 BarElement,
 CategoryScale,
 LinearScale
);

export default {
 name: "App",
 components: {
  "a-layout": Layout,
  "a-layout-header": Layout.Header,
  "a-layout-content": Layout.Content,
  "a-menu": Menu,
  "a-menu-item": Menu.Item,
  PieChart,
 },
 data: () => ({
  todos: null,
 }),
 mounted() {
  get(child(dbRef, "todos"))
   .then((snapshot) => {
    if (snapshot.exists()) {
     const result = snapshot.val();
     const groupedTodos =
Object.values(result).reduce(function (r, a) {
      r[a.status] = r[a.status] || [];
      r[a.status].push(a);
      return r;
     }, Object.create(null));
     console.log(groupedTodos);
     const sortedTodos = [];
     Object.keys(groupedTodos)
      .sort()
      .forEach((column) => {
       sortedTodos.push(groupedTodos[column].length);
      });
     this.todos = sortedTodos;
    } else {
     console.log("No data available");
    }
   })
   .catch((error) => {
    console.error(error);
   });
 },
};
</script>

<style>
</style>
```

public/index.html

```
<!DOCTYPE html>
<html lang="">
 <head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport"
content="width=device-width,initial-scale=1.0">
  <link rel="icon" href="<%= BASE_URL %>favicon.ico">
  <title><%= htmlWebpackPlugin.options.title %></title>
 </head>
 <body>
  <noscript>
```

```
    <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly without JavaScript enabled. Please enable it to continue.</strong>
  </noscript>
  <div id="app"></div>
  <!-- built files will be auto injected -->
 </body>
</html>
```

components/Chart.js

```
import { defineComponent, h } from 'vue';

import { Pie } from 'vue-chartjs';
import {
 Chart as ChartJS,
 Title,
 Tooltip,
 Legend,
 ArcElement,
 CategoryScale,
} from 'chart.js';
ChartJS.register(Title, Tooltip, Legend, ArcElement, CategoryScale)

export default defineComponent({
 name: 'PieChart',
 components: {
  Pie,
 },
 props: {
  data: Object,
 },
 setup(props) {
  const chartData = {
   labels: ['Задачі до реалізації', 'Виконано', 'У роботі', 'На тестуванні'],
   datasets: [
    {
     backgroundColor: ['#ff0000', '#008000', '#FFFF00', '#0000ff'],
     data: props.data,
    }
   ]
  }
  const chartOptions = {
   responsive: true,
   maintainAspectRatio: false
  }

  return () =>
   h(Pie, {
    chartData,
    chartOptions,
    chartId: 'pie-chart',
   })
 }
})
```

firebase.js

```
import { initializeApp } from "firebase/app";
import { getDatabase, ref } from "firebase/database";

const config = {
 apiKey: "AIzaSyBG27q3zmrroodm4YyI28dG6Eij7q_tr_A",
 authDomain: "dashboard-781fc.firebaseapp.com",
 projectId: "dashboard-781fc",
 storageBucket: "dashboard-781fc.appspot.com",
 messagingSenderId: "665601050724",
 appId: "1:665601050724:web:4162fa3727d35ac43b7771",
 databaseURL: "https://dashboard-781fc-default-rtdb.europe-west1.firebasedatabase.app",
}

export const firebaseApp = initializeApp(config);
export const getTodosRef = () => ref(getDatabase(firebaseApp), 'todos');

export const dbRef = ref(getDatabase());
```

main.js

```
import { createApp } from 'vue'
import App from './App.vue'

createApp(App).mount('#app')
```

babel.config.js

```
module.exports = {
 presets: [
   '@vue/cli-plugin-babel/preset'
 ]
}
```

jsconfig.json

```
{
 "compilerOptions": {
   "target": "es5",
   "module": "esnext",
   "baseUrl": "./",
   "moduleResolution": "node",
   "paths": {
     "@/*": [
       "src/*"
     ]
   },
   "lib": [
     "esnext",
     "dom",
     "dom.iterable",
     "scripthost"
   ]
 }
}
```

package.json

```
{
 "name": "dashboard-vue",
 "version": "0.1.0",
 "private": true,
 "scripts": {
   "serve": "vue-cli-service serve",
   "build": "vue-cli-service build",
   "lint": "vue-cli-service lint"
 },
 "dependencies": {
   "ant-design-vue": "^3.2.3",
   "chart.js": "^3.7.1",
   "core-js": "^3.8.3",
   "firebase": "^9.8.1",
   "vue": "^3.2.13",
   "vue-chartjs": "^4.1.0"
 },
 "devDependencies": {
   "@babel/core": "^7.12.16",
   "@babel/eslint-parser": "^7.12.16",
   "@vue/cli-plugin-babel": "~5.0.0",
   "@vue/cli-plugin-eslint": "~5.0.0",
   "@vue/cli-service": "~5.0.0",
   "eslint": "^7.32.0",
   "eslint-plugin-vue": "^8.0.3"
 },
 "eslintConfig": {
   "root": true,
   "env": {
     "node": true
   },
   "extends": [
     "plugin:vue/vue3-essential",
     "eslint:recommended"
   ],
   "parserOptions": {
     "parser": "@babel/eslint-parser"
   },
   "rules": {}
 },
 "browserslist": [
   "> 1%",
   "last 2 versions",
   "not dead",
   "not ie 11"
 ]
```

}

vue.config.js

```
const { defineConfig } = require('@vue/cli-service')
module.exports = defineConfig({
 transpileDependencies: true
})
```