МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
19 травня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інформаційна технологія статистичного аналізу та керування даними»
здобувача групи ІН.м-11н Кончатного Віталія Володимировича

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

_____ Віталій КОНЧАТНИЙ
(підпис)

Керівник,
ст. викл., к.ф.-м.н. Оксана ШОВКОПЛЯС _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»
здобувача групи ІН.м-11н Кончатного Віталія Володимировича

1. Тема роботи: «Інформаційна технологія статистичного аналізу та керування даними» затверджую наказом по СумДУ від «08» травня 2023 р. № _____
2. Термін здачі здобувачем кваліфікаційної роботи до 19 травня 2023 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Огляд технологій, що застосовуються для керування даними та їх візуалізації. 2) Постановка завдання й формування завдань дослідження. 3) Огляд технологій, що використовуються під час розробки веб-додатків. 4) Моделювання системи розміщення додатків для керування даними та їх візуалізації. 5) Розробка додатку. 6) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Огляд технологій, що застосовуються для керування даними та їх візуалізації*		
2	*Постановка задачі та формування завдань дослідження*		
3	*Опис архітектури додатку*		
4	*Розробка додатку з використанням Next.js*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 68 стор., 18 рис., 1 додаток, 21 джерело.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки розкриває проблему керування даними та методів їх візуалізації шляхом розробки відповідної інформаційної технології.

Об'єкт дослідження – процес керування даними та їх візуалізації.

Мета роботи – розроблення веб-додатку платформи для розміщення віджетів, які виконують функції керування даними та їх візуалізації.

Методи дослідження – алгоритми візуалізації даних, статистичні методи та інструменти.

Результати – проведений аналіз літературних джерел, методів виконання та інструментів, які дозволяють реалізувати візуалізацію даних та створення веб-додатків, збереження та організація даних за допомогою документно-орієнтованої СКБД MongoDB, вивчені особливості застосування серверного рендерингу в веб-додатках, створених з використанням фреймворку Next.js. Після аналізу існуючих рішень було розроблено алгоритм роботи програми та її реалізація у вигляді веб-додатку. Додаток був реалізований за допомогою мови програмування JavaScript та фреймворку Next.js. Проведено тестування роботи сервісу.

ВІЗУАЛІЗАЦІЯ ДАНИХ, ІНФОРМАЦІЙНА СИСТЕМА КЕРУВАННЯ ДАНИМИ, DATA MANAGEMENT INFORMATION SYSTEM, DATA VISUALIZATION, MONGODB, NEXT.JS, REACT.

ЗМІСТ

ВСТУП

Актуальність. Зростання обсягів даних, які людина зустрічає в повсякденному житті, породжує проблему їх систематизації та оптимізації. Це призводить до потреби у створенні інструментів для аналізу фінансової, медичної, статистичної та інших видів інформації. Цю проблему вже доволі давно намагаються вирішити але такі рішення зазвичай є внутрішніми системами для виконання бізнес логіки на підприємствах і не надаються для широкого загалу. Такі системи не відрізняються універсальністю і можуть працювати тільки з внутрішніми процесами а на їх підтримку витрачаються значні кошти.

Об’єкт дослідження. Процес керування даними та їх візуалізація.

Предмет дослідження. Методологія візуалізації статистичних даних та керування ними в веб-розробці.

Гіпотеза. Візуалізацію статистичних даних та керування ними можна досягнути шляхом застосування інформаційної технології, що реалізує сервіс для розміщення додатків які виконують ці функції.

Наукова новизна. Описане у даній роботі програмне рішення дозволить досягти більшої ефективності в побудові інформаційної системи підприємства, дозволяючи автоматизувати внутрішні процеси та спростити аналіз та візуалізації статистичних даних.

Апробація матеріалів роботи. Основні результати роботи оприлюднені та обговорені на міжнародній науково-технічній конференції студентів та молодих вчених «Інформатика, математика, автоматика» (IMA – 2023).

Структура. Дана робота складається зі вступу, аналізу предметної області, постановки задачі дослідження, вибір архітектури та інструментів для рішення поставленої проблеми, опису розробки програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

Зв'язок роботи з науковою темою. Кваліфікаційна робота виконана на кафедрі комп’ютерних наук та пов’язана з виконанням науково-дослідної роботи № 0118U006971 «Методи, математичні моделі та інформаційні технології аналізу і синтезу інфокомунікаційних систем» (2018-2023).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Літературний огляд

1.1.1 Переваги та недоліки веб-додатків

Основна перевага веб-додатків перед іншими формами програмного забезпечення в універсальності. Саме це робить їх популярними адже спрощує взаємодію користувача з сервісом і відповідно є дуже привабливим для бізнесу через потенційно велику аудиторію. За допомогою веб-орієнтованих сервісів користувач має змогу отримувати або обробляти дані без необхідності завантажувати та встановлювати додаткове програмне забезпечення на свої пристрої, також витрачається менше часу на налаштування програмного забезпечення та його модулів. Користувачі мають доступ до даних та сервісів на будь якому пристрої який має доступ до мережі інтернет та браузер. Ця перевага має значну роль у сучасному світі де існує неймовірно велика кількість сервісів та організацій які працюють з кінцевим користувачем. Сучасний бізнес в більшості випадків повинен мати свій веб-сайт щоб відповідати вимогам суспільства.

Іншою перевагою веб-додатків є масштабування [1] та можливості для вдосконалення та розвитку сервісу. Функціонал таких додатків легко змінити чи розширювати для охоплення нових потреб користувачів. Це робить веб-додатки більш гнучкими та адаптивними до потреб людей які їх використовують, а також дозволяють розвиватися відповідно до покращення та еволюції технологій та вимог ринку. Окрім цього висока популярність веб-сервісів породжує проблему яка пов’язана з великою кількістю користувачів та суттєвим навантаженням на систему. Використання веб-орієнтованого сервісу допомагає відповідати на запити користувачів або збільшення їх кількості одним з методів масштабування. Завдяки технологіям хмарної інфраструктури та масштабування, веб-додатки можуть легко пристосуватися до збільшення обсягу роботи та забезпечити безперебійну роботу для всіх користувачів. Таким чином, веб-додатки здатні підтримувати безперебійну роботу, навіть якщо кількість користувачів значно

збільшиться. Це збільшує відмовостійкість всієї системи і зменшує втрати бізнесу через технічні проблеми.

Веб-додатки також забезпечують можливість доступу до однієї версії додатку для всіх користувачів. Це означає, що всі користувачі можуть працювати з однаковою версією додатку та використовувати всі його функції. Крім того, веб-додатки можуть бути легко сумісними з іншими додатками та системами які використовує бізнес. Це дозволяє компаніям легко інтегрувати свої веб-додатки з іншими системами, такими як системи управління відносинами з клієнтами, системи ведення обліку, системи електронної комерції та інші.

Незважаючи на переваги веб-додатків, є ряд недоліків, які можуть вплинути на їх продуктивність та ефективність. Одним з головних недоліків веб-додатків є залежність від швидкості та надійності Інтернет-з'єднання. Якщо мережеве з'єднання нестабільне або має занизьку швидкість, це може призвести до зниження продуктивності додатку або до збою в роботі додатку.

Крім того, веб-орієнтовані додатки не завжди можуть надавати доступ до всіх можливостей операційної системи. Наприклад, веб-додатки через те як вони влаштовані не можуть взаємодіяти з локальними файлами на комп'ютері користувача або запускати процеси в операційній системі. Це може бути необхідно для деяких завдань, таких як обробка відео або робота з файлами.

Ще одним недоліком веб-додатків є їх залежність від сторонніх сервісів, які можуть бути нестабільними або недоступними у деякий час через збої або технічні роботи. Наприклад, якщо веб-додаток використовує сторонній сервіс для зберігання користувацьких даних, і цей сервіс стає недоступним, це може призвести до втрати даних та зупинки роботи додатку.

Також веб-додатки можуть бути більш вразливими до кібератак [2], оскільки їх виконання залежить від браузера на стороні користувача та налаштувань безпеки браузера чи операційної системи. У залежності від налаштувань браузера та системи, деякі ризики безпеки можуть бути збільшені або зменшені.

Загалом, веб додатки є ефективними та зручними засобами для управління даними та візуалізації даних. Вони дозволяють забезпечити легкий доступ до даних з будь-якого місця та з будь-якого пристрою, підвищують ефективність роботи бізнесу, зменшують технічні проблеми та втрати бізнесу та надають можливості для розвитку та масштабування.

1.1.2 Фреймворки для створення веб-додатків

Розробка веб-додатків – це складний комплексний процес, який може займати багато часу та зусиль в розробників. Інженери програмного забезпечення повинні враховувати значну кількість факторів, включаючи проектування інтерфейсу користувача, зберігання та управління даними, оптимізацію продуктивності та безпеку. Фреймворки допомагають зробити процес розробки веб-додатків менш складним і більш організованим, надаючи зручний та більш структурований підхід до програмування та проектування. Такі засоби для розробки програмного забезпечення, надають структуровані методи та бібліотеки, що дозволяють швидко та ефективно створювати веб-додатки притримуючись принципів та стилістики коду властивим фреймворку.

Фреймворки забезпечують розробникам стандартні методи та структури проектів, які дозволяють зменшити витрати часу та зусиль на розробку додатків, а також привести структуру проекту до сталого виду який використовується в інших проектах. Ця уніфікація дозволяє розробникам простіше переключатись між проектами та швидше адаптуватися до нової роботи.

Найбільш популярні фреймворки для веб-розробки мовою JavaScript включають React, Angular та Vue.js [3]. Кожен з них підходить під різні цілі і є повноцінним інструментом з допомогою яких створено більшість існуючих веб-додатків на ринку.

React є одним з найпопулярніших фреймворків для розробки веб-додатків на JavaScript. Початково він створений для зручної розробки користувацьких інтерфейсів та підтримується Facebook і відкритою дуже широкою спільнотою

розробників, через що має дуже обширну документацію величезну бібліотеку додаткових модулів та прикладів використання.

Однією з головних особливостей React є використання компонентного підходу. Компоненти – це окремі блоки, які містять код розмітки, JavaScript-логіку та стилі. Такі компоненти можуть бути вкладені в інші компоненти, що дозволяє створювати складні інтерфейси з декількох компонентів які легко підтримувати через те, що при зміні файлу одного компонента він зміниться у всіх місцях де використовується. Це спрощує розробку, тестування та розуміння коду.

Також React використовує Virtual DOM – віртуальне дерево елементів, яке використовується для швидкого та ефективного відображення змін на сторінці без необхідності перезавантажувати всю сторінку чи всі компоненти на ній [4]. Це зменшує витрати часу та ресурсів системи на оновлення інформації на сторінці та робить додаток більш продуктивним та швидким.

React має велику кількість розширень та бібліотек, які дозволяють значно розширити функціональність додатків та мають широкий вибір інструментів для розробки. Окрім цього, React може бути використаний для розробки як веб-додатків, так і додатків для мобільних платформ з використанням React Native. Це дозволяє розробникам створювати універсальний додаток для якого використовується одна технологія та схожий код, це значно зменшує витрати на розробку та підтримку додатків.

Альтернативою до React може бути Angular, він є JavaScript фреймворком, який був розроблений Google. Один з головних принципів Angular – це модульність розроблюваного коду, що дозволяє побудувати додаток з окремих компонентів, які легко можна використати в інших проектах чи задачах. Angular також має достатньо велику кількість готових бібліотек та інструментів, що спрощують розробку продуктів.

Ключовий компонент Angular – це директиви. Вони дозволяють використовувати HTML як мову для шаблонів та допомагають організувати

простий та зрозумілий спосіб взаємодії з веб-додатком. Крім того, Angular має систему залежностей, яка дозволяє легко керувати компонентами та їх залежностями. Основною особливістю Angular є підтримка двостороннього зв'язку даних, що дозволяє автоматично вносити зміни в дані на стороні клієнта або сервера. Такий підхід забезпечує миттєву взаємодію з додатком та зменшує кількість запитів на сервер.

Angular також як і React підтримує розробку мобільних додатків за допомогою Ionic Framework, який базується на Angular та дозволяє розробляти мобільні додатки з багатим інтерфейсом користувача для iOS та Android.

Однак, в Angular є деякі недоліки. Наприклад, Angular має досить складну структуру та вимагає від розробників вивчення багатьох нових понять та концепцій.

Насамкінець Vue.js є популярним фреймворком для розробки веб-додатків, в основному в країнах Азії та Тихоокеанського регіону. Цей рішення є дуже легким та простим у використанні та вивченні, це зробило його затребуваним серед розробників та бізнесу. Vue.js використовує Virtual DOM як і React, що дозволяє ефективно маніпулювати станом додатку та мінімізувати затримки відображення даних. Vue.js має багато вбудованих функцій, таких як компоненти, директиви та фільтри, які спрощують розробку та покращують якість коду. Цей фреймворк також має хорошу документацію та активну спільноту розробників, тому проблеми які можуть виникнути часто мають декілька готових рішень.

Особливість Vue.js це його підхід до компонентної архітектури. Кожен елемент додатку вважається компонентом, таким чином розробники можуть створювати зручні та адаптовані для повторно використання блоки коду. Це полегшує підтримку коду додатку в чистому та організованому стані, а також спрощує тестування та розгортання.

Але, порівняно з React, Vue.js може бути менш гнучким і має проблеми з масштабованістю для розробки великих проєктів. Також, Vue.js не так добре

підтримує серверний рендеринг, що може стати проблемою для деяких веб-додатків.

Розробка веб-додатків може бути складною та доволі витратною задачею, і такі рішення як фреймворки можуть допомогти розробникам зробити процес розробки більш організованим та ефективним. Використання фреймворків є важливим кроком в ефективній розробці веб-додатків, який дозволяє забезпечити організованість, структурованість та ефективність в процесі розробки.

1.1.3 Методи візуалізації даних у веб-додатках

Дані в веб-додатках мають бути не лише збережені, але й відображені в зручному для сприйняття форматі. Візуалізація даних дозволяє зрозуміти та проаналізувати велику кількість інформації в більш зручному та інтуїтивно зрозумілому форматі, що дозволяє приймати більш обґрунтовані рішення. Візуалізація є важливим етапом в процесі аналізу та представлення даних, який допомагає краще зрозуміти складні зв'язки та тренди в наборах даних. З розвитком веб-технологій та фреймворків для клієнтської частини, стало можливим створення складних візуальних ефектів та динамічних інтерактивних візуалізацій без великої кількості коду та ресурсів. Бібліотеки та інструменти, які надаються для візуалізації даних, роблять цей процес швидким та ефективним, дозволяючи розробникам взаємодіяти з даними та відображати їх у вигляді привабливих та легко зрозумілих візуальних компонентів та модулів[5].

Однією з найпопулярніших бібліотек на JavaScript для роботи з візуалізації даних є D3.js. Ця бібліотека дозволяє створювати складні та інтерактивні візуалізації даних, що можуть бути використані в різних сферах, включаючи науку про дані, веб-аналітику та інші. D3.js є дуже потужним інструментом для візуалізації даних та забезпечує багато можливостей для створення різноманітних типів візуалізацій, таких як графіки, діаграми, карти, візуалізації мереж, тощо. Для роботи з D3.js необхідно мати базові знання мови JavaScript та HTML/CSS. Бібліотека має потужний API, який дозволяє використовувати

велику кількість методів та функцій для роботи з даними та створення візуалізацій. Крім того, D3.js має велику спільноту розробників, яка допомагає знайти відповіді на більшу частину запитань, які можуть виникнути під час використання бібліотеки.

Серед недоліків D3.js можна відзначити дещо складний для новачків API та загальну складність створення більш складних візуалізацій та графіки. Крім того, часто користувачі скаржаться на те, що ця бібліотека не є достатньо ефективною для завдань які передбачають роботу з великими обсягами даних та є недостатньо зручною для використання з іншими фреймворками та бібліотеками.

Окрім D3.js в проектах в яких є необхідність в візуалізації даних часто використовують Chart.js [6]. Це JavaScript бібліотека для візуалізації даних, яка надає зручні та прості інтерфейси для створення різноманітних графіків та діаграм. Ця бібліотека підтримує велику кількість типів діаграм, включаючи лінійну, стовпчасту, колову, радарну та ієрархічну. Цього набору зазвичай достатньо для задоволення потреб більшості проектів.

Однією з головних переваг Chart.js є його простота використання та легкість налаштування для будь якого типу проекту. Бібліотека використовує простий та добре документований API, що дозволяє створювати графіки додавши лише декілька рядків коду. Цей інструмент також має велику кількість налаштувань та можливостей для налаштування зовнішнього вигляду графіків та елементів, що дозволяє легко налаштувати діаграми під свої потреби та вписувати їх у дизайн додатка. Ще однією перевагою Chart.js є те, що вона підтримує адаптивний дизайн, тобто графіки можуть підлаштовуватись для різних розмірів екранів, що дозволяє зручно відображати дані на різних пристроях. Також існує велика кількість плагінів та розширень які додають новий функціонал або розширюють стандартний. Такі плагіни легко налаштовувати та встановлювати в проекті завдяки підходу до налаштування

графіків. Chart.js є більш простою альтернативою D3.js. Її основна мета – надати швидке та просте рішення для створення графіків та діаграм.

D3.js є потужним інструментом для створення високопродуктивних та динамічних візуалізацій, але вимагає великої кількості коду та експертизи для використання. Якщо ви шукаєте більш простий варіант, варто розглянути Chart.js, який забезпечує зручний та швидкий спосіб відображення даних в графічному вигляді. Існують також інші бібліотеки та фреймворки, такі як Google Charts, Highcharts та Plotly, які можуть бути корисні для візуалізації даних в залежності від потреб та вимог проекту. Загалом, візуалізація даних – це важлива частина багатьох проектів, тому використання найбільш оптимальної для додатку бібліотеки може значно полегшити процес.

1.1.4 Можливості технології в різних сферах

Технологія керування даними та аналітики може бути використана в різних галузях та сферах через свою універсальність, та може бути застосована для великої кількості задач. Більшість задач які виникають в бізнесі потребують обробки інформації та її аналізу, а також управління нею. Ці завдання можна порівняно легко вирішити, якщо застосувати розділення задач на менші та автоматизувати їх. Ці технології дозволяють не тільки ефективніше управляти даними, але й отримувати нові знання та навички, що допомагає в прийнятті кращих рішень та вдосконаленні бізнес-процесів на підприємстві.

Наприклад на підприємстві яке керує виробництвом можна застосувати цю технологію для контролю за виробничим процесом. Невеликі віджети, які є веб-додатками можуть допомагати робітникам вносити звітність або виконувати рутинні задачі. Частина додатків які використовує підприємство може бути призначена для керівництва установи і будуть відображати звітність та мати функції генерації документів для податкової або будь яких інших цілей. Все разом це може працювати як повноцінна система керування установою з можливістю легко розширювати функціонал залежно від потреб. Це має

збільшити ефективність підприємства та зменшити витрати, що має позитивно відобразитись на загальних прибутках.

У сфері освіти така технологія може бути використана в декількох варіантах розвитку системи. Наприклад кожен міні-додаток може відповідати за статистику по одному з структурних відділів. Також може бути реалізований набір необхідних інструментів таких як відображення розкладу чи сповіщення працівників тощо. До системи можуть мати доступ не тільки викладачі та персонал установи а також студенти, для них може бути реалізовано свій набір сервісів який необхідний для навчання чи конкретних предметів. Таким чином всі учасники освітнього процесу отримають необхідні їм сервіси доступні в будь якому місці де є мережа інтернет, що актуально в сучасних умовах та збільшеному попиту на віддалене навчання.

Також технологія керування даними та аналітики має великий потенціал для використання в фінансовому секторі. Вона може бути використана для оптимізації бухгалтерського обліку, управління витратами та планування бюджету. Це покликано допомогти фінансовим установам знизити витрати та покращити ефективність ведення бізнесу. Крім того, ця технологія може бути використана для створення персоналізованих фінансових продуктів та послуг для клієнтів, що має підвищити їх задоволеність та лояльність.

Отже, технологія керування даними та аналітики може бути використана в різних сферах, включаючи бізнес, освіту та фінансовий сектор. Використання таких інструментів може значно спростити роботу в цих сферах, що дозволить автоматизувати багато рутинних процесів та відслідковувати статистику в реальному часі. Крім того, це рішення дозволяє більш ефективно оброблювати дані та аналізувати їх, що може призвести до підвищення ефективності та прибутковості підприємств. Таким чином, використання цієї технології має потенціал для покращення роботи у різних галузях та може стати ключовим інструментом для досягнення успіху в сучасному світі.

1.2 Огляд наявних рішень

У нашому сучасному світі, де кожна галузь та сфера діяльності має потребу у зборі, обробці та аналізі даних, інформаційні технології відіграють ключову роль у вирішенні цих та інших завдань. Від бізнесу до медицини, від науки до соціальних мереж – всі вони прагнуть досягти максимальної ефективності та оптимізувати свої процеси за допомогою використання технологій які допомагають керувати даними та візуалізувати для розуміння людиною. Щоб з'ясувати, як вони можуть забезпечити допомогу для різних галузей та сфер бізнесу було розглянуто найпопулярніші сервіси та програмні рішення для керування даними та аналітики. Також досліджено можливості та переваги кожного з цих рішень щоб виділити ключові аспекти які можуть допомогти при розробці.

Один з найпопулярніших сервісів для керування даними та аналітики – це сервіс Tableau [7]. Це програмне забезпечення дозволяє створювати візуальні звіти та графіки на основі даних з різних налаштовуваних джерел, таких як бази даних, електронні таблиці та інші (рис. 1.1). Однією з найбільших переваг Tableau є інтуїтивний і простий у використанні інтерфейс, що дозволяє користувачам легко взаємодіяти з даними та виконувати аналіз. Такий підхід покращує користувацький досвід та зменшує час на виконання типових завдань.

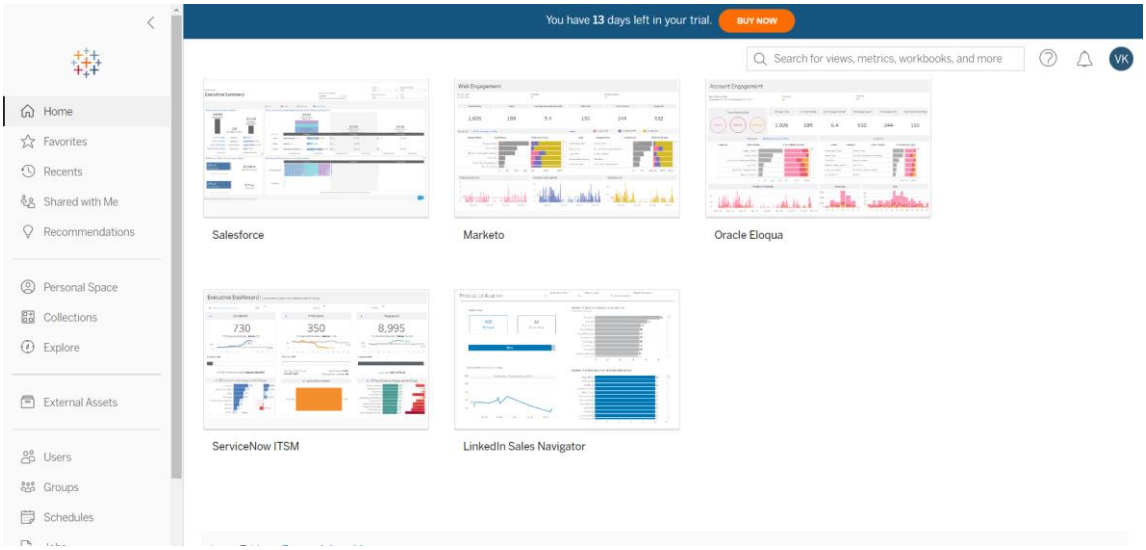

Рисунок 1.1 – Інтерфейс Tableau

Проте, на жаль, Tableau має ряд недоліків. Перш за все, це використання сервісу може бути досить затратним, зокрема для невеликих підприємств та бюджетних установ які не мають зайвих коштів на використання цієї технології. Крім того, часто користувачі вказують на обмеження в зберіганні та обробці великих обсягів своїх даних, що може бути значною проблемою для більших компаній з великою кількістю даних.

Другий варіант сервісу для керування даними та аналітики – це Power BI від Microsoft [8]. Цей інформаційний сервіс дозволяє порівняно швидко та ефективно обробляти великі обсяги даних з різних джерел, створювати графіки, таблиці та інші візуалізації для аналізу даних. Power BI пропонує також можливості для спільної роботи та співпраці між користувачами, дозволяючи обмінюватися даними та створювати спільні звіти (рис. 1.2).

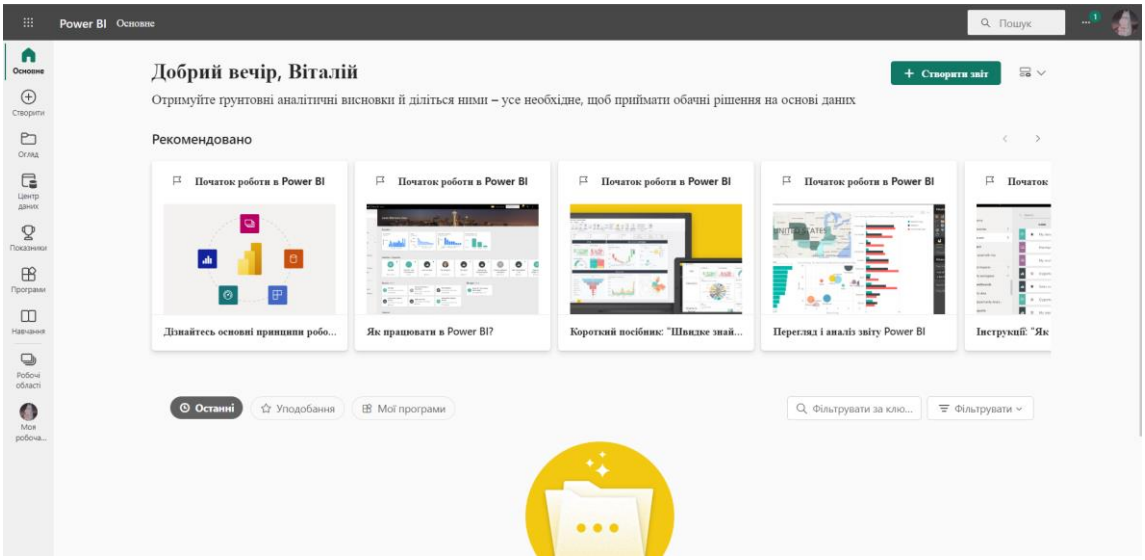

Рисунок 1.2 – Інтерфейс Power BI

Однією з ключових переваг Power BI є те, що це продукт який пропонує Microsoft, це означає, що він інтегрується з іншими продуктами цієї компанії, такими як Excel, OneDrive, SharePoint та іншими професійними рішеннями як для бізнесу так і для індивідуальних користувачів. Це дозволяє використовувати дані з цих продуктів у Power BI та використовувати його для аналізу даних, що були створені або збережені в Excel або в SharePoint. Крім того, Power BI пропонує безкоштовну версію, яка містить базові можливості для аналізу даних, та платні

пакети з додатковими функціями. Це допомагає ознайомитись з функціоналом і опанувати його не витрачаючи кошти.

Однак, Power BI має свої недоліки, такі як складність у використанні для новачків та обмежена підтримка відкритих форматів даних, що може ускладнити роботу з іншими програмними продуктами та сервісами. Також він не надає розширених можливостей для статистичного аналізу даних як інші сервіси.

Загалом, обидва сервіси – є потужними інструментами для аналізу та візуалізації даних. Кожен з них має свої переваги та недоліки, і вибір такої технології залежить від конкретних потреб користувача та характеру даних, які необхідно обробити. Використання таких сервісів може значно полегшити процес роботи з даними, підвищити ефективність та якість прийняття рішень в бізнесі та інших сферах.

1.3 Постановка завдання

Основною метою створення інформаційної технології є оптимізація роботи з даними та можливість розширювати функціонал додатку за рахунок архітектури, яка дозволяє швидко і відносно просто додавати нові можливості у вигляді окремих додатків.

Розроблений програмний продукт має задовольняти наступні вимоги:

- зручна можливість додавати нові програмні рішення в вигляді додатків;
- авторизація користувачів та можливість збереження стану віджетів для конкретного користувача;
- приклад віджету який демонструє роботу з даними;
- внутрішня нотифікація користувачів;
- демонстрація інтеграції з зовнішніми сервісами.

Для досягнення поставленої мети необхідно вирішити такі задачі:

1) виконати аналіз проблемної області;
2) провести аналіз аналогів програмних продуктів;
3) обрати технології та архітектуру проекту;

4) реалізувати модель інформаційної системи.

Предметом дослідження є методологія візуалізації статистичних даних та керування ними в веб-розробці.

Наукова новизна полягає в тому, що розроблене програмне рішення, яке описане у цій роботі, дозволить підприємству досягти ефективності в роботі з даними візуалізуючи їх для спрощення аналізу та надаючи інструменти для керування ними.

Практичною значимістю буде те, що застосування розроблюваного веб-додатку зменшить кількість ресурсів витрачених на побудову інформаційної інфраструктури підприємства, та дозволить оптимізувати його процеси.

2 ОГЛЯД АРХІТЕКТУРИ ТА ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ

2.1 Огляд архітектури розроблюваного додатка

Для створення ефективного та функціонального додатка необхідно попрацювати над його архітектурою. Архітектура додатка – це сукупність засобів та методів, які використовуються для його побудови та розробки. Вона описує структуру додатку, його компоненти та залежності між ними [9]. Архітектура має бути ефективною і забезпечувати відповідність продукту до вимог задачі.

Важливо розробляти архітектуру додатку з урахуванням його майбутнього розвитку та можливості масштабування у випадку збільшення кількості користувачів. Правильно спроектована архітектура дозволяє забезпечити гнучкість та легкість модифікації додатку в майбутньому, а також зменшити вартість його розробки та підтримки коли додаток вже запущено і ним користуються юзери.

Одним із способів реалізації функціональної та гнучкої архітектури додатку є використання віджетів [10]. Віджети – це окремі компоненти, які можуть бути використані для додавання функціональності до додатку та містять у собі окремі функції та можливості. Вони дозволяють створювати складні додатки, що складаються з невеликих компонентів, які можуть бути легко модифіковані та використані ще раз в майбутньому. Використання віджетів дозволяє зменшити кількість коду, що потрібно написати, оскільки багато з них може бути згенеровано автоматично зі спеціальних бібліотек або додано як окремі вже працюючі додатки різних сервісів. Крім того, відокремлені додатки забезпечують більшу безпеку та масштабованість, оскільки проблеми з одним додатком не впливають на інші. Загалом, архітектура на основі віджетів є потужним інструментом для створення зручних та ефективних додатків які можуть приносити користь в різних галузях та на різних підприємствах [11].

Архітектура додатку буде складатися з основної частини сервісу яка представляє собою платформу для розповсюдження та відтворення вмісту додатків, а також самих додатків які по факту є незалежними React компонентами. Такі додатки можна легко змінити чи розширити функціонал за потреби, без змін в коді самого сервісу. Достатньо лише створити додаток за шаблоном та підключити його одним з способів, створивши його файл в окремій директорії для додатків або підключивши з стороннього джерела вже скомпонований пакет як динамічний компонент. Окрім цього додатки матимуть доступ до деяких компонентів самого сервісу, наприклад для реалізації функціоналу нотифікації користувачів або збереження свого стану в базі даних (рис. 2.1).

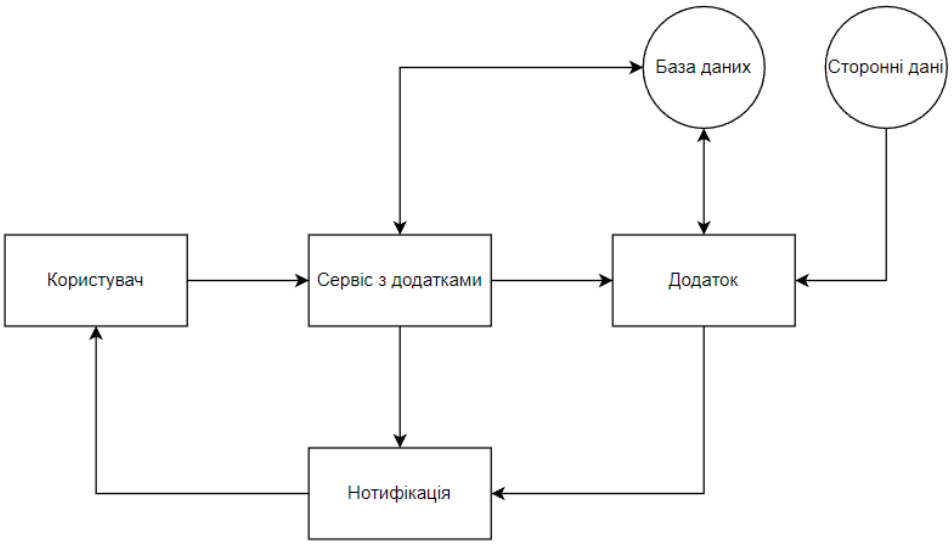

Рисунок 2.1 – Схема архітектури сервісу

Збереження стану додатка може бути реалізовано як один стан для одного користувача так і в формі одного стану віджету для групи користувачів. Такий підхід може допомогти в реалізації функції суспільної роботи над додатком для команд користувачів.

Окремо варто звернути увагу на архітектуру самих додатків (рис. 2.2), для їх створення пропонується використати три основні принципи які мають впорядкувати віджети та зробити їх уніфікованими та ефективним.

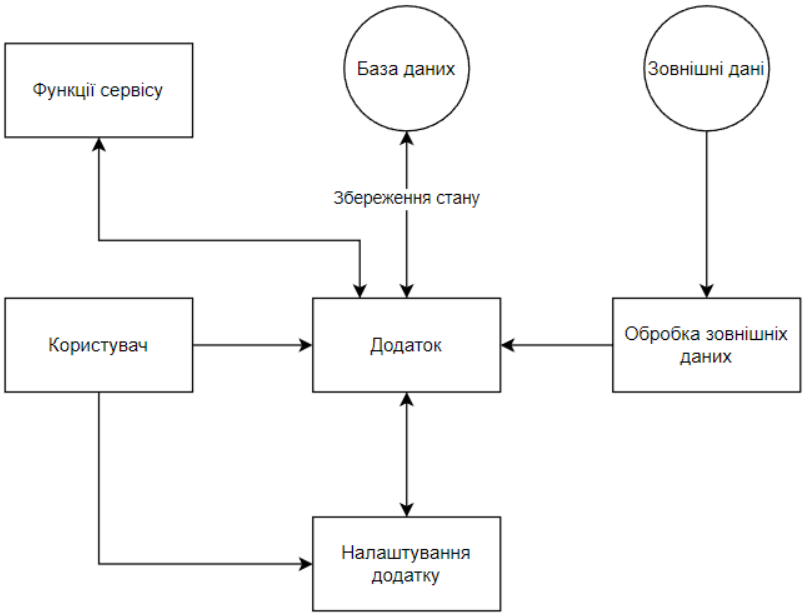

Рисунок 2.2 – Схема архітектури додатку

Першим принципом створення додатків для такого сервісу є «Компонент». Кожен віджет має бути реалізованим як компонент React і має записувати свій стан до власного сховища. Вміст віджету передається в шаблон віджету і відкривається на сторінці віджетів. Цей принцип в архітектурі додатків для сервісу підтримує ідею модульності і розділення функціоналу на окремі компоненти, що спрощує розробку і підтримку додатків. Кожен віджет, як окремий компонент, має свою внутрішню логіку та може мати свій власний стан, що дозволяє зберігати дані на стороні клієнта. Використання компонентів дозволяє зручно редагувати та доповнювати функціонал додатків, без впливу на інші частини системи. Шаблон віджету відповідає за його базовий вигляд на сторінці віджету. Цей принцип організує зручну і просту систему для розробки та розширення додатків, забезпечуючи збереження чистоти та організації коду.

Другим принципом для успішної реалізації можна виділити «Функцію». Кожен віджет має виконувати чітку і означену функцію, не має бути зайвого

функціоналу, який не пов’язаний з основною функцією віджету. Результат або проміжні результати мають бути записані в сховище, що гарантує чистоту та точність виконання функцій віджета. Це також забезпечує відсутність конфліктів з даними при доступі до зовнішніх джерел. Надмірний функціонал може зробити віджет складним у використанні, що призведе до того, що користувачі відмовляться від його використання. Тому, важливо зосередитись на основній функції віджету, щоб забезпечити якість та ефективність роботи всього сервісу.

Останній принцип – це «Вид». Віджети можуть використовувати будь-які необхідні бібліотеки та ресурси, щоб забезпечити певний функціонал який заданий його функцією. Але дуже важливо дотримуватися загальної стилістики сервісу, щоб уникнути суперечностей між віджетами та сервісом. Наприклад, якщо сервіс використовує специфічну тему оформлення, то всі віджети мають використовувати ті ж самі кольори, шрифти та стилі. Це не тільки забезпечує єдність дизайну, але також допомагає забезпечити гладку та просту інтеграцію віджетів в додаток.

Таким чином, використання віджетів дозволяє створювати додатки, які легко масштабувати та модифікувати, що є важливою перевагою для розвитку проекту у майбутньому. Також, використання віджетів дозволяє побудувати додатки, які можуть бути повністю автономними та незалежними один від одного. З цим підходом, додатки можуть розвиватись та змінюватись окремо, не впливаючи на функціонал сервісу в цілому, а також дозволяє створювати додатки з єдиним дизайном та стилістикою. Використання віджетів також забезпечує високу перевикористовуваність коду, що зменшує час та зусилля, потрібні для розробки нових додатків або модифікації наявних. Крім того, віджети можуть бути перенесені з одного проекту в інший, що дозволяє значно економити час та зусилля розробника. Використання віджетів також сприяє більшій гнучкості та швидкості в розробці, адже окремі віджети можуть бути розроблені паралельно та інтегровані в додаток, що дозволяє розробникам працювати одночасно над різними частинами додатка та зменшує час розробки

в цілому. Крім того, використання віджетів дозволяє забезпечити більшу стабільність та безпеку, оскільки окремі віджети можуть бути тестовані та підтримувані окремо від інших частин додатку.

2.2 Вибір фреймворків та бібліотек

Використання правильних технологій є важливим елементом розробки будь-якого проекту, оскільки вони забезпечують ефективність, масштабованість та надійність програмного продукту. Вибір технологій для розробки сервісу був здійснений з урахуванням можливості майбутнього розширення та модифікації додатків. Використані технології дозволяють ефективно працювати з базою даних, забезпечують швидкий відгук на дії користувачів та надають великі можливості з візуалізації даних. Тому такий вибір повинен позитивно вплинути на процес розробки сервісу та додатків.

Основною технологією яку можна використати для розробки сервісу є Next.js. Next.js – це фреймворк для розробки веб-додатків на базі React, який дозволяє розробникам швидко і якісно створювати веб-додатки з високою продуктивністю та підвищеною масштабованістю [12]. Використання React з Next.js дозволяє забезпечити швидку відповідь веб-додатків та високу швидкість завантаження сторінок через використання технології серверного рендерингу [13], збільшуючи зручність користувачів та покращуючи їхнє враження від використання додатку. Next.js набуває все більшої популярності. Так як він базується на React та додає до нього додаткові функції та можливості, такі як серверний рендеринг та статичний експорт сторінок, в сучасному світі розробки веб додатків йому приділяють все більше уваги. Він дозволяє розробникам побудувати додаток з більш високою продуктивністю та розширюваністю через особливості побудови додатків з його допомогою, а також забезпечує кращу оптимізацію для SEO. Next.js також дозволяє просто налаштувати різні параметри, такі як маршрутизація, а також підтримка CSS модулів, що спрощує роботу дизайнера проекту.

Загалом, використання Next.js дозволяє розробникам зосередитися на функціональності додатку, а не на налаштуванні інфраструктури. Це зменшує час на розробку та впливає на комфорт розробника і безвідмовність системи в цілому, через те, що не вносивши зміни в налаштування більшості основних процесів неможливо щось зламати.

Наступна бібліотека може бути використана для створення графічного інтерфейсу додатків та самого сервісу. Material-UI надає колекцію вже готових для використання React компонентів з широкими можливостями для налаштування [14]. Цей інструмент значно економить час адже не потрібно налаштовувати різні поля, кнопки чи графічні елементи або створювати їх вручну. Достатньо просто додати на сторінку вже готовий елемент та задати йому невелику кількість параметрів. Бібліотека Material-UI надає широкий вибір компонентів з різними стилями та можливостями налаштування, що дозволяє створювати графічний інтерфейс, який відповідає потребам проекту та його користувачів.

Ця бібліотека надає гнучкий інтерфейс для налаштування вигляду компонентів, що дозволяє створювати додатки зі специфічним дизайном та стилістикою. Крім того, Material-UI підтримує відповідність стандартам дизайну Google Material Design, що забезпечує єдиний стиль вигляду елементів на всіх сторінках додатку. Компоненти Material-UI також добре оптимізовані, що дозволяє робити швидкий та ефективний рендеринг сторінок. Крім того, наявність документації та активної спільноти розробників, які активно підтримують проект, робить Material-UI одним з найкращих виборів для розробки інтерфейсу користувача.

Для реалізації функцій які зачіпають візуалізацію даних можна використати бібліотеку побудови графіків і діаграм Chart.js. Цей універсальний інструмент дозволяє використовувати в віджетах які розміщені на розроблюваній платформі засоби з побудови різних типів діаграм та візуалізувати дані які завантажені в віджет користувачем. Також, Chart.js є досить легкою у використанні та має

зручну і обширну документацію, що забезпечує ефективний процес розробки візуальних елементів на платформі. Крім того, вона підтримується активною спільнотою розробників та має відкритий код, що дозволяє швидко вирішувати проблеми та знаходити нові можливості для розробки функціональних візуальних елементів на платформі. Не менше цьому допомагає величезна кількість плагінів, що дозволяє значно розширити можливості візуалізації даних в проекті.

Гнучкість архітектури проекту дозволяє легко і швидко інтегрувати різноманітні інструменти, що розширює можливості та функціонал розроблюваної платформи. Наприклад щоб додати інтеграцію з сервісом для планування Trello можна підключити бібліотеку trello-for-wolves [15]. Це дозволить розроблюваному віджету зберегти результат в картку на обраній дошці в цьому сервісі, або виконати інші дії пов’язані з цією платформою, залежно від поставленого завдання яке має виконувати віджет.

Крім інтеграції з сервісом Trello, архітектура проекту може дозволити інтегрувати інші популярні сервіси за допомогою бібліотек які можна підключити, такі як Slack, Asana, Google Drive та інші. Це може бути корисно для команд, які використовують різноманітні інструменти для співпраці та організації своєї роботи. Особливості архітектури дозволять швидко змінювати та доповнювати функціонал платформи в залежності від потреб користувачів та ринкових умов. Це може допомогти забезпечити конкурентоспроможність продукту та задоволення потреб користувачів.

В цілому, можливості вибору інструментів дуже широкі завдяки відкритій архітектурі проекту. Це дає змогу обирати ті інструменти, які найкраще відповідають потребам проекту та дозволяють досягати бажаних результатів. Така гнучкість дозволяє розробникам використовувати не тільки стандартні інструменти, але і спеціалізовані бібліотеки для різних галузей. Наприклад, для взаємодії з API стороннього сервісу можна використати бібліотеку Axios, яка дозволяє виконувати HTTP запити і отримувати відповіді в потрібному форматі.

Загалом, використання гнучкої архітектури та багатофункціональних бібліотек дозволяє розширювати можливості проекту та втілювати різноманітні ідеї.

2.3 Спосіб автентифікації користувачів

Процес автентифікації є важливим етапом для будь-якого додатку, який працює з будь якою інформацією користувачів. Основною метою автентифікації є перевірка особи користувача і забезпечення доступу до його персональних даних тільки для нього самого. Це може бути досягнуто за допомогою різноманітних методів та підходів, таких як введення логіна і пароля, використання соціальних мереж, а також використання спеціальних інструментів, наприклад Telegram Login Widget який можна застосувати в розроблюваному додатку.

Telegram Login Widget – це інструмент, що дозволяє користувачам ввійти в додаток за допомогою свого Telegram акаунту [16]. Це рішення є інтегрованою платформою для авторизації користувачів, що забезпечує безпеку та конфіденційність даних користувача. Для використання Telegram Login Widget потрібно мати Telegram бота та отримати його ключ доступу.

Після підключення Telegram Login Widget в додаток, користувач може здійснювати вхід за допомогою свого Telegram акаунту, натиснувши кнопку "Увійти з Telegram" на сторінці авторизації. Після цього відбувається перенаправлення на сторінку входу в Telegram, де користувач має ввести свій номер телефону та натиснути кнопку "Надіслати код". Після цього на вказаний номер телефону або в відкритий додаток месенджера приходить повідомлення з кодом для входу або кнопкою після натискання на яку користувач підтверджує свою особу на сторінці додатку. Після успішної автентифікації користувач отримує доступ до його функціоналу.

Використання цього способу автентифікації має декілька переваг, зокрема він є безпечним, зручним та простим у використанні. Telegram Login Widget забезпечує зв'язок між користувачем і додатком, це зменшує кількість даних, що потрібно надавати користувачеві для реєстрації та входу в систему. Загалом,

використання Telegram Login Widget є ефективним способом автентифікації користувачів у додатку, що забезпечує зручність, безпеку та простоту імплементації.

2.4 Вибір системи керування базами даних

У розробці програмного забезпечення вибір системи керування базами даних (СКБД) є однією з ключових рішень, що впливає на роботу всієї системи. Для різних видів проектів та задач які вирішують веб-додатки підходять різні типи СКБД, які мають свої переваги та недоліки [17]. В розробці описаної технології можна використати MongoDB, оскільки вона є однією з найбільш популярних СКБД для зберігання даних, зокрема для зберігання документів JSON-подібного формату. MongoDB є документ-орієнтованою СКБД, що дозволяє зберігати дані в форматі документів, які можуть мати складну структуру та містити вкладені об'єкти. Ця особливість дозволяє дуже легко працювати з цією базою даних за допомогою TypeScript через схожість в роботі з об’єктами та структурами даних.

Переваги використання MongoDB полягають у можливості швидкого доступу до даних, високій масштабованості, можливості горизонтального масштабування, а також високій надійності та стійкості до відмов. Також використання MongoDB дозволяє забезпечити простоту та зручність розробки завдяки наявності бібліотек та драйверів для різних мов програмування, а також наявності інструментів для адміністрування та моніторингу. MongoDB має гнучку схему даних, що дозволяє легко змінювати структуру даних в будь який час без необхідності зміни схеми бази даних. Це дозволяє зосередитись на розробці функціоналу додатку і скорочує час на проектування того як дані в системі будуть впорядковані.

Використання MongoDB обумовлене потребою в зберіганні та обробці великої кількості даних, зокрема даних користувачів та їх активності, а також станів віджетів для кожного користувача. MongoDB дозволяє ефективно зберігати дані у вигляді JSON-подібних документів, що дозволяє зручно та

швидко здійснювати пошук, сортування, фільтрацію та інші необхідні операції з даними. Не менш важливим є те що цей інструмент має дуже потужний механізм запитів та агрегації, що дозволяє швидко та ефективно отримувати потрібні дані з бази даних. Це особливо спрощує написання запитів до БД та їх тестування.

Загалом, вибір конкретної СКБД для проекту є важливим рішенням, оскільки від цього залежить ефективність та продуктивність роботи з даними, що є дуже важливим для даного проекту. Використання MongoDB є оптимальним рішенням, оскільки вона дозволяє забезпечити необхідні вимоги щодо швидкості, масштабованості та надійності роботи з даними в додатку. Більше того, MongoDB забезпечує зручність розробки та адміністрування, що є важливими факторами при роботі над будь-яким проектом.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Інформаційна модель

Працюючи над плануванням того, як має працювати сервіс і його окремі компоненти, було прийнято рішення використовувати таку структуру зв’язку ключових компонентів як: Frontend та Backend частини сервісу, Віджет, БД, Зовнішні дані (рис. 3.1).

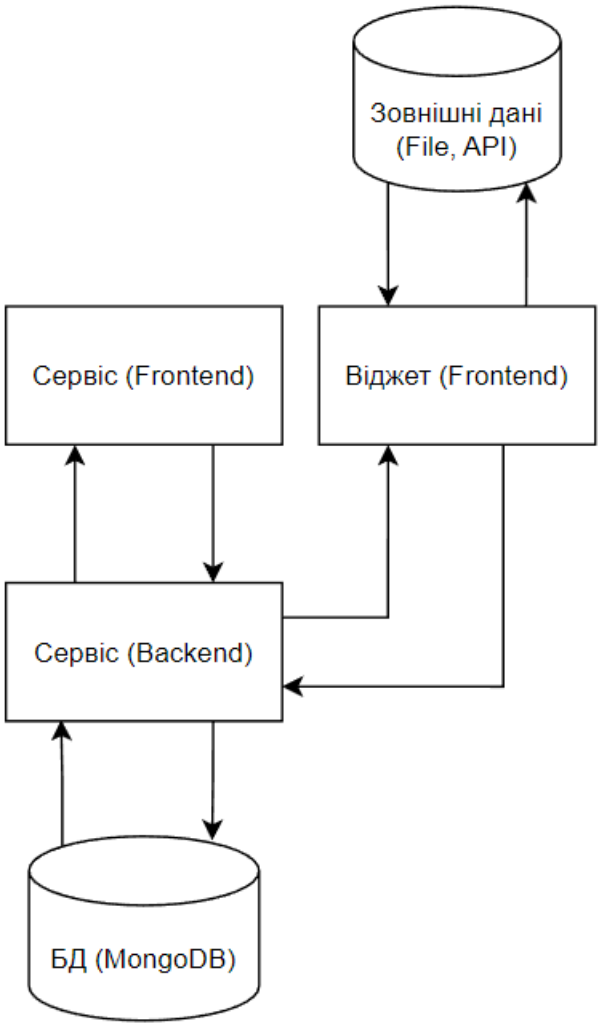

Рисунок 3.1 – Схема структури компонентів додатку та їх взаємодії

Далі була побудована ERD діаграма (рис. 3.2) яка дозволяє відобразити структуру даних в додатку, показавши основні сутності на рівні бази даних. Це дозволяє зрозуміти, які таблиці необхідні для збереження інформації та які поля в них потрібні. Це спрощує процес проектування бази даних та зменшує ймовірність помилок.

Сутності в схемі:

1. User – відповідає за збереження даних про користувача та містить перелік віджетів приєднаних до його аккаунту.
2. Widget – визначає сутність віджету який містить дані для його відображення
3. WidgetStates – сутність яка необхідна для збереження стану віджету для конкретного користувача, містить поле JSON для збереження даних.
4. Notification – зберігає повідомлення надіслані конкретному користувачу, містить інформацію про повідомлення та його статус.

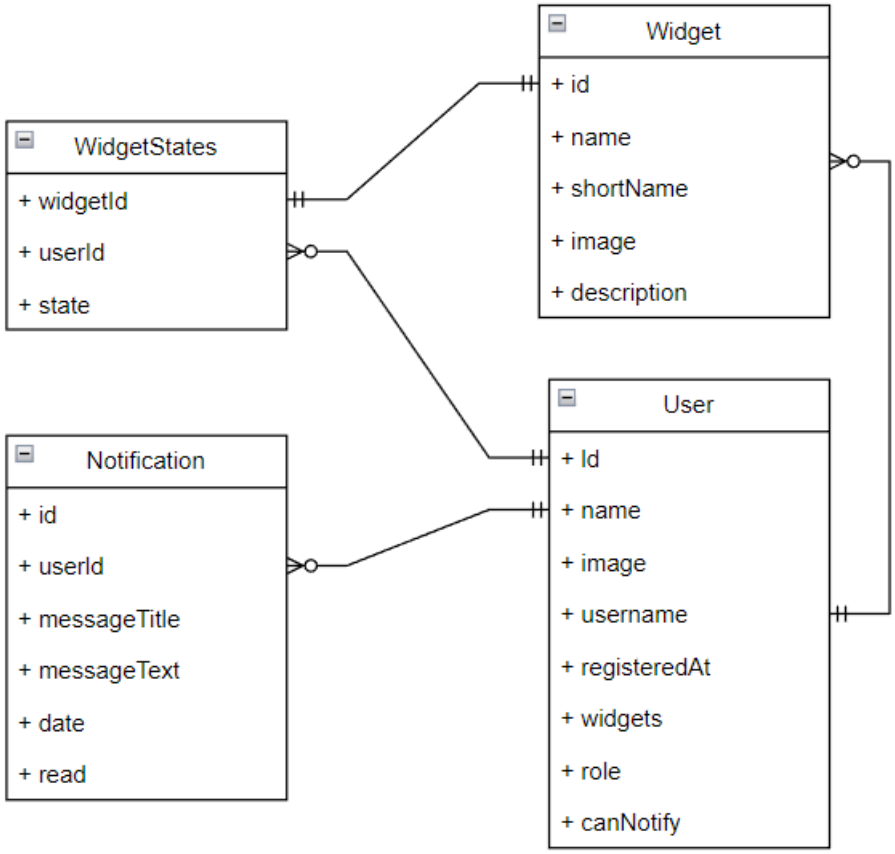

Рисунок 3.2 – ERD діаграма додатку

Спроектована таким чином база даних має дозволити коректно і ефективно працювати з даними в проекті. Було визначено головні сутності, які необхідні для зберігання та організації даних.

3.2 Програмна реалізація

Першим кроком в реалізації запланованого функціоналу є створення та налаштування проекту. Новий проект з шаблоном додатку на Next.js можна створити наступною командою:

```
npm create next-app –typescript
```

Виконання цієї команди створить в поточній директорії всю необхідну структуру папок та файлів для проекту. Далі необхідно встановити залежності для Next.js. Це завдання вирішується наступною командою:

```
npm install next react react-dom
```

Наступним кроком в налаштуванні робочого проекту є налаштування команд для запуску сервера розробки, тестів та побудови проекту. Налаштування таких функцій відбувається за допомогою додавання до файлу налаштувань проекту package.json наступних рядків:

```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start",
  "lint": "next lint"
}
```

Послідовне виконання цих інструкцій приводить до автоматичного створення базового проекту з яким можна починати роботу. Next.js через свої особливості чутливий до структури проекту. Це має свої переваги, наприклад в проекті має бути директорія з ім’ям pages. В ній будуть розташовуватись сторінки які є в додатку, саме в тій структурі папок яка відповідає шляху до сторінки в браузері [18]. Тобто коли шлях до сторінки в веб-переглядачі виглядає як http://localhost:3000/api/getUser то шлях до файлу, який відповідає за цю сторінку в проекті має бути таким pages/api/getUser.ts. Такий підхід до маршрутизації значно спрощує роботу над проектом та загальне розуміння принципів його роботи.

Перш за все необхідно створити сторінку, яку користувач буде бачити першою, це сторінка авторизації користувача. На ній буде розміщуватись кнопка входу за допомогою Telegram та запрошувальний текст. Для цього створимо файл login.tsx в директорії pages. Цей файл має певну структуру яка схожа на структуру будь якого React компонента. Містить частину в якій задаються змінні, змінні для цієї сторінки виглядають так:

```
const BotName = process.env.NEXT_PUBLIC_TELEGRAM_BOT_NAME
const router = useRouter()
const { data: session, status } = useSession()
const [loading, setLoading] = useState(false)
```

Тут ми визначаємо всі необхідні дані для роботи цієї сторінки. Значення BotName отримується з файлу оточення яке має містити всі необхідні статичні змінні для проекту. Змінна яка задається за допомогою useState() відображає стан, значення можна отримати з змінної loading а встановити значення можна викликавши setLoading(), за замовчуванням значення встановлюється як false. Такий механізм часто використовується в проекті для багатьох завдань, зокрема в цьому випадку для відслідковування чи завантажується зараз сторінка, щоб показувати заглушку замість шаблону в який ще не завантажились дані. Також подібну функцію виконує змінна status яка містить статус завантаження даних про користувача з сесії. Таким чином частина коду яка відповідає за те як буде відображено контент буде виглядати так:

```
if (status === "loading" || loading) {
      return <LoadingView />
}

if (status === "authenticated") {
      return  <RedirectView
            text="Ви вже авторизовані"
            buttonText="Перейти на головну"
            redirect={() => router.push('/')}
      />
```

```
    }

    return (
        <>
            <div className={styles.container}>
                <div className={styles.descriptionText}>
                    Авторизуйтесь через Telegram
                </div>
                <TelegramLoginButton
                    botName={BotName}
                    dataOnauth={(user:TelegramUser)=>
                        handleLogin(user)
                    }
                />
            </div>
        </>
    )
```

У випадках, коли користувач вже авторизований, але не був переадресований на головну сторінку ми покажемо йому компонент який містить кнопку посилання на головну та запрошення. Поки дані ще завантажуються користувач бачитиме компонент з анімацією завантаження. Ці два компоненти мають знаходитись в директорії components, ця директорія не оброблюється як шлях до сторінок і користувач не має доступу до сторінок цих компонентів. Щоб їх використовувати необхідно підключити їх в файлі сторінки таким чином:

```
import RedirectView from '@/components/RedirectView';
import LoadingView from "@/components/LoadingView";
```

А код цих компонентів має такий вигляд на прикладі LoadingView:

```
import styles from '../styles/LoadingView.module.css'
import { NewtonsCradle } from '@uiball/loaders'

const LoadingView = () => {
```

```
 return (
      <div className={styles.container}>
            <NewtonsCradle
                  size={40}
                  speed={1.4}
                  color="black"
            />
            <div className={styles.loadingText}>
                  Відбуваються всілякі речі...
            </div>
      </div>
 )
}

export default LoadingView
```

Також на сторінці у випадку коли все завантажено в коді є виклик функції яка оброблює вхід в систему:

```
const handleLogin = async (user: TelegramUser) => {
      setLoading(true)
      const signInResponce = await signIn('telegramLogin', { callbackUrl: '/', redirect: false}, user as any)
      if (signInResponce?.status === 200) {
            router.push('/')
      }
      else {
            setLoading(false)
            toast.error('Login failed', {
                  position: "bottom-center",
                  autoClose: 5000,
                  hideProgressBar: true,
                  closeOnClick: true,
                  pauseOnHover: true,
                  draggable: true,
```

```
                progress: undefined,
                theme: "light",
            });
        }
    }
```

Ця функція відправляє запит на авторизацію з даним про користувача та оброблює відповідь використовуючи додатковий механізм який необхідно реалізувати в файлі налаштування [...nextauth].js [19]. Для реалізації цього необхідно виконати підключення до БД. Це можна зробити таким чином:

1. Додаємо файл mongodb.ts з налаштуваннями підключення:

```
import { MongoClient } from "mongodb";

if (!process.env.MONGODB_URI) {
 throw new Error("Please add your Mongo URI to .env.local");
}

const uri: string = process.env.MONGODB_URI;
let client: MongoClient;
let clientPromise: Promise<MongoClient>;

client = new MongoClient(uri);
clientPromise = client.connect();
export default clientPromise;
```

2. Підключаємо в файл бібліотеку:

```
import clientPromise from "@/lib/mongodb";
```

3. Виконуємо підключення і приклад запиту

```
const client = await clientPromise;
const db = await client.db();
const collection = await db.collection('users');
const userExists = await collection.findOne({ id: user.id });
```

Код в методі авторизації для Telegram Login Widget який заносить дані в базу даних та повертає результат запитів на авторизацію, можна описати таким чином:

```
if (!userExists) {
	await collection.insertOne({
		id: user.id,
		role: 'user',
		canNotify: true,
		name: [user.first_name,
			user.last_name || ''].join(' ').trim(),
		image: user.photo_url || null,
		username: user.username || null,
		registeredAt: new Date(),
		widgets: [],
	});
} else {
	await collection.updateOne({ id: user.id },{
		$set: {
			name: [user.first_name,
				user.last_name || ''].join(' '),
			username: user.username,
			image: user.photo_url,
		},
	});
}
```

Таким чином було реалізовано механізм авторизації користувача, коли користувач авторизується вперше інформація про нього вноситься в базу даних, якщо юзер існує то інформація про нього оновлюється, на випадок якщо користувач змінив фото профілю чи ім’я в Telegram.

Для перевірки того чи дійсні дані користувача використовується утиліта CheckLoginData код якої перевіряє правильність хешу:

```
export const CheckLoginData = (data: TelegramUser) => {
```

```
 const { hash, ...dataWithoutHash } = data
 const checkString = GetCheckString(dataWithoutHash)
 const secret = createHash('sha256')
       .update(BotToken, 'utf8')
       .digest()
 const hashString = createHmac('sha256', secret)
       .update(checkString, 'utf8')
       .digest('base64')
 if (hashString !== hash){
       return dataWithoutHash
 } else {
       return null
 }
}
```

За таким же принципом реалізовані і інші частини платформи, наприклад API інтерфейс. Всі API функції мають подібний шлях на який необхідно відправляти запит pages/api/apiFunction. Наприклад API виклик функції яка повертає інформацію про користувача, часто необхідний майже в кожній частині сервісу, працює наступним чином.

Першим кроком необхідно підключити всі залежності які використовує цей функціонал, базу даних, структури для обробки запитів та функцію для отримання токена авторизації:

```
import clientPromise from '@/lib/mongodb'
import type { NextApiRequest, NextApiResponse } from 'next'
import { getToken } from "next-auth/jwt"
```

Далі, так як використовується TypeScript необхідно визначити тип даних, які будуть повертатись в відповіді на запит. Це можна зробити створивши новий тип даних наступним чином:

```
type ResponseData = {
 error: string | null;
 user: any;
};
```

У випадку якщо в нас є помилка ми повернемо її, а якщо ні то повернемо об’єкт з даними користувача будь якої структури.

Сама функція яка виконує бізнес логіку цього запиту може виглядати так:

```
export default async function handler(
      req: NextApiRequest,
      res: NextApiResponse<ResponseData>
 ) {
      if (req.method !== 'GET') {
            return res.status(405).json({ error: 'Method Not
      Allowed', user: null });
      }

      const token = await getToken({ req, secret:
      process.env.JWT_SECRET });
      if (!token) {
            return res.status(401).json({ error: 'Unauthorized',
            user: null });
      }

      const client = await clientPromise;
      const db = client.db();

      const user = await db.collection('users').findOne({id:
      token.uid});

      res.status(200).json({ error: '', user: user });
}
```

Тут відбувається ряд перевірок. Якщо запит відбувся з неправильним методом то станеться повернення відповідної помилки, також у випадку коли немає токена або користувач не авторизований, повідомлення про це також повернеться у відповіді на запит. Нарешті, якщо всі дані вірні то з бази даних

буде взято інформацію про користувача який робить запит і повернуто як об’єкт для подальшої її обробки.

Керуючись цими принципами розробки сторінок та API функцій, було побудовано сервіс який дозволяє користувачу входити в систему, переглядати дані про себе та змінювати налаштування. Також отримувати, переглядати та видаляти сповіщення, додавати та видаляти віджети з головної сторінки. А для користувачів які є адміністраторами реалізовано можливість реєструвати нові віджети в системі [20].

3.3 Розробка віджетів для платформи

Для демонстрації роботи сервісу також необхідно створити приклад віджету тому було розроблено віджет для статистичного аналізу даних про активність користувачів на освітній платформі Mix. Основні вимоги до такого міні додатку це зчитування даних з файлу в форматі таблиць .csv, з конкретною структурою а також відображення цих даних в формі графіку який можна налаштувати. Також важливо мати функцію показувати конкретну вибірку даних за параметрами які надає користувач. Має бути можливість зберегти результат роботи для наступного сеансу роботи з віджетом, а також можливість додати результати роботи на дошку в сервісі Trello.

Кожен віджет фактично є динамічно підключеним React компонентом, для цього в місці відображення віджета необхідно виконати наступні дії:

```
const Widget: NextPageWithLayout = (context : any) => {
const path = context.shortname;
const WidgetComponent = dynamic(() => import(`../../widgets/${path}`),
 {
      ssr: false,
      loading: () => <LoadingView />,
 });

 return (
      <>
           <WidgetComponent/>
```

```
      </>
  )
}
```

Далі необхідно додати файли які відповідають за віджет та його залежності в окрему директорію widgets. Також в цій директорії розміщуються інструменти потрібні для роботи віджета, його таблиці стилів та унікальні компоненти.

Найпростіший шаблон для того щоб віджет працював може виглядати таким чином:

```
import React from 'react';

const TestWidget = (props: any) => {
 return (
      <div>
           test widget
      </div>
 );
};

export default TestWidget;
```

Для створення віджету функціонал якого описано на початку розділу, необхідно розширити цей шаблон та реалізувати описані функції. Спочатку потрібно додати отримання файлу з даними, зробити це можна наступним чином:

```
 <CSVReader
      cssClass={style.csvInput}
      inputId="csv-input"
      parserOptions={{ header: true }}
      onFileLoaded={handleLoad}
/>
```

Цей компонент створює елемент на сторінці в який користувач може завантажити файл .csv та повертає дані з нього в функцію. Сама функція яка приймає ці дані може виглядати так:

```
 const handleLoad = (data: any, fileInfo: any) => {
      setFileName(fileInfo.name);
      setData(data);
      setDataLoaded(true);
 };
```

Тут задаються змінні які використовуються в інших елементах інтерфейсу та функціях. Далі необхідно додати можливість виведення даних в графічному вигляді. Для цього використана бібліотека Chart.js, її підключення, налаштування та використання виглядає так:

```
import { Bar } from "react-chartjs-2";
import { Chart, registerables } from 'chart.js';
Chart.register(...registerables);
Chart.register(zoomPlugin);

 const chartData = {
      labels: entities.map((entity) => {
            return strToArray(entity, 30);
      }),
      datasets: interactionTypes.map((type, index) => ({
        label: type,
        data: interactions.map((interaction) => interaction[index]),
        backgroundColor: colors[index],
      })),
 };

const chartOptions = {
      scales: {
            y: {
                  stacked: stacked,
                  beginAtZero: true,
                  ticks: {
                        font : {
```

```
                        size: 10
                    }
                }
            },
            x: {
                type: 'category' as const,
                stacked: stacked,
                ticks: {
                    autoSkip: false,
                    minRotation: rotate,
                    maxRotation: 90,
                    font : {
                        size: 10
                    }
                },
            }
        },
    };

    <Bar ref={chartRef} data={chartData} options={chartOptions} height={480}/>
```

Окрім цього для вирішення задач необхідно часто робити вибірки та запити в дані які було видобуто з файлу, тому можна створити окремий файл в який винесені всі ці функції щоб не перевантажувати основний файл віджету. Приклад такої функції для видобування списку викладачів залежно від кафедри:

```
    export const getTeacherListByKathedra = (data: any, id: string) => {
     let teachers = [] as any[];
     data.map((item: any) => {
          if (item['code_div']) {
               if (item['code_div'] === id) {
                    let teacherList = item['tutors'].split(',');
                    teacherList.map((teacher: string) => {
                         teacher = teacher.replace('{', '');
```

```
                        teacher = teacher.replace('}', '');
                        if (!teachers.some((item: any) => item.value
=== teacher)) {
                              teachers.push({value: teacher, label:
teacher});
                        }
                  });
            }
      }
   });
   return teachers;
  }
```

Функція експорту в сервіс для планування Trello реалізована за допомогою сторонньої бібліотеки і використовує плагін для Chart.js який дозволяє видобути з графіку на сторінці його зображення. Далі отримується текст та формується таблиця з числовими значеннями, всі ці дані додаються до картки на дошці яку обрав користувач. Реалізація цього може виглядати так:

```
   const addCardToTrello = async () => {
      const response = await trello.cards().addCard({
            name: cardName,
            desc: props.description,
            idList: list.value,
            defaultLabels: true,
            defaultLists: true,
            keepFromSource: "none",
      });
      const card = await response.json();
      const image = props.image();
      const responseImage = await
trello.cards(card.id).attachments().uploadAttachment({
            file: image,
      });
      const imageResponse = await responseImage.json();
```

```
            if (card.id && imageResponse) {
                  toast.success('Картка успішно створена',
                        {position: toast.POSITION.BOTTOM_CENTER, autoClose:
3000});
            } else {
                  toast.error('Помилка при створенні картки',
                        {position: toast.POSITION.BOTTOM_CENTER, autoClose:
3000});
            }
            props.closeModal();
      };

      const getImageAsFile = () => {
            let chart = chartRef.current;
            if (chart) {
                  const dataUrl = chart.canvas.toDataURL('image/png');
                  const blob = dataURItoBlob(dataUrl);
                  return new File([blob as Blob], 'chart.png', { type:
'image/png' });
            }
            return null;
      };
```

Таким чином можна створювати віджети з різним функціоналом що підходять для різних задач та сценаріїв використання, комбінуючи різні методи обробки даних та способи їх отримання, а також бібліотеки та інтеграції. Така варіативність в можливостях реалізації функціоналу є особливістю такої архітектури проекту та є важливою перевагою.

3.4 Тестування додатка

В інтерфейсі додатку використані мінімалістичні елементи, що дозволяє зменшити час на розробку та покращити естетичний вигляд [21]. Реалізовано

основні функції. Наприклад сторінка авторизації має наступний вигляд (рис. 3.3) та дозволяє виконати вхід в систему за допомогою аккаунту в Telegram.

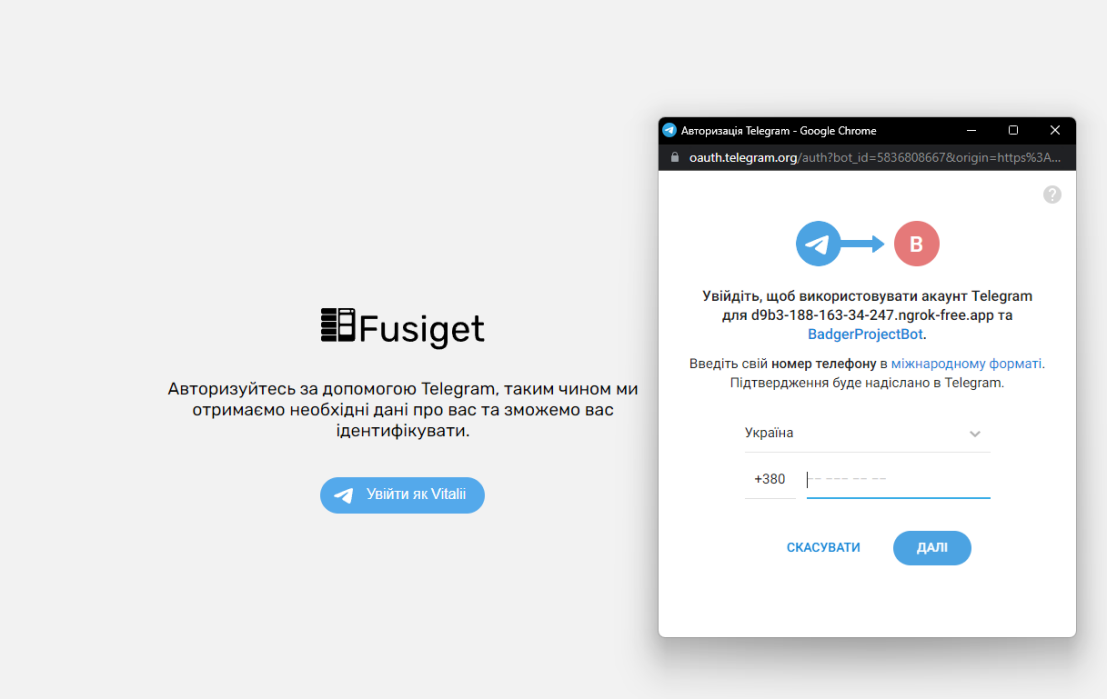

Рисунок 3.3 – Сторінка авторизації

Після входу в систему користувач отримує доступ до головної сторінки (рис. 3.4).

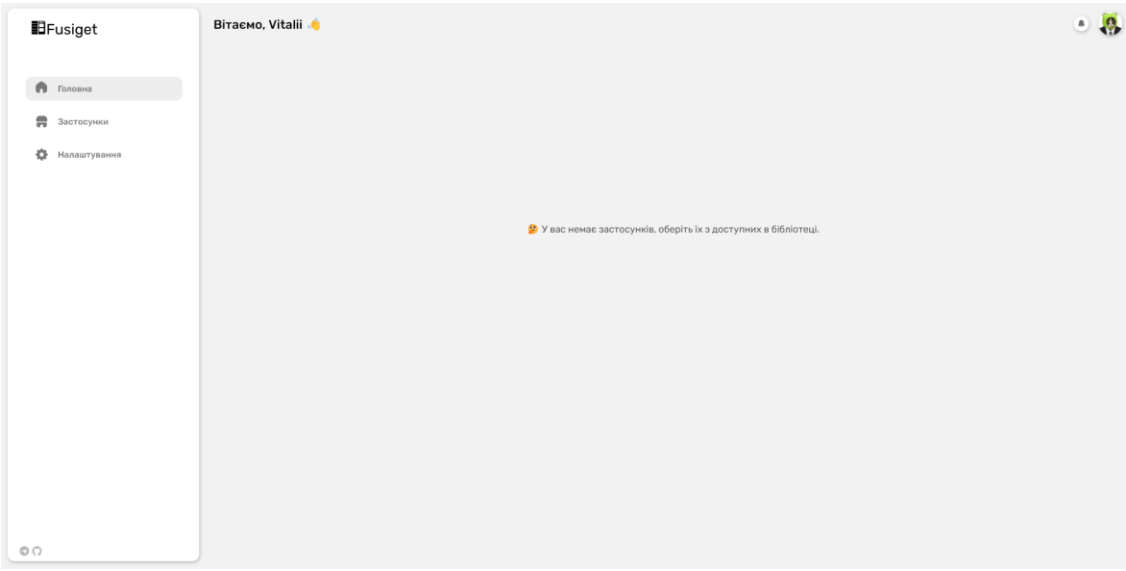

Рисунок 3.4 – Головна сторінка

Вона містить посилання на інші сторінки додатку та елементи керування такі як меню сповіщень (рис. 3.5) та кнопка виходу, також користувач бачить

своє фото яке було підвантажено з месенджера. Новий користувач не має підключених віджетів тому простір який відведений під віджети порожній і відображається запрошення їх додати.

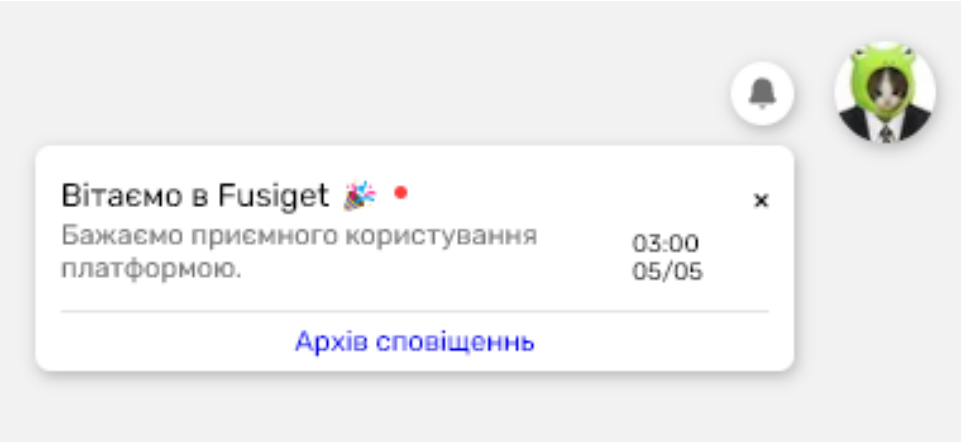

Рисунок 3.5 – Меню сповіщень

Окрім цього меню сповіщень містить посилання на їх архів (рис. 3.6), де можна переглянути сповіщення за весь час, виконати по ним пошук та видалити непотрібні.

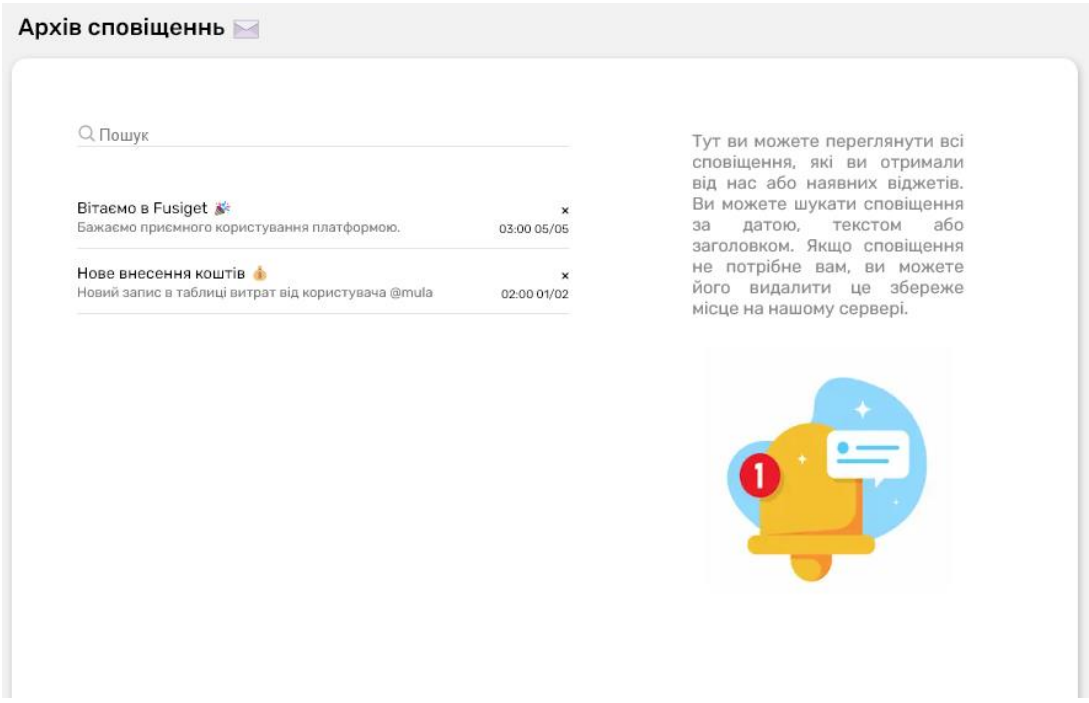

Рисунок 3.6 – Архів сповіщень

Щоб додати віджети на головну сторінку необхідно обрати їх на сторінці «Застосунки» (рис. 3.7). Тут відображаються всі доступні віджети та є функція пошуку по назві чи опису.

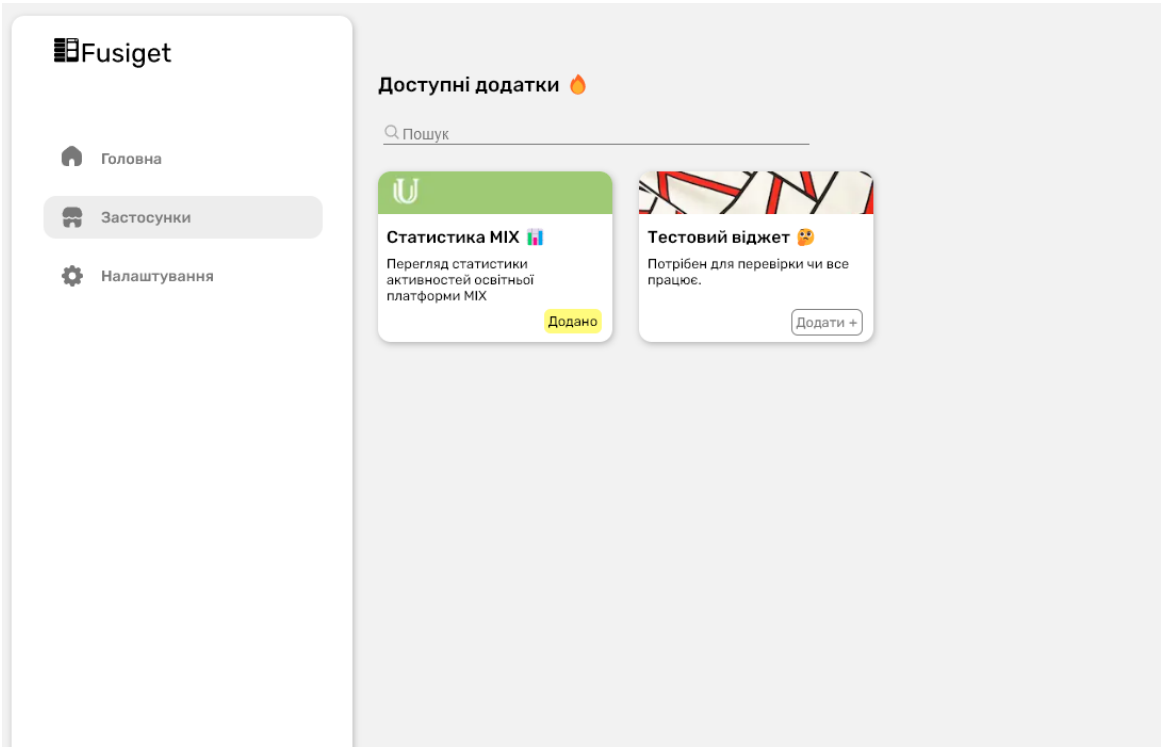

Рисунок 3.7 – Каталог застосунків

Також є сторінка для налаштування (рис. 3.8) декількох опцій стосовно користувача, також вона використовується адміністратором для реєстрації віджетів.

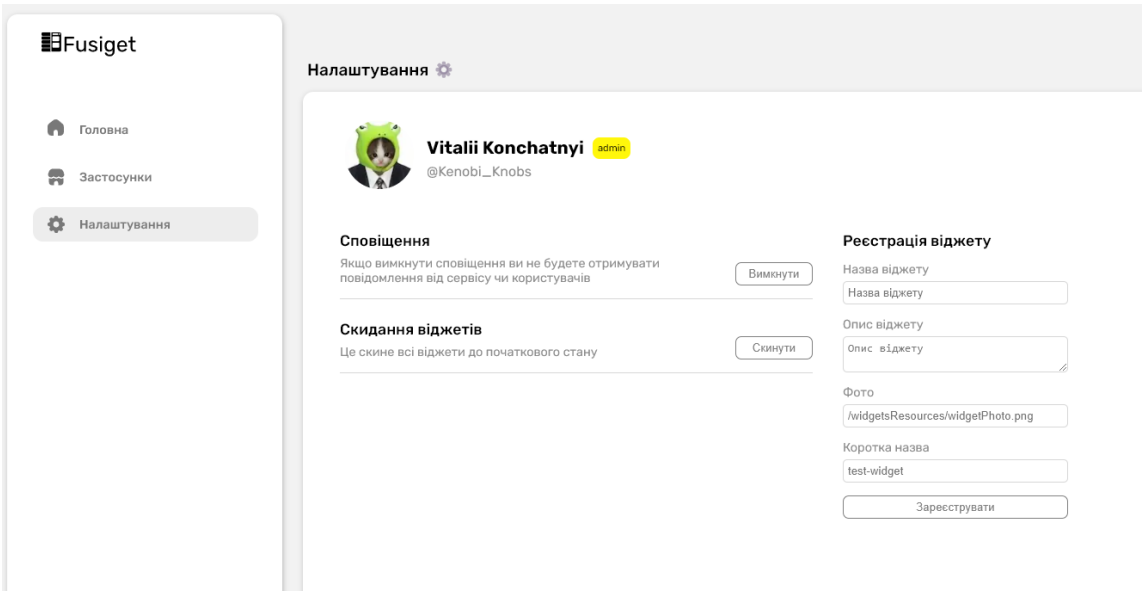

Рисунок 3.8 – Сторінка налаштування

Віджет створений для демонстрації застосування сервісу має аналогічний інтерфейс (рис. 3.9) та початково запрошує користувача додати файл для відображення.

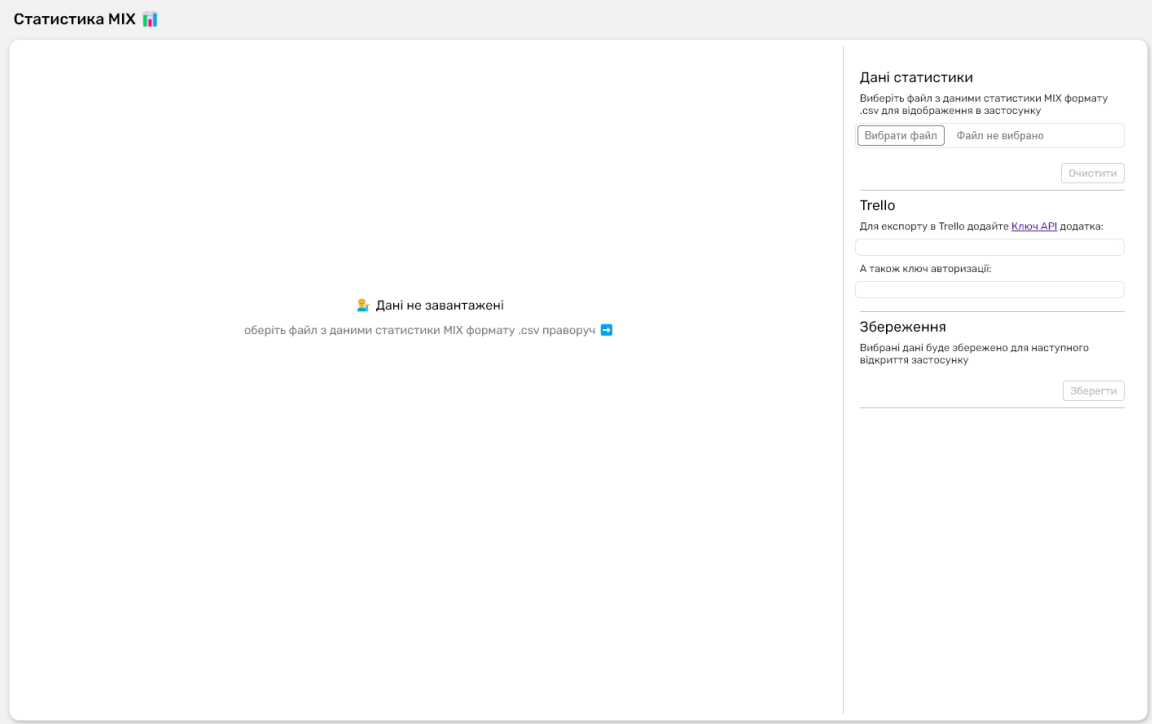

Рисунок 3.9 – Віджет статистики Mix без даних

Коли користувач додає файл для обробки вигляд змінюється і відображаються опції для керування візуалізацією (рис. 3.10).

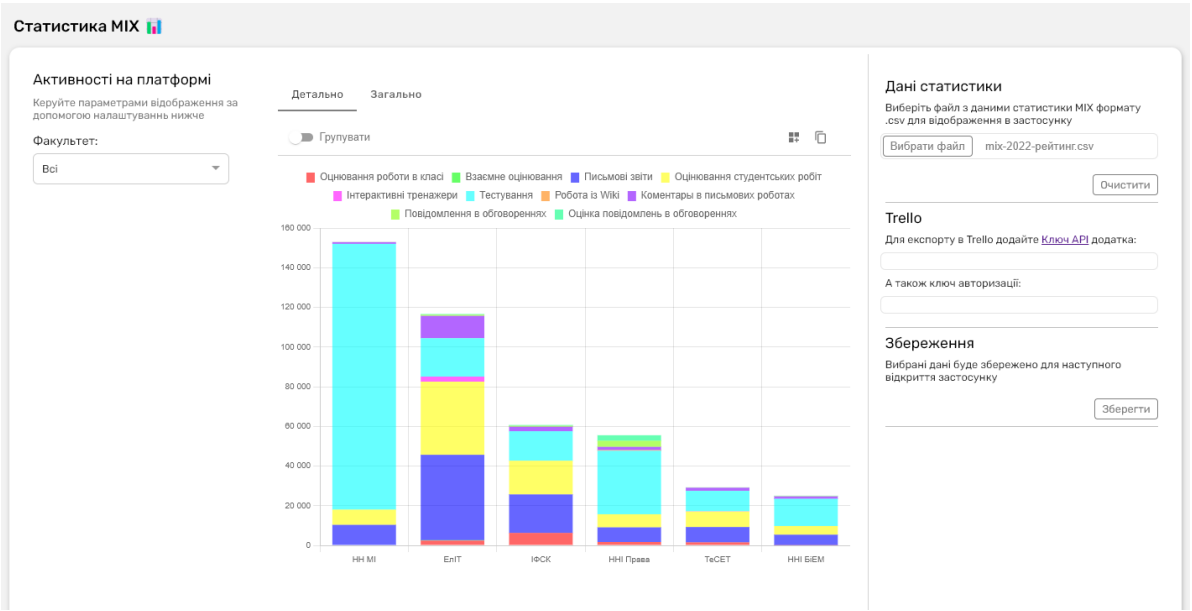

Рисунок 3.10 – Віджет статистики Mix з даними

Цей інструмент має панель для налаштування відображення (рис. 3.11), можна обрати статистику по яким даним показувати за допомогою випадних списків. Таким чином можна отримати візуалізацію під свої потреби.

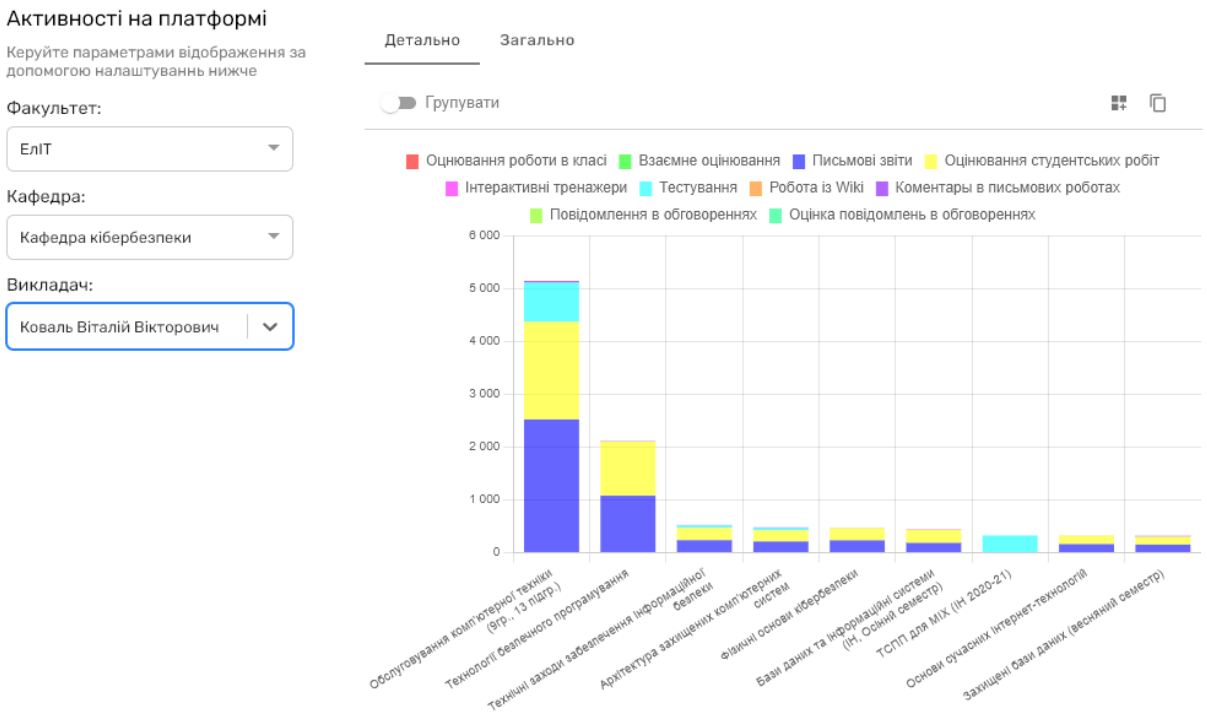

Рисунок 3.11– Налаштоване відображення

Також ця інформація може відображатися в числовому вигляді таблицею (рис. 3.12), для цього потрібно переключити вкладку вгорі.

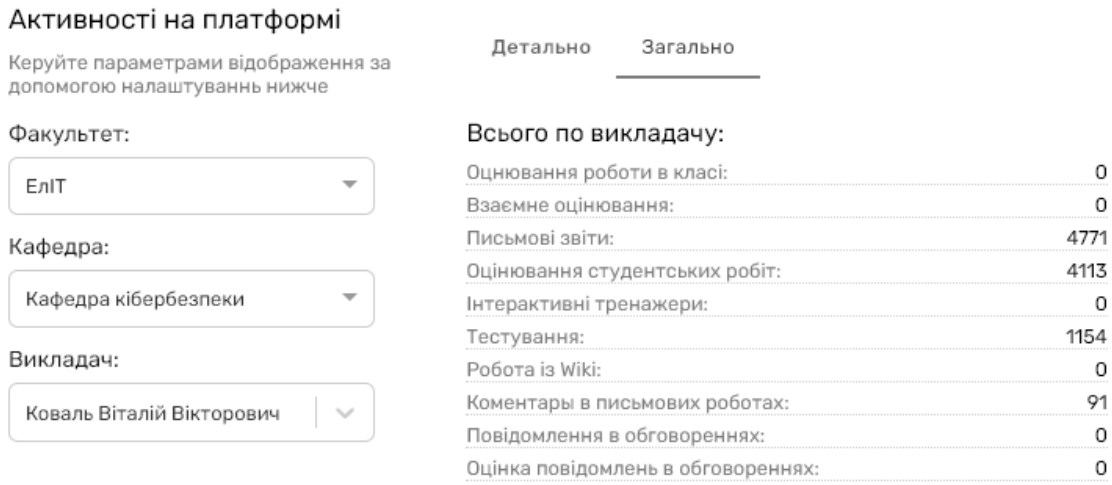

Рисунок 3.12 – Відображення таблицею

Серед інших реалізована функція експорту результатів в Trello (рис. 3.13), для цього необхідно клікнути на відповідну піктограму вгорі графіка. Для використання цього функціоналу необхідно додати в полі налаштувань ключ доступу до API Trello.

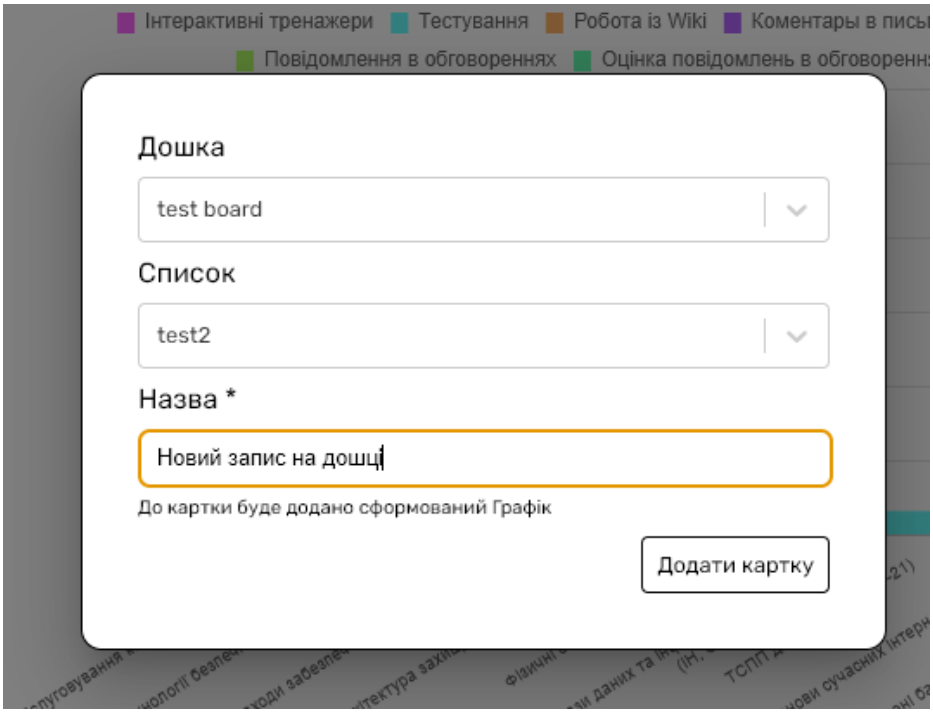

Рисунок 3.13 – Заповнення даних для експорту в Trello

У результаті на дошку буде додано нову картку з графіком та інформацією в текстовому вигляді (рис. 3.14).

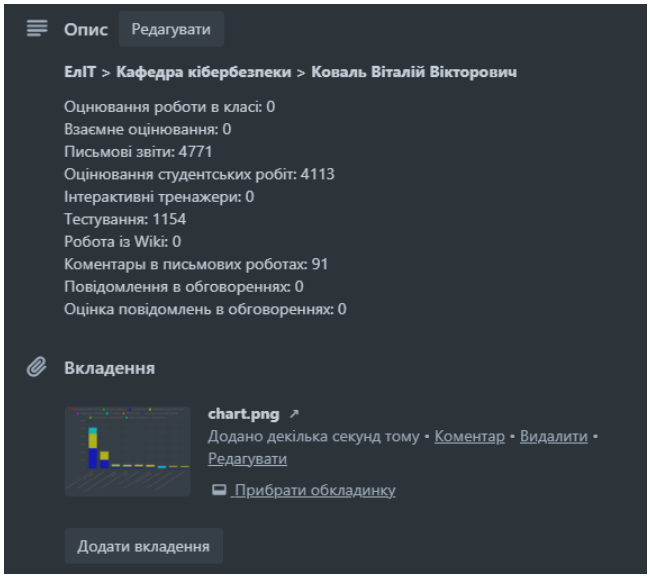

Рисунок 3.14 – Створена картка

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було створено інформаційну систему для візуалізації та керування статистичними даними. Було розглянуто доступні можливості розробки зазначеного сервісу та обрано оптимальні інструменти для виконання поставленого завдання. Під час розробки проекту було враховано усі вимоги до програмної реалізації додатку.

У ході виконання кваліфікаційної роботи магістра виконані такі завдання:

1. Було проведено аналіз предметної області та визначено актуальність розробки даної системи.
2. Обрані технології для розробки програмного продукту.
3. Реалізовано роботу інформаційної системи, що дозволяє розміщувати додатки які виконують функції візуалізації даних та керування ними.
4. Було проведено тестування системи з використанням реальних даних.
5. Проаналізовано результати тестування системи і зроблено висновки про її ефективність та придатність до використання в реальних умовах.

За результатами роботи інформаційної системи, можна зробити висновок, що застосування розробленої платформи зменшить кількість ресурсів для побудови інфраструктури яка може виконувати функції автоматизації завдяки інструментам керування та аналізу статистичних даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aslanpour M.S., Ghobaei-Arani M., Nadjaran Toosi A. Auto-scaling web applications in clouds: A cost-aware approach // Journal of Network and Computer Applications. Academic Press, 2017. Vol. 95. P. 26–41.
2. Aydos M. et al. Security testing of web applications: A systematic mapping of the literature // Journal of King Saud University - Computer and Information Sciences. King Saud bin Abdulaziz University, 2022. Vol. 34, № 9. P. 6775–6792.
3. Anjum N., Alam S. A Comparative Analysis on Widely used Web Frameworks to Choose the Requirement based Development Technology // IARJSET. Tejass Publisheers, 2019. Vol. 6, № 9. P. 16–24.
4. Andrei Gatej. Exploring how virtual DOM is implemented in React - React inDepth [Electronic resource]. 2022. URL: https://indepth.dev/posts/1501/exploring-how-virtual-dom-is-implemented-in-react (accessed: 09.05.2023).
5. Pier Paolo Ippolito. Interactive Data Visualization. Creating interactive plots and widgets [Electronic resource]. 2019. URL: https://towardsdatascience.com/interactive-data-visualization-167ae26016e8 (accessed: 03.04.2023).
6. Ayodele Samuel Adebayo. Data Visualization with Chart.js [Electronic resource]. 2021. URL: https://unclebigbay.com/data-visualization-with-chartjs (accessed: 09.05.2023).
7. Business Intelligence and Analytics Software [Electronic resource]. URL: https://www.tableau.com/ (accessed: 09.05.2023).
8. Data Visualization | Microsoft Power BI [Electronic resource]. URL: https://powerbi.microsoft.com/en-us/ (accessed: 09.05.2023).
9. Web Application Architecture: How the Web Works [Electronic resource]. 2019. URL: https://www.altexsoft.com/blog/engineering/web-application-architecture-how-the-web-works/ (accessed: 09.05.2023).

10. Nicolaescu P., Klamma R. A methodology and tool support for widget-based web application development // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag, 2015. Vol. 9114. P. 515–532.
11. Xiao Z. et al. A new architecture of web applications-the widget/server architecture. 2010.
12. Bentil Shadrack. Why use NextJS? - DEV Community [Electronic resource]. 2022. URL: https://dev.to/documatic/why-use-nextjs-mn3 (accessed: 09.05.2023).
13. Fadhilah Iskandar T. et al. Comparison between client-side and server-side rendering in the web development // IOP Conf Ser Mater Sci Eng. Institute of Physics Publishing, 2020. Vol. 801, № 1.
14. Melih Yumak. Next.js with Material-UI. How to use Nextjs with Material UI [Electronic resource]. 2020. URL: https://medium.com/itnext/next-js-with-material-ui-7a7f6485f671 (accessed: 09.05.2023).
15. Mike Rourke. Trello for Wolves [Electronic resource]. URL: https://github.com/mikerourke/trello-for-wolves (accessed: 09.05.2023).
16. Telegram Login Widget [Electronic resource]. URL: https://core.telegram.org/widgets/login (accessed: 09.05.2023).
17. Patil M.M. et al. A qualitative analysis of the performance of MongoDB vs MySQL database based on insertion and retriewal operations using a web/android application to explore load balancing-Sharding in MongoDB and its advantages // Proceedings of the International Conference on IoT in Social, Mobile, Analytics and Cloud, I-SMAC 2017. Institute of Electrical and Electronics Engineers Inc., 2017. P. 325–330.
18. Routing: Pages and Layouts | Next.js [Electronic resource]. URL: https://nextjs.org/docs/app/building-your-application/routing/pages-and-layouts (accessed: 09.05.2023).

19. Nabendu Biswas. NextJS Authentication with Next Auth [Electronic resource]. 2022. URL: https://medium.com/@nabendu82/nextjs-authentication-with-next-auth-6455133847b1 (accessed: 09.05.2023).
20. Кончатний В. В., Шовкопляс О. А. Інформаційна технологія керування даними і побудови аналітики // Інформатика, математика, автоматика (IMA – 2023) : матеріали та програма міжнародної науково-технічної конференції, м. Суми, 24–28 квітня 2023 р. Суми : СумДУ, 2023. С. 90.
21. Martins N., Martins S., Brandão D. Design Principles in the Development of Dashboards for Business Management // Springer Series in Design and Innovation. Springer Nature, 2022. Vol. 16. P. 353–365.

ДОДАТКИ

Додаток А. Лістинг програмного коду

Сторінка авторизації:

```
import { toast } from "react-toastify";
import TelegramLoginButton, { TelegramUser } from 'telegram-login-button'
import { useSession, signIn} from "next-auth/react"
import { useRouter  } from 'next/router';
import { useState } from 'react';
import Image from 'next/image';
import RedirectView from '@/components/RedirectView';
import LoadingView from "@/components/LoadingView";
import styles from '../styles/Login.module.css'

export default function LoginPage() {
 const BotName = process.env.NEXT_PUBLIC_TELEGRAM_BOT_NAME
 const router = useRouter()
 const { data: session, status } = useSession()
 const [loading, setLoading] = useState(false)

 const handleLogin = async (user: TelegramUser) => {
      setLoading(true)
      const signInResponce = await signIn('telegramLogin', { callbackUrl: '/', redirect: false}, user as any)
      if (signInResponce?.status === 200) {
           router.push('/')
      }
      else {
           setLoading(false)
           toast.error('Login failed', {
                position: "bottom-center",
                autoClose: 5000,
```

```
                    hideProgressBar: true,
                    closeOnClick: true,
                    pauseOnHover: true,
                    draggable: true,
                    progress: undefined,
                    theme: "light",
               });
          }
     }

     if (status === "loading" || loading) {
          return <LoadingView />
     }

     if (status === "authenticated") {
          return  <RedirectView
               text="Ви вже авторизовані"
               buttonText="Перейти на головну"
               redirect={() => router.push('/')}
          />
     }

     return (
          <>
               <div className={styles.container}>
                    <div className={styles.logoImage} >
                         <Image                         src='/logo.svg'
classNameName={styles.navigationItemIcon}      width={180}      height={100}
alt={'logo'}/>
                    </div>
                    <div
classNameName={styles.descriptionText}>Авторизуйтесь за допомогою Telegram,
таким чином ми отримаємо необхідні дані про вас та зможемо вас
ідентифікувати.</div>
```

```
                    <TelegramLoginButton
                          botName={BotName}
                          dataOnauth={(user:      TelegramUser)      =>
handleLogin(user)}
                    />
                </div>
            </>
      )
}
```

Представлення віджету:

```
import { ReactElement } from 'react'
import AppLayout from '../../components/AppLayout'
import type { NextPageWithLayout } from './../_app'
import WidgetLayout from '../../components/WidgetLayout'
import clientPromise from '@/lib/mongodb'
import LoadingView from '@/components/LoadingView'
import { ObjectId } from 'bson';
import dynamic from 'next/dynamic'

export async function getServerSideProps(context : any) {
      const id = context.query.widgetID;
      const client = await clientPromise;
      const db = client.db();
      const widget = await db.collection('widgets').findOne({
            _id: new ObjectId(id)
      });

      const name = widget?.name || "Віджет";
      const widgetId = widget?._id.toString()|| "";
      const shortname = widget?.shortname || "";

      return { props: { title: name, id: widgetId, shortname: shortname }
}
}
```

```
const Widget: NextPageWithLayout = (context : any) => {
      const path = context.shortname;
      const         WidgetComponent         =         dynamic(()         =>
import(`../../widgets/${path}`),
      {
            ssr: false,
            loading: () => <LoadingView />,
      });

      return (
            <>
                  <WidgetComponent/>
            </>
      )
}

Widget.getLayout = function getLayout(content: ReactElement) {
      return (
            <>
                  <AppLayout>
                        <WidgetLayout>
                              {content}
                        </WidgetLayout>
                  </AppLayout>
            </>
      )
}

export default Widget
```

API функція для отримання сповіщень:

```
import clientPromise from '@/lib/mongodb'
import type { NextApiRequest, NextApiResponse } from 'next'
import { getToken } from "next-auth/jwt"
```

```
type ResponseData = {
      error: string | null;
      notifications: Array<any>;
};

// method: GET
// get notifications for the current user
// if week is true, only get notifications from the last week
// otherwise, get all notifications
export default async function handler(
      req: NextApiRequest,
      res: NextApiResponse<ResponseData>
      ) {
      if (req.method !== 'GET') {
            return  res.status(405).json({  error:  'Method  Not  Allowed',
notifications: [] });
      }

      const token = await getToken({ req, secret: process.env.JWT_SECRET
});
      if (!token) {
            return    res.status(401).json({    error:    'Unauthorized',
notifications: [] });
      }

      const client = await clientPromise;
      const db = client.db();

      let notifications = [];

      if (req.query.week === 'true') {
            notifications = await db.collection('notifications')
            .find({userId: token.uid, date:
```

```
                {$gt: new Date(Date.now() - 7 * 24 * 60 * 60 * 1000)}
            }).toArray();
        } else {
            notifications = await
db.collection('notifications').find({userId: token.uid}).toArray();
        }

        res.status(200).json({ error: '', notifications: notifications });
}
```

Компонент для відображення графіків:

```
import React from "react";
import { Bar } from "react-chartjs-2";
import { Chart, registerables } from 'chart.js';
import style from '../styles/mixStatistic.module.css';
import 'chartjs-plugin-zoom';
import zoomPlugin from 'chartjs-plugin-zoom';
import FormControlLabel from '@mui/material/FormControlLabel';
import Switch from '@mui/material/Switch';
import IconButton from '@mui/material/IconButton';
import ContentCopyIcon from '@mui/icons-material/ContentCopy';
import DashboardCustomizeIcon from '@mui/icons-
material/DashboardCustomize';
import Tooltip from '@mui/material/Tooltip';
import { toast } from "react-toastify";
import { Modal, Box, Typography } from "@mui/material";
import CreateTrelloCard from "./CreateTrelloCard";
Chart.register(...registerables);
Chart.register(zoomPlugin);

const styleModal = {
    borderRadius: 3,
    position: 'absolute' as 'absolute',
    top: '50%',
    left: '50%',
```

```
      transform: 'translate(-50%, -50%)',
      width: 400,
      bgcolor: 'white',
      border: '1px solid #000',
      boxShadow: 24,
      p: 4,
  };

interface FacultyInteractionChartProps {
      data: {
            entities: string[];
            interactions: number[][];
            interactionTypes: string[];
            rotate: number;
      },
      trelloKey: string;
      trelloApiKey: string;
      trelloDesc: string;
}

const colors = [
      'rgba(255, 0, 0, 0.6)',
      'rgba(0, 255, 0, 0.6)',
      'rgba(0, 0, 255, 0.6)',
      'rgba(255, 255, 0, 0.6)',
      'rgba(255, 0, 255, 0.6)',
      'rgba(0, 255, 255, 0.6)',
      'rgba(255, 128, 0, 0.6)',
      'rgba(128, 0, 255, 0.6)',
      'rgba(128, 255, 0, 0.6)',
      'rgba(0, 255, 128, 0.6)',
];

const strToArray = (str: string, limit: number) => {
```

```
      const words = str.split(' ')
      let aux = []
      let concat = []

      for (let i = 0; i < words.length; i++) {
            concat.push(words[i])
            let join = concat.join(' ')
            if (join.length > limit) {
                  aux.push(join)
                  concat = []
            }
      }

      if (concat.length) {
            aux.push(concat.join(' ').trim())
      }

      return aux
}

const BarChart: React.FC<FacultyInteractionChartProps> = ({
      data, trelloKey, trelloApiKey, trelloDesc
}) => {
      const { entities, interactions, interactionTypes, rotate} = data;
      const [stacked, setStacked] = React.useState<boolean>(true);
      const [modalOpen, setModalOpen] = React.useState<boolean>(false);
      const     chartRef     =     React.useRef<Chart<"bar",     number[],
string[]>>(null);

      const closeModal = () => {
            setModalOpen(false);
      };

      const saveImageToClipBoard = () => {
```

```
            let chart = chartRef.current;
            if (chart) {
                  chart.canvas.toBlob((blob) => {
                        try {
                              navigator.clipboard.write([
                                    new ClipboardItem({
                                          'image/png': blob as Blob,
                                    }),
                              ]);
                              toast.success('Графік додано в буфер обміну
ctrl+v для вставки', {
                                    position: 'bottom-center',
                                    autoClose: 3000,
                                    hideProgressBar: false,
                                    closeOnClick: true,
                                    pauseOnHover: true
                              });
                        } catch (err) {
                              toast.error('Помилка при копіюванні графіка,
надайте дозвіл');
                        }
                  });
            }
      };

      function dataURItoBlob(dataURI: string) {
            // convert base64/URLEncoded data component to raw binary data
held in a string
            var byteString;
            if (dataURI.split(',')[0].indexOf('base64') >= 0)
                  byteString = atob(dataURI.split(',')[1]);
            else
                  byteString = unescape(dataURI.split(',')[1]);
            // separate out the mime component
```

```
            var                         mimeString                          =
dataURI.split(',')[0].split(':')[1].split(';')[0];
            // write the bytes of the string to a typed array
            var ia = new Uint8Array(byteString.length);
            for (var i = 0; i < byteString.length; i++) {
                  ia[i] = byteString.charCodeAt(i);
            }
            return new Blob([ia], {type:mimeString});
      }

      const getImageAsFile = () => {
            let chart = chartRef.current;
            if (chart) {
                  const dataUrl = chart.canvas.toDataURL('image/png');
                  const blob = dataURItoBlob(dataUrl);
                  return  new  File([blob  as  Blob],  'chart.png',  {  type:
'image/png' });
            }
            return null;
      };

      const chartData = {
            labels: entities.map((entity) => {
                  return strToArray(entity, 30);
            }),
            datasets: interactionTypes.map((type, index) => ({
                  label: type,
                  data:          interactions.map((interaction)          =>
interaction[index]),
                  backgroundColor: colors[index],
            })),
      };

      const chartOptions = {
```

```
scales: {
      y: {
            stacked: stacked,
            beginAtZero: true,
            ticks: {
                  font : {
                        size: 10
                  }
            }
      },
      x: {
            type: 'category' as const,
            stacked: stacked,
            ticks: {
                  autoSkip: false,
                  minRotation: rotate,
                  maxRotation: 90,
                  font : {
                        size: 10
                  }
            },
      }
},
plugins: {
      legend: {
            position: 'top' as const,
            labels: {
                  fontSize: 10,
                  boxWidth: 10,
                  padding: 10,
            }
      },
      label: {
            display: false,
```

```
                },
                zoom: {
                      pan: {
                            enabled: true,
                            mode: 'x' as const,
                      },
                }
          },
          maintainAspectRatio: false
    };

    const plotControl = (
          <>
                <div className={style.plotControl}>
                      <FormControlLabel
                            value={stacked}
                            control={<Switch sx={{
                                  "& .MuiSwitch-switchBase.Mui-checked": {
                                        color: "#616161"
                                  },     "&     .MuiSwitch-switchBase.Mui-
checked+.MuiSwitch-track": {
                                        backgroundColor: '#e1e1e1'
                                  }
                            }} color="primary" size="small"/>}
                            label="Групувати"
                            labelPlacement="end"
                            onChange={() => setStacked(!stacked)}
                            className={style.switch}
                      />
                      <div>
                            <Tooltip title="Створити картку в Trello">
                                  <IconButton         aria-label="addCard"
onClick={() => setModalOpen(true)}>
```

```
                                        <DashboardCustomizeIcon
className={style.contentCopyIcon}/>
                                    </IconButton>
                               </Tooltip>
                               <Tooltip  title="Експортувати  як  картинку  в
буфер">
                                    <IconButton       aria-label="copyImage"
onClick={() => saveImageToClipBoard()}>
                                        <ContentCopyIcon
className={style.contentCopyIcon}/>
                                    </IconButton>
                               </Tooltip>
                          </div>
                     </div>
                     <Modal
                          open={modalOpen}
                          onClose={() => setModalOpen(false)}
                     >
                          <Box sx={styleModal}>
                               <CreateTrelloCard       trelloKey={trelloKey}
trelloApiKey={trelloApiKey}                        closeModal={closeModal}
image={getImageAsFile} description={trelloDesc}/>
                          </Box>
                     </Modal>
               </>
          );

          return(
               <>
                     {plotControl}
                     <div>
                          <Bar         ref={chartRef}         data={chartData}
options={chartOptions} height={480}/>
                     </div>
```

```
            </>


      );
};

export default BarChart;
```