Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.896

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: "Розроблення автоматизованої системи менеджменту організаційно-виробничих процесів на сільськогосподарському підприємстві"

Спеціальність – 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ - 22.00.00.000

Студент
ІПЗ-42 _____ /Ренат ЧЕРНОВ/

Науковий керівник
к.ф.-м.н., доц._____/Ольга СУПРУН/

Допускається до захисту
з питань нормоконтролю
_____/Тамара ЧАПОВСЬКА/

д.т.н., проф. _____/Олексій БИЧКОВ/

Київ - 2021

Рішенням Екзаменаційної комісії

випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії

д. т. н., проф. Віктор ВИШНІВСЬКИЙ

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра програмних систем і технологій
Освітньо-кваліфікаційний рівень бакалавр
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО
Зав. кафедри програмних систем і технологій
_____(Олексій БИЧКОВ)

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Чернову Ренату Сергійовичу

1. Тема бакалаврської роботи “Розроблення автоматизованої системи менеджменту організаційно-виробничих процесів на сільськогосподарському підприємстві”, керівник роботи Ольга СУПРУН, к.ф.-м.н., доцент затверджені на засіданні кафедри програмних систем і технологій, протокол №6 від «11» листопада 2020 р.

2. Строк подання студентом роботи «__» 2021 р.

3. Вихідні дані до роботи: Концепції обліку та менеджменту на підприємстві, технологія автоматизації методів обліку та організаційно-виробничих процесів.

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз ефективності впровадження
2. Аналіз завдань, які вирішуються системою
3. Аналіз аналогів та прототипів
4. Обґрунтування вибору програмної реалізації
5. Обґрунтування вибору технічних засобів

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Неблокуюча модель введення(рис. 1.1, ст. 22)
2. Організація асинхронного введення(рис. 1.2, ст. 22)
3. Схема зіставлення об’єктів між node і mongodb(рис. 1.3, ст. 29)
4. Рейтинг мов програмування в 2017 за версією GitHub(рис. 1.4, ст. 30)
5. Структура СУБД MongoDB(рис. 1.5, ст. 32)
6. Схема клієнт-серверної архітектури (рис. 3.1, ст. 41)
7. Клієнт-серверна архітектура додатку (рис. 3.2, ст. 42)
8. Схема MVC (рис. 3.3, ст. 45)
9. Авторизація (рис. 3.4, ст. 47)
10. Помилка введеного паролю (рис. 3.5, ст. 49)
11. Реєстрація ними (рис. 3.6, ст. 50)
12. Вхід до системи. Логін не авторизований (рис. 3.7, ст. 50)
13. Головна сторінка (рис. 3.8, ст. 51)
14. Сторінка editField (рис. 3.9, ст. 52)
15. Сторінка editField / карта місцевості (рис. 3.10, ст. 52)
16. Сторінка Ділянки (рис. 3.11, ст. 53)
17. Сторінка Ділянки/Створити ділянку(рис. 3.12, ст. 54)
18. Сторінка Ділянки / createLandlord(рис. 3.13, ст. 55)
19. Сторінка Ділянки / CreateContract (рис. 3.14, ст. 56)

20. Сторінка Орендодавець (рис. 1.15, ст. 56)
21. Сторінки Відділки(рис. 3.16, ст. 57)
22. Сторінки Договори оренди (рис. 3.17, ст. 57)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
РОЗДІЛ 1	Ольга Супрун		
РОЗДІЛ 2	Ольга Супрун		
РОЗДІЛ 3	Ольга Супрун		

7. **Дата видачі завдання** 27 жовтня 2020 р.

Керівник _____ (Ольга СУПРУН)

Завдання прийняв до виконання _____ (Ренат ЧЕРНОВ)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	**Назви етапів бакалаврської роботи**	**Строк виконання етапів роботи**	**Відмітка про виконання**
1	Уточнення постановки задачі	25.10.2020	виконано
2	Аналіз літератури	10.11.2020	виконано
3	Аналіз існуючих методів, концепцій та алгоритмів вирішення завдання	30.11.2020	виконано
4	Опис розробленого алгоритму	15.03.2021	виконано
5	Розроблення програмного забезпечення	02.04.2021	виконано
6	Тестування розробленого програмного забезпечення	01.05.2021	виконано
7	Оформлення і друк пояснювальною записки	16.05.2021	виконано
8	Оформлення презентації	27.05.2021	виконано
9	Отримання рецензії	05.06.2021	
10	Затвердження пояснювальної записки роботи завідувачем	10.06.2021	
11	Захист дипломної роботи	23.06.2021	

Студент – бакалавр _____(Ренат ЧЕРНОВ)

Керівник роботи _____(Ольга СУПРУН)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 60 с., 21 рис., 1 табл., 10 додат., 8 джерел.

Тема: Розроблення автоматизованої системи менеджменту організаційно-виробничих процесів на сільськогосподарському підприємстві

Об'єкт дослідження: процес менеджменту та обліку полів на сільськогосподарському підприємстві

Предмет дослідження: технологія автоматизації методів обліку та їх вдосконалення, на прикладі створення системи менеджменту для підприємства селянське (фермерське) господарство "Лан"

Мета роботи: автоматизути виробничі процеси в сільському господарстві, максимально виключаючи людський фактор, на скільки це можливо.

Результати досліждення: Розроблено додаток, за допомогою якого сільськогосподарське підприємство може контролювати виробничі процеси у компанії.

Висновок: В роботі розроблено систему для обліку та менеджменту полів на сільськогосподарському підприємстві. Система спрощує роботу фермерам та їх бухгалтерам, дозволяє швидко занести в базу потрібну інформацію, відобразити інформацію про стану ділянок, орендарів, орендодавців, контрактів та автоматично генерувати необхідні документи на підставі існуючої інформації.

ФЕРМЕРСЬКЕ ГОСПОДАРСТВО, АВТОМАТИЗАЦІЯ, АВТОМАТИЗОВАНА СИСТЕМА, МЕНЕДЖМЕНТ, ОБЛІК, ОПТИМІЗАЦІЯ.

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 60 с., 21 рис., 1 табл., 10 доп., 8 источники.

Тема: Разработка автоматизированной системы управления организационно-производственных процессов на сельскохозяйственном предприятии

Объект исследования: процесс менеджмента и учета полей на сельскохозяйственном предприятии

Предмет исследования: технология автоматизации методов учета и их совершенствования на примере создания системы менеджмента для предприятия крестьянское (фермерское) хозяйство "Лан"

Цель работы: автоматизировать производственные процессы в сельском хозяйстве, максимально исключая человеческий фактор, на сколько это возможно.

Результаты исследования: Разработано приложение, с помощью которого, сельскохозяйственное предприятие может контролировать производственные процессы в компании.

Вывод: В работе была разработана система для учета и менеджмента полей на сельскохозяйственном предприятии. Система упрощает работу фермерам и их бухгалтерам, позволяет быстро занести в базу нужную информацию, отразить информацию о состояния участков, арендаторов, арендодателей, контрактов и автоматически генерировать необходимые документы на основании существующей информации.

СЕЛЬСКОХОЗЯЙСТВЕННОЕ, АВТОМАТИЗАЦИЯ, АВТОМАТИЗИРОВАННЫХ СИСТЕМАХ, МЕНЕДЖМЕНТ, УЧЕТ, ОПТИМИЗАЦИЯ.

ANNOTATION

Thesis: 60 pp., 21 fig., 1 table, 10 appendix, 8 sources.

Topic: Development of an automated management system for organizational and production processes at an agricultural enterprise

Object of research: the process of field management and accounting in an agricultural enterprise

Subject of research: technology of automation of accounting methods and their improvement, on the example of creating a management system for the enterprise peasant (farmer) farm "Lan"

Purpose: to automate production processes in agriculture, eliminating the human factor as much as possible.

Research results: An application developed with which an agricultural enterprise can control production processes in the company.

Conclusion: In the work has been developed a system for accounting and field management at an agricultural enterprise. The system simplifies the work of farmers and their accountants, allows you to quickly enter the necessary information in the database, display information about the status of plots, tenants, landlords, contracts and automatically generate the necessary documents based on existing information.

FARM, AUTOMATION, AUTOMATED SYSTEM, MANAGEMENT, ACCOUNTING, OPTIMIZATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД - база даних.

ЕОМ - електронно-обчислювальна машина.

ІС - інформаційна система.

ІТ - інформаційні технології.

НДІ - науково-дослідницький інститут.

ООП - об’єктно-орієнтоване програмування.

ОС - операційна система.

ПЗ - програмне забезпечення.

СУБД - система управління базами даних.

ГК - гамма каротаж.

НГК - нейтронний гамма каротаж.

АК - акустичний каротаж.

МВП - метод власних потенціалів.

СФГ - селянське (фермерське) господарство.

ВСТУП

Якщо в деяких галузях інформаційні технології ще не знайшли ефективного застосування і скоріше є модним віянням, то прогресивне сільське господарство просто не може без нього існувати.

Здавалися раніше футуристичними, але тепер нові рішення IT у сільському господарстві є абсолютною буденністю сьогодні. Причому якщо раніше найбільш передові технології були долею виключно великих агрохолдингів, тепер невеликі фермерські господарства використовують роботизовані системи, різного роду сенсори, передове програмне забезпечення та дронів. Через те, що популяція людей на планеті збільшується і к 2050-ому року, передбачається, що населення нашої планети буде 10 мільярдів. Тому розвиток та впровадження нових технологій в агропромислову галузь є дуже важливим та актуальним.

IT-технології дійсно є обов'язковою складовою в сучасному ефективному сільському господарстві. При масовому впровадженні подібних технологій в Україні, ми не тільки продовжимо досягати чергових світових рекордів з поставки сільськогосподарської сировини, а й підемо в глибшу переробку для створення складного продукту, який репутаційно і економічно зміцнить нашу країну.

Однією з гострих проблем агропромислового виробництва є невисокі оперативність і ефективність прийнятих управлінських рішень внаслідок недостатнього розвитку інтелектуальної і культурної середовища в сільських районах, недостатнього використання, в тому числі в господарській практиці на місцях, нових інформаційних технологій.

Взагалі, ознакою застосування інформаційних технологій в сільському господарстві є наявність комп'ютерів, а також їх з'єднання з Інтернетом. Інформаційні технології використовуються в основному для обліку та автоматизації сільськогосподарських процесів.

Останнім часом в сфері сільського господарства все частіше з'являються умови і додаються значні зусилля по впровадженню інформаційних технологій. Найбільш відомі технології реалізовані в рамках прикладних комп'ютерних програм. Також, через обмеження обсягів інформації, що зберігається

в голові людини тільки кілька факторів можуть бути розглянуті одночасно, в зв'язку з цим також застосовують інтуїтивні методи.

Процес менеджменту та обліку полів на сільськогосподарському підприємстві

Технологія автоматизації методів обліку та їх вдосконалення, на прикладі створення системи менеджменту для підприємства селянське (фермерське) господарство "Лан".

Мета бакалаврської роботи, автоматизути виробничі процеси в сільському господарстві, максимально виключаючи людський фактор, на скільки це можливо.

У селянському (фермерському) господарстві «ЛАН» до сих пір для пошуку та менеджменту необхідних даних о пайовиках та полях, переглядають величезні стовпи документів та не зручні для перегляду таблиці exel.

З розвитком прогресу і ринку програмного забезпечення, на підприємстві виникла необхідність створити програмний продукт, який буде здатен скоротити витрати і зусилля робітників компанії, а головне оперативно видавати результат необхідний працівникові, а також замінити архіви на структуроване зберігання в електронному вигляді.

Для системи важливим фактором є технічні характеристики обладнання) - від цього залежить результат і продуктивність такої системи. Ще одним важливим фактором є як добре організована логіка програми, так і сам користувальницький інтерфейс.

Про своєчасність і актуальність даної проблеми говорить той факт, що більшу частину свого часу бухгалтери та керівники витрачають на оформлення звітів.

Тому в даний час все більш актуальним стає автоматизація видів діяльності людей шляхом створення спеціалізованих систем.

Була розроблена система агрономічного обліку власних полів СФГ "ЛАН" та полів, які беруться у довгорічну оренду.

Ця програма збільшує оперативність роботи робітників, дозволяє швидко занести в комп'ютер потрібну інформацію. Вона зменшує роботу з паперами співробітників організації, зберігаючи великі обсяги інформації в базі даних, в якій можна швидко знайти будь-яку необхідну інформацію всього за кілька секунд.

Основним результатом є:

1. запропонований автором власна система та підхід до її реалізації;

2. приведена реалізація та впроваджено додаток для менеджменту та обліку у селянському (фермерському) господарстві «ЛАН», Запорізька обл. м. Вільнянськ.

За результатами досліджень та розробок, проведених у бакалаврській роботі, підготовлено доповідь для участі в 8-ій Східно-Європейський конференції «Математичні та програмні технології Internet of Everything», що відбулася в м. Києві 14 травня 2021 р. та опубліковано тези

РОЗДІЛ 1
АНАЛІТИЧНА ЧАСТИНА

У цьому розділі необхідно розглянути завдання, які буде виконувати розроблена система, вивчити аналоги та прототипи, виявити їх достоїнства та недоліки. А також обрати СУБД та середовище програмування.

1.1. Завдання, які вирішуються СФГ «ЛАН»

Селянське (Фермерське) господарство «ЛАН» займається вирощюванням зернових та масляних культур, а також вирощуванням інших сезонних культур та їх зберіганням.

Основними завданнями сільськогосподарського підприємства є:

– забезпечення збереження та вирощування зернових та масляних культур на власних або орендованих полях;

– дотримання встановлених норм;

– формування звітів для податкової служби;

– отримання точних відомостей про юридичний стан полів для видачі паїв.

1.2. Технічне завдання на проектування

Ціль роботи - розробка системи для обліку та менеджменту полів у СФГ «ЛАН» Передбачається, що використовувати цю систему будуть працівники підприємств: бухгалтера та керівництво, які займаються обліком полів та підписанням договорів оренди. Працівникам підприємств доводиться виконувати велику кількість дій для пошуку зведеної інформації про підприємство в цілому. Виконання цієї роботи вручну вимагає значного часу. Розроблена і система дозволяє значно зменшити час роботи співробітників та розширити процес отримання різної інформації. Система розробляється на основі заказу від СФГ «ЛАН».

Дана програма повинна виконувати наступні завдання:

– організувати зберігання, обробку даних та вивід на друк;

– максимально зменшити затрати праці на обробку інформації;

– мати зручні форми вводу-виводу з інтерфейсом для користувача;

– виключити ймовірність допущення арифметичних та логічних помилок;

– виводити на друк певні форми документів;

– організувати створення звітів та документів.

1.3. Ефективність впровадження

Існує ряд показників ефективності системи, які відображаються на результатах діяльності, за рахунок підвищення рівня управління, оперативності прийнятих рішень.

До них відноситься:

– автоматичний розрахунок обробленої площі поля;

– облік земельного банку, реєстр договорів оренди;

– підвищення оперативності та актуальності інформації;

–скорочення термінів вирішення окремих завдань та прийняття управлінських рішень;

– підвищення якості інформації, її точність, детальність;

– зниження кількості часу, який витрачається на підготовку документів, швидкість видачі вихідних документів;

– посилене контролювання, запобігання зловживанню;

– підвищення якості праці за рахунок скорочення рутинних операцій.

1.4. Завдання, які вирішуються системою, що розробляється

Платформа допомагає ефективно займатися сільським господарством, а також підвищує маржинальність бізнесу. Автоматизує виробничі процеси в сільському господарстві, виключає людський фактор на скільки це можливо. Автоматична фіксація польових робіт, витрати палива, розрахунок. Користувач в єдиному вікні бачить всі процеси виробництва та може вчасно виконувати важливі рішення.

В системі закладений широкий функціонал для обліку полів та їх оренди. При цьому програма зручна у використанні і навіть непрофесійному користувачеві з нею легко працювати і орієнтуватися.

Програма забезпечує працюючого цілою низкою узагальнюючих і аналітичних звітів, які допомагають швидко і легко знайти будь-яку необхідну інформацію в простому для розуміння інтерфейсі користувача.

Система пропонує ряд таких корисних для користувача можливостей, як:

Облік договорів оренди:

- Кадастровий номер паю;
- Реєстраційний номер;
- Дата реєстрації;
- Термін дії;
- Дата підписання договору;
- Дата закінчення договору.

Облік відділків;

Облік ділянок:

- Кадастровий номер паю;
- Площа;
- Тип;
- Статус;
- Номер акта;
- Орендодавець;
- Орендар;
- Договір оренди.

Облік орендарів;

Облік орендодавців:

- Прізвище Імя Побаткові;
- ІПН;
- Адреса
- Номер та серія паспорта;
- Номер Телефону;
- Статус.

1.5. Огляд аналогів і прототипів

Було проаналізовано основні українські системи, які допомагають агрокомпаніям оптимізувати виробництво, поліпшують планування і можуть

мінімізувати втрати врожаю. Також описано їх можливості і склано порівняльну таблицю, завдяки якій кожен зможе підібрати рішення, яке більше підійде саме його компанії.

1.5.1. Soft.farm

Soft.Farm - комплексне IT рішення для агровиробників, яке дозволяє об'єднати дані з інших систем в єдиний формат, створити аналітичну систему сільськогосподарської діяльності для прийняття управлінських рішень.

В одному сервісі об'єднанні агрономічні IT інструменти, які необхідні для впровадження точного землеробства та інших технологій:

- земельний банк;
- агротехнологія;
- агроскаутінг;
- картограми;
- контроль витрат ON-LINE;
- метеоспостережень.

Система дає можливість уникнути помилок в обліку і плануванні зоотехнічних заходів, включає в себе:

- облік поголів'я стада;
- журнали зоотехнічних заходів;
- раціони харчування;
- економічний аналіз.

1.5.2. Агроконтролер

АгроКонтролер - програмне рішення, що дозволяє моделювати, контролювати і управляти виробничими процесами агропідприємства. Система включає в себе основні блоки управління рослинництвом: планування, обстеження полів, моніторинг техніки, списання матеріалів, управлінський облік, аналітику даних.

- Моделювання, облік і аналітика показників
- Технічний супровід і консультування онлайн
- Моніторинг за все бізнесу в одній системі

- Покроковий офлайн майстер обстеження
- агрономічний довідник
- Інтеграція з будь-якими системами підприємства (GPS, 1C, ISOBUS)

1.5.3. Agro-online

AgroOnline («Агро Онлайн») - компанія є розробником і системним інтегратором передових рішень в області управління аграрними підприємствами. Повний комплекс рішень: управління активами, бізнес планування, контроль і моніторинг виробничої діяльності, план / фактний економічний аналіз.

Сервіс «АгроOnline» - сучасна онлайн платформа управління аграрним бізнесом. Замість десятків вузькоспрямований рішень - одне комплексне, інтеграція будь-яких зовнішній сервісів в єдину платформу прийняття рішень.

Управління активами:

- Земельний банк.
- Відносини з пайовиками.
- Агрохімічний склад грунту.
- Управління складськими запасами.
- Персонал.
- Консультанти.
- Машини.
- Агрегати.
- Інфраструктура.

Бізнес планування:

- Електронні технологічні карти.
- Карти точного землеробства.
- Електронний сівозміну.
- планування закупівель.
- Планування навантаження техніки.
- передсезонна аналітика.

Моніторинг сезонних процесів

Контроль стану посівів

- Супутниковий моніторинг
- моніторинг дронамі
- маршрутизація агрономів
- польові звіти

1.5.4. PreAgri

PreAgri - це онлайн сервіс для збору і аналізу всіх просторових даних сільськогосподарських підприємств, а так само для планування і контролю виконання польових робіт.

Використовуючи апаратні рішення для точного землеробства в комплексі з геоінформаційних сервісом, Ви будете мати повну картину про процеси, які відбуваються в поле. Спираючись на зібрані дані, Ви зможете приймати ефективні рішення, що дозволить економити на кількості внесених продуктів, часу виконання і задіяних ресурсах.

- картування полів
- сівозміну
- Актуалізація оброблюваних площ
- Аналіз польових робіт
- Виявлення пропусків і накладень
- Підрахунок посівного матеріалу і внесених продуктівАгрохіманаліз
- Карти агрохімічного обстеження полів
- Формування карт диференціального внесення добрив
- Підрахунок потреби добрив в конкретних зонах
- Дистанційне зондування
- Моніторинг вегетативної активності
- Виявлення зон з різною продуктивністю
- Розрахунок потреби в азотних добривах
- Карта рельєфу поля
- Трекінг польовий та обслуговуючої техніки

- Візуалізація даних БПЛА з прив'язкою до треку
- Відео інспектування полів

Розглянуті вище аналоги дійсно хороші, зручні в зверненні, багатофункціональні програми, проте вони мають недоліки. Основним недоліком можна назвати високу вартість придбання, впровадження і супроводу. Що є неприпустимим в даний момент для СФГ «ЛАН».

Серед інших недоліків є те, що обидві ці системи розраховані на масового клієнта, а значить, до переліку функцій входять в основному стандартні, приписувані законодавством операції, які виконуються всіма або більшістю підприємств країни. Однак при впровадженні часто виникають такі ситуації, коли система, незважаючи на свою гнучкість, не може здійснити ті чи інші нестандартні дії, виконання яких принципово важливо для користувача.

Висновки до розділу

Було прийнято рішення розробити власну інформаційну систему для обліку руху товарів на складі СФГ «ЛАН». Що забезпечить відносно невисоку вартість замовленого продукту, виконання специфічних завдань, зручний призначений для користувача інтерфейс, що задовольняє вимогам даної організації і, звичайно, виконання програмою її основних функцій.

РОЗДІЛ 2
ПРОЕКТНА ЧАСТИНА

2.1. Обгрунтування вибору середовища програмування

Для реалізації цієї задачі була обрана IDE WebStorm.

WebStorm - це інтегроване середовище розробки для кодування в JavaScript та пов'язаних з ним технологій. Подібно до IntelliJ IDEA та інших середовищ розробки середовищ JetBrains, WebStorm зробить ваш досвід розробки приємнішим, автоматизуючи повсякденну роботу та допомагаючи вам легко виконувати складні завдання.

Ось декілька ключових функцій, які можна отримати за допомогою WebStorm:

- Розумний редактор із заповненням коду , виявленням помилок на льоту , безпечним переробкою коду та швидкою навігацією по всій кодовій базі .
- Різноманітність вбудованих інструментів для розробників , які дозволяють налагоджувати і тестувати на сторону клієнта і Node.js додатки, а також для роботи з контролем версій , пухом , будівельними інструментами , терміналів і HTTP клієнта .
- Інструменти для ефективної роботи в команді , включаючи послугу для віддаленої спільної розробки та програмування пар та можливість ділитися конфігурацією проекту з іншими.
- Можливість налаштувати робоче середовище , експериментуючи з такими речами як теми і плагіни .

Основними причинами вибору цієї IDE була його тісна інтерграція з фреймворками JavaScript, також дуже важливим було те що в WebStorm є вбудований отладчик тобто можливо було відкрити точки останова в вихідному коді, переглянути значення в стеку, простежити за значеннями і використовувати інтерактивну консоль.

Також перевагою було інтеграція з системами контролю версій. Була можливість використовувати простий універсальний інтерфейс для роботи з Git,

GitHub, Mercurial і іншими системами контролю версій. Робити комміти, переглядати внесені зміни і вирішувати конфлікти можна було прямо в IDE.

2.2. Обґрунтування вибору програмної реалізації

Використання JavaScript як серверної мови стало можливим за допомогою Node.js - програмної платформи, який транслює JavaScript в машинний код і працює на двигуні V8. Ця платформа перетворює JavaScript з вузькоспеціалізованої мови програмування на мову загального призначення.

Використання Node.js має багато суттєвих переваг.

Однією з головних є використання Неблокуючої моделі введення і виведення. Принцип роботи цієї моделі зображений на рис. 1.1.

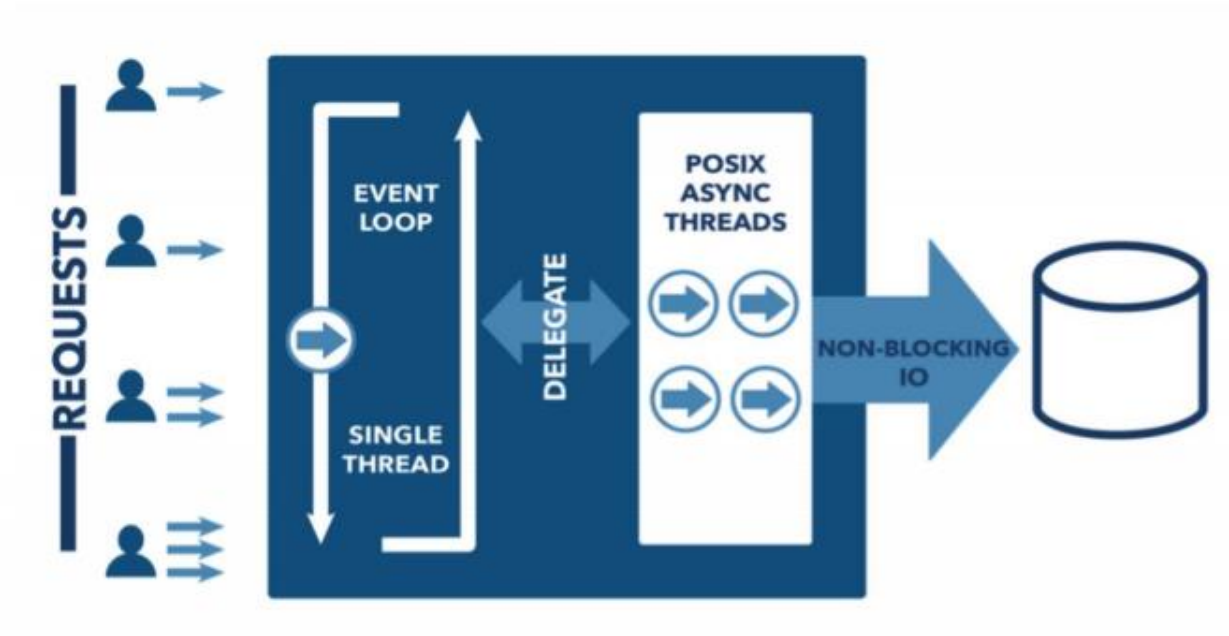


Рис. 2.1. Неблокуюча модель введення і виведення

Така модель дає можливість обробляти велику кількість з'єднань, що має величезний вплив на масштабування системи. З використанням Node.js немає потреби турбуватися про можливе блокування процесів, оскільки функції не працюють з пристроями введення / виведення напряму.

Розглянемо докладніше асинхронне введення і виведення

Node.js використовує єдиний потік, а цикл подій бере на себе відповідальність завсі асинхронні операції введення і виведення. Поведінка програми регулюєтьсяасинхронними викликами функцій зворотного виклику. Такий підхід дозволяє уникнути проблем, пов'язаних з формуванням окремих потоків і їх блокуванням. Крім того, це дає істотний приріст в швидкості відповідей сервера і масштабованості системи.

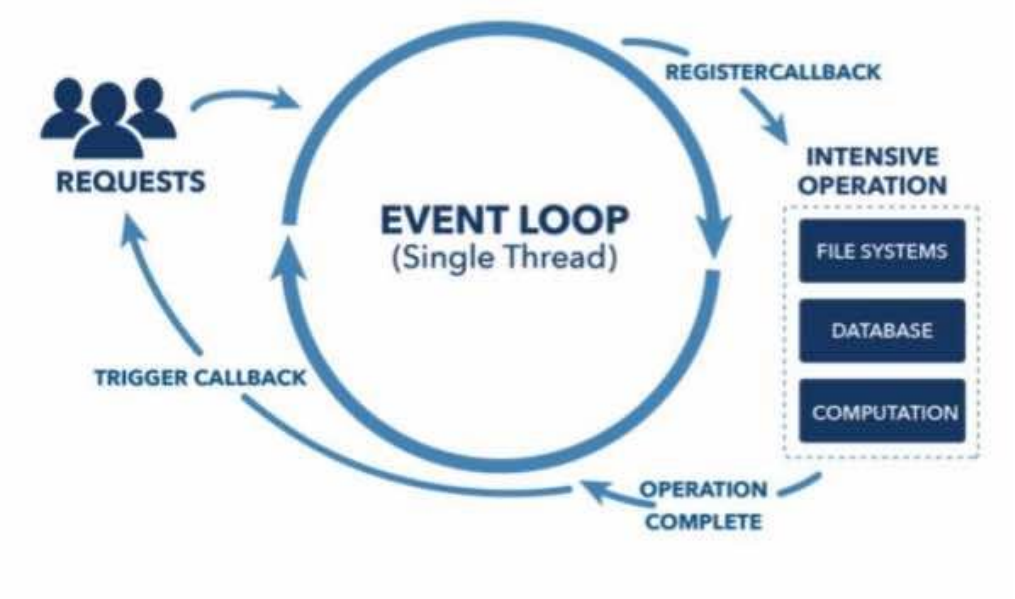


Рис. 2.2. Організація асинхронного введення / виводу в Node.js

Використання JavaScript як серверної мови дає значний приріст в швидкості.

Так як Node.js працює на основі архітектури управління подіями, то він як найкраще підходить для створення додатків реального часу оскільки і серверна і клієнтська частина написані на JavaScript, процес синхронізації між ними набагато швидше, легше і зручніше.

Крім вбудованого функціоналу, який поставляє сам Node.js, в його склад входить також менеджер модулів npm (node package manager).

Його основна ідея - це створення малого програмного блоку, який вирішує одну конкретну проблему. Це дає можливість компонувати великі проекти з цих маленьких побудованих блоків. До того ж, це задовольняє принцип відкритості і багаторазового використання коду.

Найбільш популярними npm-модулями є:

- express - потужний фреймворк для веб-програмування. є стандартом для більшості існуючих на даний момент додатків Node.js;
- connect - це розширюваний фреймворк, який працює з Node.js як HTTPсервер, що надає набір високопродуктивних плагінів як сполучного програмного забезпечення;
- socket.io і socket.js - серверні бібліотеки для обміну даними в режимі реального часу; Ще однією особливістю Node.js є те, що він орієнтований на роботу з NoSQL базами даних, зокрема MongoDB. Такий підхід дозволяє ще поліпшити масштабованість продукту і швидкість його роботи.

Також, слід зазначити, що Node.js підтримує архітектурний шаблон програмування MVC (model-view-controller), який вважається загальноприйнятим стандартом. Із самої назви платформи «node» (вузол), можна побачити, що вона акцентує увагу на можливість побудови систем з великої кількості невеликихрозподілених обчислювальних вузлів, які мають можливість обмінюватися даними між собою. Сервер на Node.js ніколи не виявиться могутніше, ніж потрібно. Вся суть архітектури Node - в її мінімалізмі. При цьому серверну частина програми можна масштабувати в залежності від потреб проекту.

Для перевірки усіх типів запитів та отримання відповідей використовувалась програма – POSTMAN.

Також було використано менеджер пакетів npm – це найбільший у світі реєстр програмного забезпечення. Розробники з відкритим кодом з усіх континентів використовують npm для спільного використання та запозичення пакетів, а багато організацій використовують npm для управління приватною розробкою.

npm складається з трьох різних компонентів:

- веб-сайт
- інтерфейс командного рядка (CLI)
- реєстру

Npm може:

- Адаптувати пакети коду для програм або включати пакети такими, які вони є.
- Завантажувати окремі інструменти, якими ви можете скористатися відразу.

- Запускати пакунки без завантаження за допомогою npx .
- Ділитися кодом з будь-яким користувачем npm де завгодно.
- Обмежувати код лише певними розробниками.
- Створювати організації для координації обслуговування пакетів, кодування та розробників.
- Формувати віртуальні команди, використовуючи організації.
- Керувати кількома версіями коду та залежностями коду.
- Легко оновлювати програми, коли оновлюється базовий код.

Також підтримка Express.js механізму eTag значно підвищує продуктивність програми і полегшує роботу з кешуванням. ETag або entity tag - це частина HTTP протоколу, а саме механізм, за допомогою якого HTTP виконує перевірку кеша і який дозволяє клієнту робити умовний запит. Це суттєво економить пропускну здатність і дозволяє кешу бути більш ефективним, оскільки вебсервера не потрібно відправляти повну відповідь, якщо зміст ресурсів не змінився.

Розглянемо докладніше технологію eTag. ETag є закритий ідентифікатор, який присвоюється веб-сервером на певну версію ресурсу, який знаходиться за певною URL. Якщо зміст даного ресурсу для цієї адреси змінюється на нове, призначається новий eTag. Використання eTag в такому напрямку аналогічно використанню відбитків пальців, які можна швидко порівняти і визначити, є дві версії даного ресурсу однаковими.

Загальні методи створення eTag включають в себе використання стійкі до колізій хеш-функції змісту даного ресурсу, хеш часу останньої модифікації ресурсу або навіть просто номер версії. Щоб уникнути отримання неактуальних даних кеша методи повинні гарантувати, що кожен eTag є унікальним значенням.

Однією з найбільших переваг Express.js є те, що фреймворк базується на middleware - функціях проміжного виконання. найбільш поширеними прикладами middleware є:

- компресія - дозволяє стискати веб-сторінки в формат gzip для більш швидкої передачі на сервер;

- парсер cookie - дає можливість працювати з cookie;
- сесія cookie - дає можливість генерувати інформацію сесії, поки користувач залишається на сайті;
- статичне обслуговування - дозволяє повертати статичні файли, які зберігаються в директорії сервера;
- csrf-захист - надає захист серверної інформації від csrf-атак.

Оскільки Express.js підтримує model-view-controller архітектуру, це надає зручний спосіб для побудови веб-додатків, використовуючи формат modeldriving, тобто управління моделями. Цей стиль розробки програмного забезпечення означає, що основними об'єктами розробки є моделі - абстрактний опис програмного забезпечення, який приховує інформацію про деякі аспекти з метою надання спрощеного опису інших аспектів. Модель фіксує інформацію у формі, придатній для інтерпретації людьми і обробки іструментами. Відповідно до цього підходу, спочатку розробляється модель додатку, незалежна від імплементуючої технології, потім трансформується в платформозалежну модель, яка вже потім переводиться у вихідний код.

React-Native. Клієнтська частина

Для створення клієнтської частини додатку був обраний фреймворк для створення кросплатформенних застосунків React Native.

React Native - це фреймворк з відкритим кодом, який дозволяє створювати міжплатформні мобільні додатки за допомогою JavaScript та React, яка є бібліотекою JavaScript з відкритим кодом, розробленою Facebook, Instagram та іншими розробниками для побудови користувальницьких інтерфейсів. React Native використовує JSX для створення додатків для Android, а також iOS.

Все більша кількість розробників обирають React Native для розробки міжплатформенних мобільних додатків.

Програми React Native функціонують так само, як і iOS (створені за допомогою Swift або Objective-C) та додатки для Android (розроблені за допомогою Kotlin або Java). React Native - це потужний інструмент, який допомагає створювати

програми з прекрасними інтерфейсами. Як результат, створення мобільних додатків стає простим та ефективним як за часом, так і за вартістю.

Тим не менш, цей фреймворк має деякі обмеження і не завжди є найкращою альтернативою для всіх проектів. Є деякі плюси і мінуси React Native, про які розробники повинні знати. Давайте розглянемо переваги та недоліки React Native, які можуть допомогти вам вибрати найкраще рішення. Починаючи обговорення плюсів і мінусів фреймворку React Native, давайте почнемо з переваг фреймворку React Native.

Плюси React Native:

1. Кросплатформеність. Хотіли писати додаток спочатку під iOS, але додатковий запуск під Android - величезний плюс.
2. Швидко і дешево. Як наслідок першого пункту, не потрібно підтримувати дві платформи. Додаємо нову функціональність на React Native під одну платформу, а 80% коду працює і на інший.
3. Велике співтовариство. У Telegram є спільнота React Native на 3000 розробників. Можна не тільки отримати відповідь на будь-яке питання, але і подивитися, хто на які граблі наступав, і від чого варто відмовитися.
4. Багатий вибір доступних компонентів і готових рішень. Головна умова завдання - швидкий запуск, тому не було часу писати бібліотеки і рішення з нуля. Готові компоненти і рішення нам в цьому допомогли, як і в переході від ReactJS до React Native.

   З React Native є два способи розробити додаток: використовувати Expo або React Native CLI.

2.3. Обгрунтування вибору БД

MongoDB - це база даних NoSQL без схеми. Це означає, що можна зберігати в ньому документи JSON, і структура цих документів може варіюватись, оскільки вона не застосовується так, як бази даних SQL. Це одна з переваг використання NoSQL, так як це прискорює розробку додатків і зменшує складність розгортання.

На рис.2.3. наведено схему зіставлення об'єктів між Node і MongoDB, кероване через Mongoose:

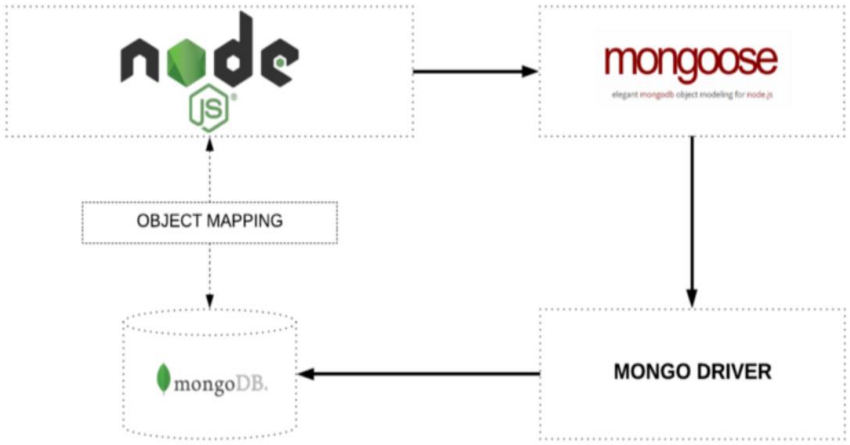


Рис. 2.3. Схема зіставлення об'єктів між Node і MongoDB

Згідно зі звітом платформи длярозробки програмного забезпечення GitHub за 2017, JavaScript впевненозаймає перше місце за популярністю серед учасників (рис. 2.4).

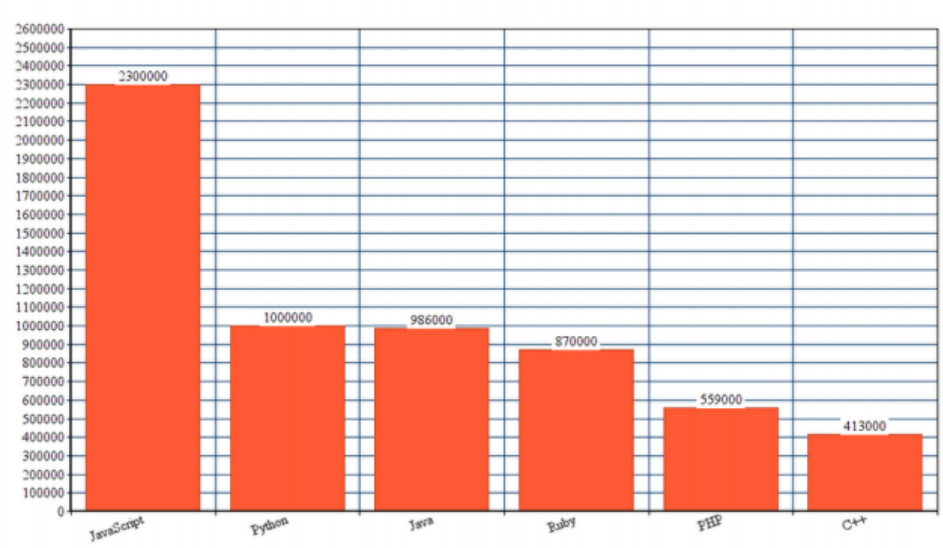


Рис. 2.4. Рейтинг мов програмування в 2017 за версією GitHub

Переваги використання MongoDB

Будь-яка реляційна база даних має типовий дизайн схеми, який показує кількість таблиць та взаємозв'язок між цими таблицями. Поки в MongoDB немає поняття стосунків.

Переваги MongoDB перед RDBMS

- Без схеми.
- Структура окремого об'єкта чітка.
- Жодних складних об’єднань.
- Глибокі можливості запитів. MongoDB підтримує динамічні запити в документах з використанням мови запитів на основі документів, яка майже така ж потужна, як SQL.

Навіщо використовувати MongoDB?

- Дані зберігаються у формі документів у стилі JSON.
- Індекс за будь-яким атрибутом
- Реплікація та висока доступність
- Автозаточування
- Розширені запити
- Швидке оновлення на місці
- Професійна підтримка MongoDB

Де використовувати MongoDB?

- Великі дані
- Управління вмістом та доставка
- Мобільна та соціальна інфраструктура
- Керування даними користувача
- Центр даних

Структура MongoDB зображена на рисунку 2.5.

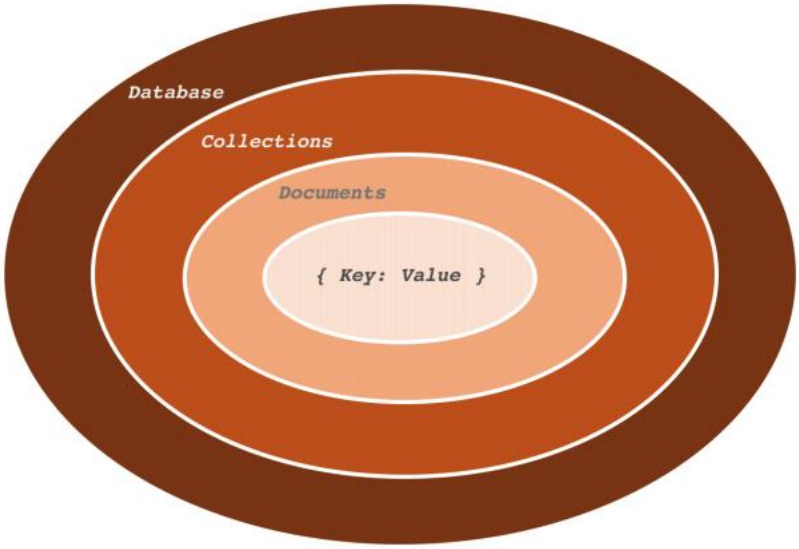


Рис. 2.5. Структура СУБД MongoDB

Бази даних NoSQL були створені у відповідь на обмеження традиційної технології реляційних баз даних. При порівнянні з реляційними базами даних , бази даних NoSQL часто є більш масштабованим і забезпечують чудову продуктивність. Крім того, гнучкість та простота використання їх моделей даних можуть пришвидшити розвиток у порівнянні з реляційною моделлю, особливо в середовищі хмарних обчислень.

Кожен конкретний тип бази даних NoSQL має різні переваги, але всі мають спільні основні характеристики, що дозволяють їм:

- Обробляйти великі обсяги даних на високій швидкості за допомогою масштабованої архітектури
- Зберігайти неструктуровані, напівструктуровані або структуровані дані
- Увімкнути легке оновлення схем та полів
- Бути зручними для розробників
- Бази даних SQL найчастіше реалізуються в масштабній архітектурі, яка базується на використанні дедалі більших комп'ютерів з більшою кількістю центральних процесорів і більшою пам'яттю для підвищення продуктивності.

Бази даних NoSQL були створені в епоху Інтернету та хмарних обчислень, що дозволило легше впровадити масштабовану архітектуру. В масштабній архітектурі масштабованість досягається розподілом сховища даних та роботою з обробки

даних на великому кластері комп’ютерів. Для збільшення потужності в кластер додається більше комп’ютерів.

Цю масштабну архітектуру особливо безболісно застосовувати в середовищах хмарних обчислень, де нові комп'ютери та сховища можна легко додати до кластера.

Розширена архітектура систем NoSQL забезпечує чіткий шлях до масштабованості, коли обсяг даних або трафік зростають. Досягнення того самого типу масштабованості за допомогою баз даних SQL може бути дорогим, вимагати великої кількості інженерії або може бути неможливим.

Фірма фінансових послуг, така як IHS Markit, вимагає високих показників як для прийому даних, так і для їх отримання. Переходячи від реляційної бази даних до MongoDB, IHS Markit повідомляє, що він здатний своєчасно надавати своїм клієнтам фінансову інформацію в 250 разів швидше.

Реляційні бази даних зберігають дані у структурованих таблицях, які мають заздалегідь визначену схему. Для використання реляційних баз даних слід розробити модель даних, а потім дані перетворюються та завантажуються в базу даних.

Коли дані використовуються в додатках, дані повинні бути отримані за допомогою SQL та адаптовані до форми, яка використовується в додатку. Потім, коли дані записуються назад, їх потрібно знову перетворити назад у реляційні таблиці.

Крім того, схеми багатьох баз даних NoSQL є гнучкими і контролюються розробниками, що полегшує адаптацію бази даних до нових форм даних. Це усуває вузькі місця у процесі розробки, пов’язані з проханням адміністратора бази даних переробити базу даних SQL.

Бази даних NoSQL підтримують широко використовувані формати даних:

- Великі дані всіх видів - текстові дані, а також дані часових рядів
- Файли JSON, які є вкладеними в люди читабельними файлами, що складаються з імен та пар значень. Цей формат може охоплювати дуже складні ієрархічні структури батьків-дочірніх, які можуть ефективно зберігатися в базах даних документів

- З простими двійковими значеннями, списками, картами та рядками можна швидко обробляти сховища ключових значень
- Розріджені дані можна ефективно зберігати в стовпчастих базах даних, де нульові значення взагалі не займають місця. Вони також ефективні для інформації, яка не змінюється часто (енергонезалежні дані)
- Мережі взаємопов’язаної інформації можуть зберігатися в базах даних графіків.

Впровадження баз даних NoSQL було зумовлене переважно розробниками, яким легше створювати різні типи програм порівняно з використанням реляційних баз даних.

Бази даних документів, такі як MongoDB, використовують JSON як спосіб перетворення даних у щось набагато більше подібне до коду. Це дозволяє структурі даних знаходитись під контролем розробника.

Крім того, бази даних NoSQL зберігають дані у формах, близьких до типу об’єктів даних, що використовуються в додатках, тому потрібно менше перетворень при переміщенні даних у бази даних та з них.

Бази даних NoSQL можуть зберігати дані у власних форматах, що означає, що розробникам не потрібно адаптувати дані до сховища. Зберігання даних "як є" означає відсутність інтерфейсної системи ETL для набивання напівструктурованих даних у форматі рядків і стовпців, а також менше програм для розробки або придбання для запуску нової бази даних.

У більшості баз даних NoSQL існує сильна спільнота розробників, яка їх оточує. Це означає, що існує екосистема доступних інструментів та спільнота інших розробників, з якими можна зв’язатись.

Крім того, багато баз даних NoSQL можна модернізувати і дозволяти структурі бази даних змінюватися з нульовим простоєм.

Amazon S3

Amazon S3 - це об'єктне сховище, розраховане на зберігання та вилучення будь-яких обсягів даних з будь-якої точки мережі. Це простий сервіс сховища, який

відрізняється найвищою надійністю, доступністю, продуктивністю і безпекою в галузі, а також практично необмеженої масштабованість при дуже низьких витратах. Amazon S3 надає простий інтерфейс веб-сервісу, який можна використовувати для зберігання та вилучення будь-яких обсягів даних в будь-який час з будь-якого місця. Такий сервіс дозволяє просто створювати додатки, що використовують повністю хмарне сховище. Оскільки сервіс Amazon S3 забезпечує широкі можливості масштабування, а плата нараховується тільки за фактично використані ресурси, можна почати роботу з невеликих масштабів і нарощувати додаток в міру необхідності, не жертвуючи при цьому продуктивністю або надійністю.

Сервіс Amazon S3 спроектований для максимальної гнучкої роботи. Ви можете зберігати дані будь-якого типу і в будь-якій кількості; зчитувати ті ж самі дані мільйон разів або тільки для аварійного відновлення; створювати просте FTP-додаток або складне веб-додаток, порівнянне з роздрібним інтернет-магазином Amazon.com. Amazon S3 дозволяє розробникам зосередитися на інноваціях, які не переживаючи про те, як зберігати дані.

Сервіс Amazon S3 дозволяє будь-якому розробнику використовувати переваги, які дає Amazon робота у великому масштабі, без попередньої оплати або зниження власної продуктивності. Тепер розробники можуть вільно вводити нововведення, знаючи, що вони зможуть без праці забезпечити швидкий, постійний і безпечний доступ до даних, незалежно від того, наскільки успішним буде бізнес.

Можна зберігати практично будь-які типи даних в будь-якому форматі. Додаткову інформацію див. В Ліцензійної угоди Amazon Web Services .
Загальний обсяг збережених даних і кількість об'єктів не обмежені. Розмір окремих об'єктів Amazon S3 може становити від 0 байт до 5 ТБ. Найбільший об'єкт, який можна завантажити через один запит PUT - 5 гігабайт. Для об'єктів крупніше 100 Мегабайт клієнтам рекомендується використовувати можливість багатокомпонентної завантаження .

Сервіс Amazon S3 пропонує кілька класів сховища, призначених для різних прикладів використання. До них відносяться сховище загального призначення S3 Standard для часто використовуваних даних, S3 Intelligent-Tiering для даних з

невідомими або мінливими схемами доступу, S3 Standard-Infrequent Access (S3 Standard - IA), S3 One Zone-Infrequent Access (S3 One Zone - IA) для даних, що вимагають тривалого зберігання, але менш частого доступу, Amazon S3 Glacier (S3 Glacier) і Amazon S3 Glacier Deep Archive (S3 Glacier Deep Archive) для довгострокового зберігання та цифрової архівації даних. Якщо ваші вимоги до розміщення даних не можуть бути задоволені існуючими регіонами AWS, ви можете використовувати сховище класу S3 Outposts для локального зберігання своїх даних типу S3. Детальніше про ці класах сховищ см. Насторінці «Класи сховищ Amazon S3» .

Amazon S3 - це просте сховище об'єктів на основі ключа. При зберіганні даних об'єктів призначається унікальний ключ, який може використовуватися згодом для доступу до даних. Ключі можуть мати будь-які строкові значення; їх можна створювати так, щоб імітувати ієрархічні атрибути. Крім того, для організації даних у всіх корзинах і (або) з будь-якими префіксами S3 можна скористатися можливістю призначення тегів об'єктів S3.

Amazon S3 надає прості стандартизовані інтерфейси веб-сервісів REST і SOAP, призначені для роботи з будь-яким інструментарієм інтернет-розробки. Ми навмисно спростили всі системні процеси, щоб зробити простіше додавання нових протоколів роздачі і функціональних рівнів.

Amazon S3 надає будь-якій розробнику доступ до тієї ж високомасштабіруемой, високодоступних, швидкої і недорогий інфраструктурі зберігання даних, яку Amazon використовує для управління своєю власною глобальною мережею веб-сайтів. Клас сховища S3 Standard забезпечує доступність на рівні 99,99%, класи S3 Standard - IA і S3 Intelligent-Tiering - на рівні 99,9%, клас S3 One Zone - IA - на рівні 99,5%, а класи S3 Glacier і S3 Glacier Deep Archive забезпечують доступність на рівні 99,99% і супроводжуються SLA на рівні 99,9%. Використання сховищ всіх перерахованих класів регулюється Угодою про рівень обслуговування Amazon S3.

2.4. Вибір технічних засобів

Для вибору характеристик комп'ютера необхідно врахувати наступні моменти: час отримання результатів запитів до бази даних допустимо в межах декількох секунд, тобто вимоги до швидкості обробки даних невисокі; необхідно передбачити можливий розвиток системи (збільшення обсягу бази даних, перехід до нових версій операційної системи і СУБД, включення системи в локальну обчислювальну мережу і т.д.). Таким чином, технічні вимоги наведені в таблиці 1.

Табл. 2.1.

Технічні вимоги

Характеристика	Вимоги	
	мінімальні	рекомендовані
Процессор	Pentium 166	Pentium II – 400
Об'єм оперативної пам'яті,	128	256
Ємність жорсткого диска,Мбайт	115	675
Мережеві засоби	-	Так
Привід FDD	Так	Так
Привід DVD-ROM	Так	Так
Монітор	15’	17’
Клавіатура	Так	Так
Миша	Так	Так
Прінтер	-	Так

Висновки до розділу

Для реалізації поставлених у даному проекті завдань буде використана СУБД MongoDB та Amazon S3, які найбільше застосування знайшли для управління БД веб-сайтів і різних сервісів. Для розробки системи було обрано середовище програмування WebStorm. Воно дозволяє швидко створювати безпечні і підтримувані веб-сайти. WebStorm бере на себе більшу частину турбот веб-розробки, тому можна зосередитися на написанні свого веб-додатку. Для розробки клієнтської частини додатку було обрано фреймворк React-Native

РОЗДІЛ 3
РОЗРОБКА СТРУКТУРИ ТА ІНТЕРФЕЙСУ

3.1. Розробка архітектури системи

Для даного проекту був обраний клієнт-серверна архітектура, яка, як відомо, буває двох типів: з двома та трьома ланками. Для нашої системи найкраще підходить архітектура з трьома ланками, яка буде складаючись із наступних компонентів:

1. Клієнтський рівень - частина програми, яка буларозроблена для взаємодії з користувачем (User Interface).

2. Сервер - частина програми, в якій міститься вся бізнес та серверна логіка програми. Клієнтський рівень може отримати доступ до функцій сервера через певний інтерфейс.

3. Data Layer - рівень роботи з базою даних. Ця частина системи надає серверу доступ до бази даних.

Вибір такого підходу гарантує наступні переваги:

- високий рівень масштабованості і гнучкості;

- висока продуктивність.

- високий рівень захищеності системи;

Схема клієнт-серверної архітектури з трьома ланками зображено на рис. 3.1.

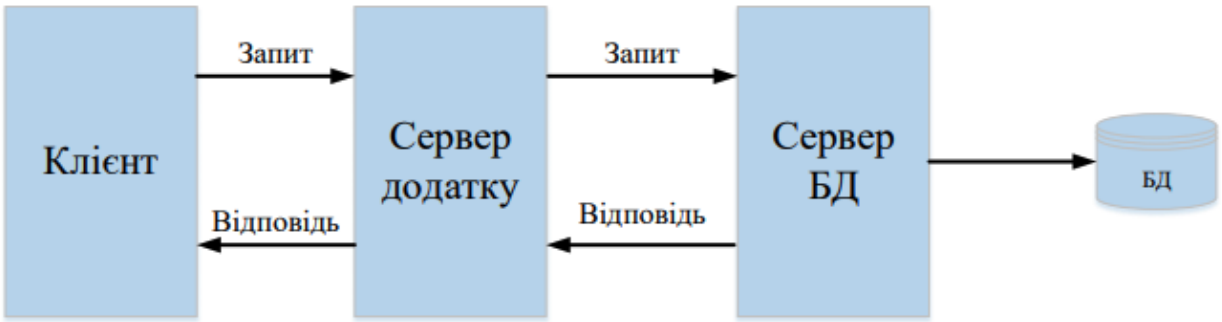


Рис. 3.1. Схема клієнт-серверної архітектури

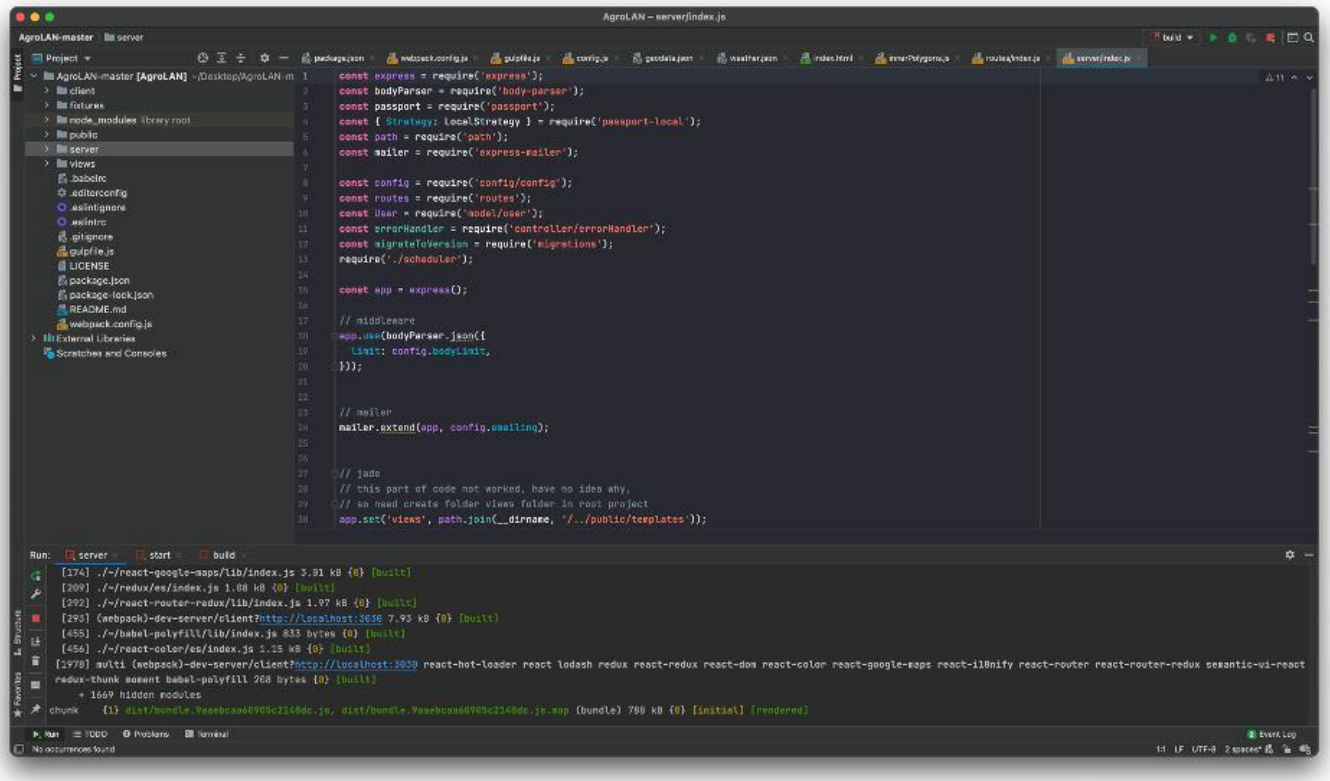


Рис. 3.2. Клієнт-серверна архітектура програми

3.1.1. Клієнт-серверна архітектура

Клієнт-серверна архітектура - це архітектура, в Який розподіляється НАВАНТАЖЕННЯ між постачальником послуг - серверами, і замовниками послуг - клієнтами. ЦІ дві Сторони взаємодіють між собою за допомогою мережевих протоколів (найчастіше це HTTP).

Клієнт-серверна архітектура характеризується:

- відсутністю дублювання коду програми-сервера програмаміклієнтамі;

- оскільки всі обчислення і основна робота по даному віконуються на сервері, значний зніжуються вимоги до комп'ютерів, на яких Працюють клієнтські програми;

- зазвичай сервер захищений набагато краще, ніж програми-клієнти. На цій ділянці системи значень простіше обладнати контроль повноважень, щоб надавати доступ до Даних лише Клієнтам, які ма ють на це відповідні права.

Приведення моделі клієнт-сервер лежить в Основі архітектурного стилю REST. Завдяки цьому обмеженою система набуває такі бажань характеристик як

продуктивність, простота і гнучкість, масштабованість, адаптівність і здатність до змін, портативність та Надійність.

Розмежування потреб є основним принципом, який лежить в основі даної архітектури. Відокремлення потреб інтерфейсу клієнта від потреб серверної частини, яка зберігає дані, підвищує портативність і переносимість коду клієнтської програми на інші платформи. Спрощення серверної частини помітно покращує масштабованість всієї системи. Це дає можливість окремим частинам розвиватися незалежно один від одноі.

Трирівнева клієнт-серверниа архітектура, що складається з представлення даних (User Interface), прикладного компонента (проміжне програмне забезпечення) забезпечує прошарок між сервером додатка і рівнем бази даних) і рівня управління ресурсами (доступ до бази даних).

Трирівнева клієнт-серверних архітектура також може бути розширена до багаторівневої шляхом відділення додаткових серверів, кожний із яких буде здатне надавати власні послуги, а також користуватися послуги друге серверів різного рівня.

3.2 Шаблон проектування системи

Для проектування даної системи використано архітектурний шаблон MVC. MVC - це схема розподіл рівня даних програми, призначений для користувача інтерфейсу і логіки самого додатка на три окремо та незалежні компоненти від одне одного. Модифікація кожного з трьох компонентів може відбуватися незалежно, це дає переваги в портативності і масштабованості системи.

Розглянемо компоненти MVC більш детально:

- модель (Model) - надає дані для обробки і відображення і реагує на відрядження контролера, що змінює своє положення;
- уявлення (View) - відповідає за відображення даних, які надає модель користувача, що реагують на зміни в моделі;
- контролер (Controller) - реагує на дії користувача і оповіщає модель про необхідність змін.

Ідеєю використання цього підходу є розмежування бізнес-логіки (Model) від її візуального виду - уявлення (View). За рахунок такого розмежування виконується принцип повторного використання коду. Крім того, це дуже корисно в ситуаціях, коли користувач повинен мати можливість бачити одні й ті ж дані одночасно, але в різних контекстах або з різних точок зору. А також:

- не вносячи змін до реалізації моделі, до неї можна пріеднаті кілька уявлень. Наприклад, дані можуть бути одночасно представлені у вигляді тексту, графіка, таблиці або гістограмі. Причому на саму модель це ніяк не буде впливати;

- бізнес-логіка і візуальне представлення програми можуть розроблятись повністю незалежно один від одного. Розробник програмної логіки можуть навіть не мати уявлення про те, ласка уявлення буде використовувати.

Структура архітектурного шаблоном MVC зображена на рис. 3.3.

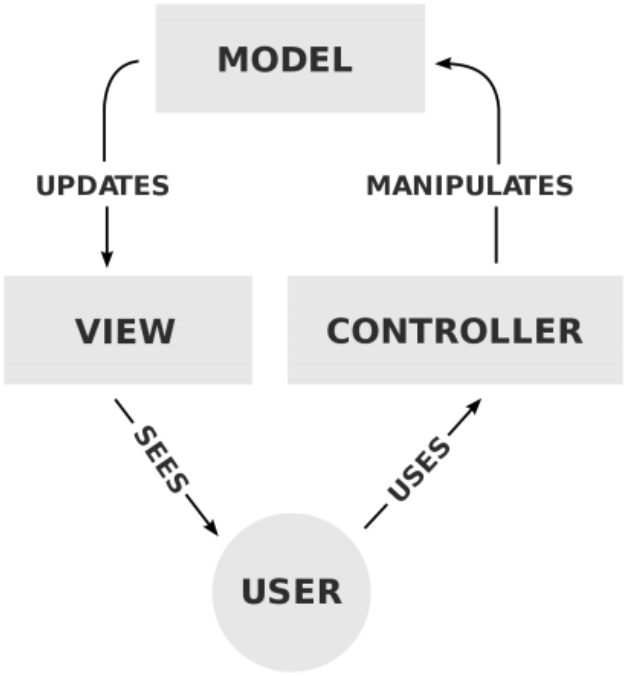


Рис. 3.3. Схема MVC

Модель є незалежною від уявлення, вона зовсім не знає, як ці дані візуалізувати. Також вона повністю незалежна від контролера і не має загальних

точок взаємодії з користувачем. Вона просто надає доступ до даних і управління ними. Подання, в свою чергу, відповідає за отримання необхідних даних від моделі і направляє їх користувачеві. Подання не має можливості впливати на стан моделі, спілкуючись з нею безпосередньо. Також в поданні не виконує жодних дані, які ввів користувач. Контролер забезпечує зв'язок між користувачем і системою. Він вирішує, методи повинні виконуватися в разі того чи іншого дії користувача. Для реалізації необхідного дії він використовує модель і уявлення. MVC є загальним шаблоном і не має суворої реалізації. Немає загальноприйнятого певного місця, де повинна розміщуватися бізнес-логіка додатку. Вона може знаходитися як в контролері, так і в моделі. Деякі фреймворки чітко задають, де повинен знаходитися логіка програми, інші не мають таких установок. Валідація даних, введених користувачем, також не має чітко певного місця розташування. Якщо валідація проста, вона може перебувати навіть в поданні. Але зазвичай вона все-таки знаходиться в контролері або моделі.

Залежно від складності архітектурного рішення для реалізації схеми Model-View-Controller можуть використовуватися такі шаблони проектування:

- «спостерігач»;
- «стратегія»;
- «компоновщик».

Типовою реалізацією шаблону «компоновщик» - коли уявлення відмежоване від моделі шляхом встановлення між ними протоколу взаємодії, використовує «апарат подій».

«Апарат подій» - це позначення «подіями» визначених ситуацій, які виникають в ході виконання програми і розсилки повідомлень про них всім, хто підписався на їх отримання (subscribers). При кожній зміні внутрішніх даних в моделі, яке позначено як «подія», вона сповіщає про це всі уявлення (Views), які підписані на цю подію. при отриманні оповіщення уявлення оновлює свій стан.

Використання шаблону «стратегія» означає, що при обробці реакції користувача саме уявлення вибирає, який контролер буде обробляти цю реакцію і

забезпечувати той чи інший зв'язок з моделлю. Цей шаблон також має модифікацію, яка називається «команда».

Шаблон «компоновщик» використовується переважно для можливості однотипного спілкування з подоб'ектов складно-складеного виду ієрархії.

Модель MVC може існувати в активному і пасивному вигляді. В такому випадку модель (Model) програми містить лише сукупність методів для доступу до даних, а вся програмна логіка додатка міститься в контролері.

В об'єктно-орієнтованому програмуванні застосовується переважно активна модель Model-View-Controller, де модель крім методів для доступу до СУБД містить також і бізнес-логіку програми. Крім того, моделі можуть инкапсулировать в себе і інші моделі. Разом з цим контролери тримаються «Тонкими» - тобто, відповідають лише за:

- прийом запиту від користувача;
- аналіз отриманого запиту;
- вибір наступного дії системи відповідно до результатів аналізу

Таким чином контролер виконує лише функцію сполучної ланки (glue layer) між окремими компонентами інформаційної системи.

3.3. Опис роботи програми

Програма, покликана прискорити роботу користувача, повинна мати інтуїтивно зрозумілий інтерфейс, надавати швидкий доступ до своїх функцій. Це допоможе більше підвищити продуктивність працівника (фермера) і не буде витрачатися багато часу на навчання роботи з програмою.

При запуску програми можна побачити вікно ідентифікації (рис. 3.4.), яке пропонує користувачеві ввести імейл та пароль.

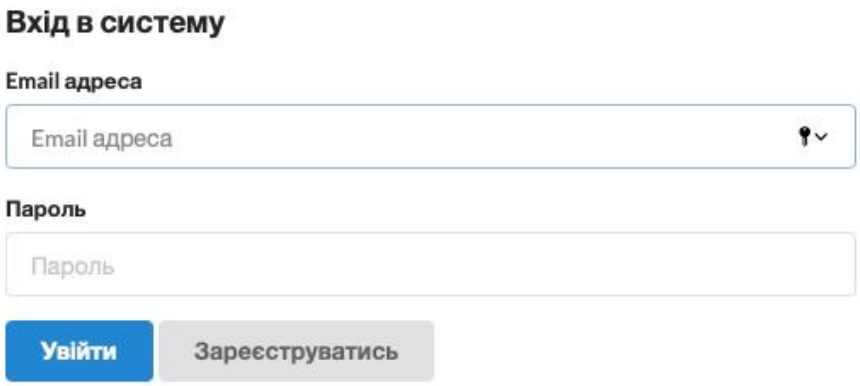


Рис. 3.4. Аутентифікація

При правильному введенні здійснюється вхід в програму. Якщо пароль був введений неправильно, то з'являється вікно що повідомляє про помилку введеного пароля.

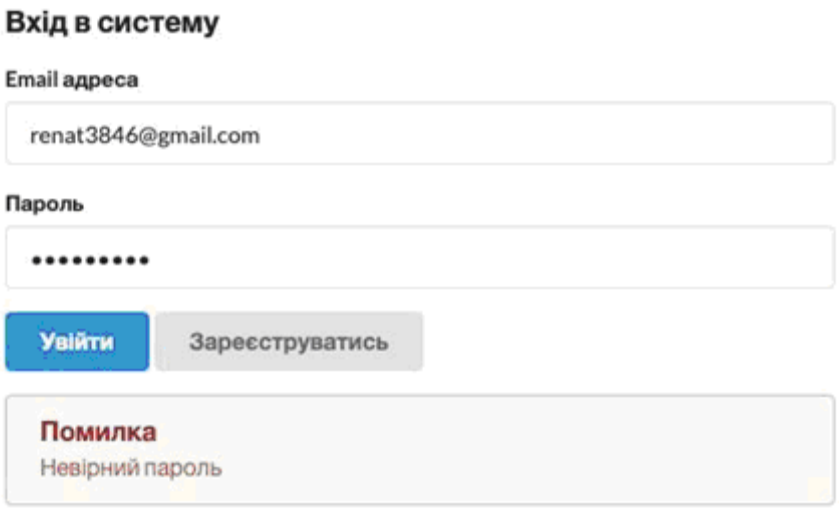


Рис. 3.5. Помилка введеного пароля

Також вікно пропонує користувачеві зареєструватись та заповнити усі необхідні поля для вводу. Усі поля – обов'язкові для введення.

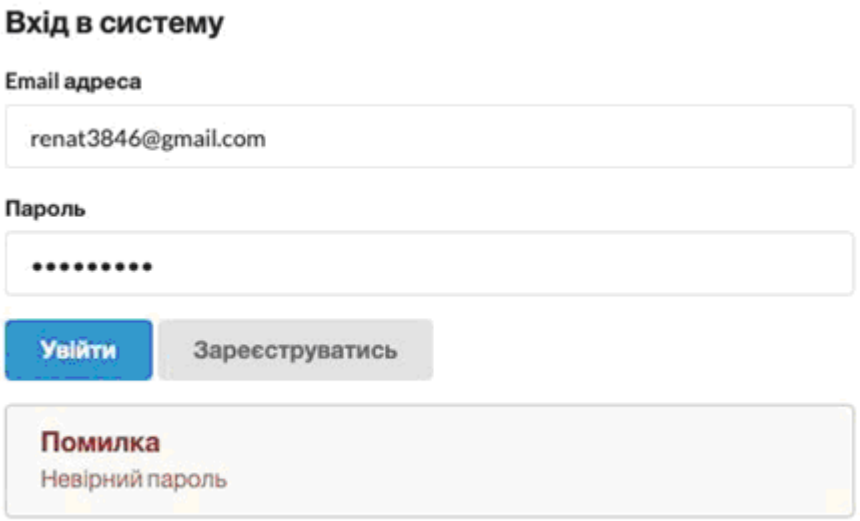


Рис. 3.6. Реєстрація

Після реєстрації, адміністратор повинен підтвердити реєстрацію користувача, та після цього можна переходити до панелі входу до системи.

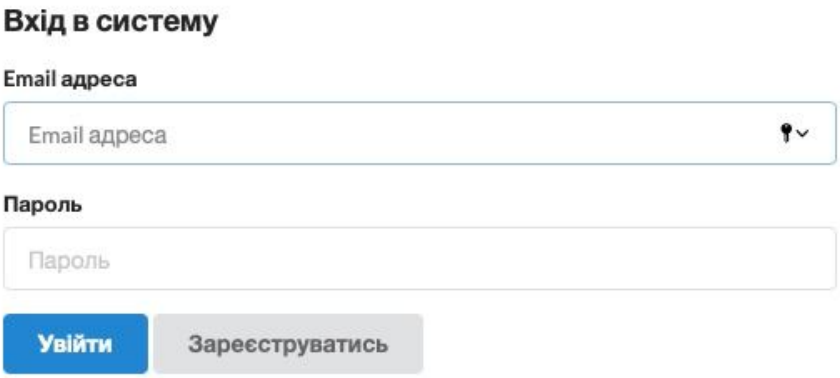


Рис. 3.7. Вхід в систему. Логін не авторізово

Після успішного входу до системи користувач переходить на головну сторінку додатку, на якій відображається блок з інформацією про профіль користувача та показує його рівень доступу (admin, employee). Також користувач вже має змогу працювати з мапою та з панеллю Довідники.

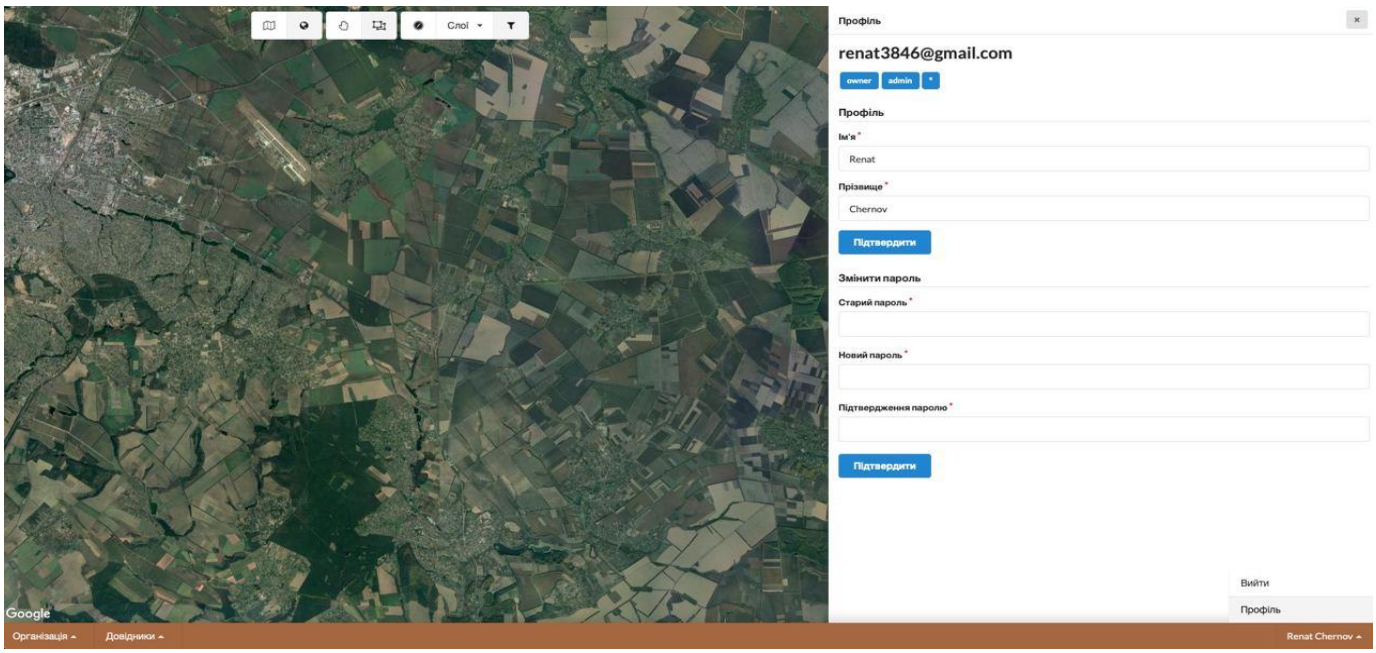


Рис. 3.8. Головна сторінка

На мапі користувач має змогу, за допомогою курсора, виділити полігон по контуру поля. Після чого з’являється панель додання нового поля у базу даних системи. Автоматично рахується площа поля та у майбутньому буде виводитися паї, які зареєстровані у системі. Після введення даних у поля Назва поля та Населений пункт (до якого прилягає) ми додаємо поле до бази даних.

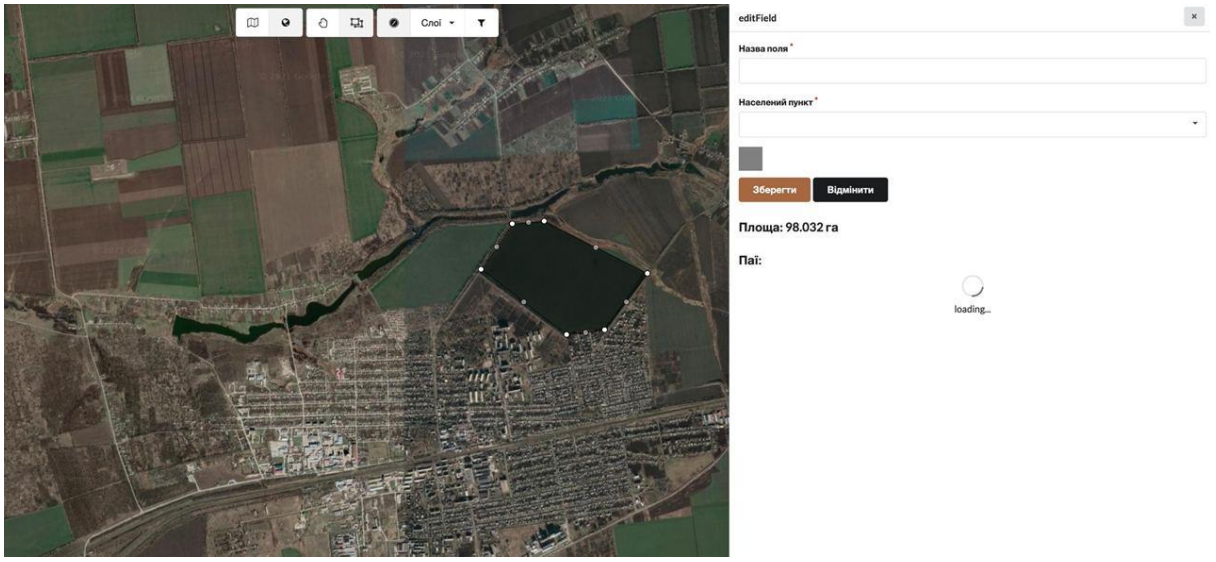


Рис. 3.9. Сторінка editField / супутникова карта

Також можна перемикати вигляд мапи.

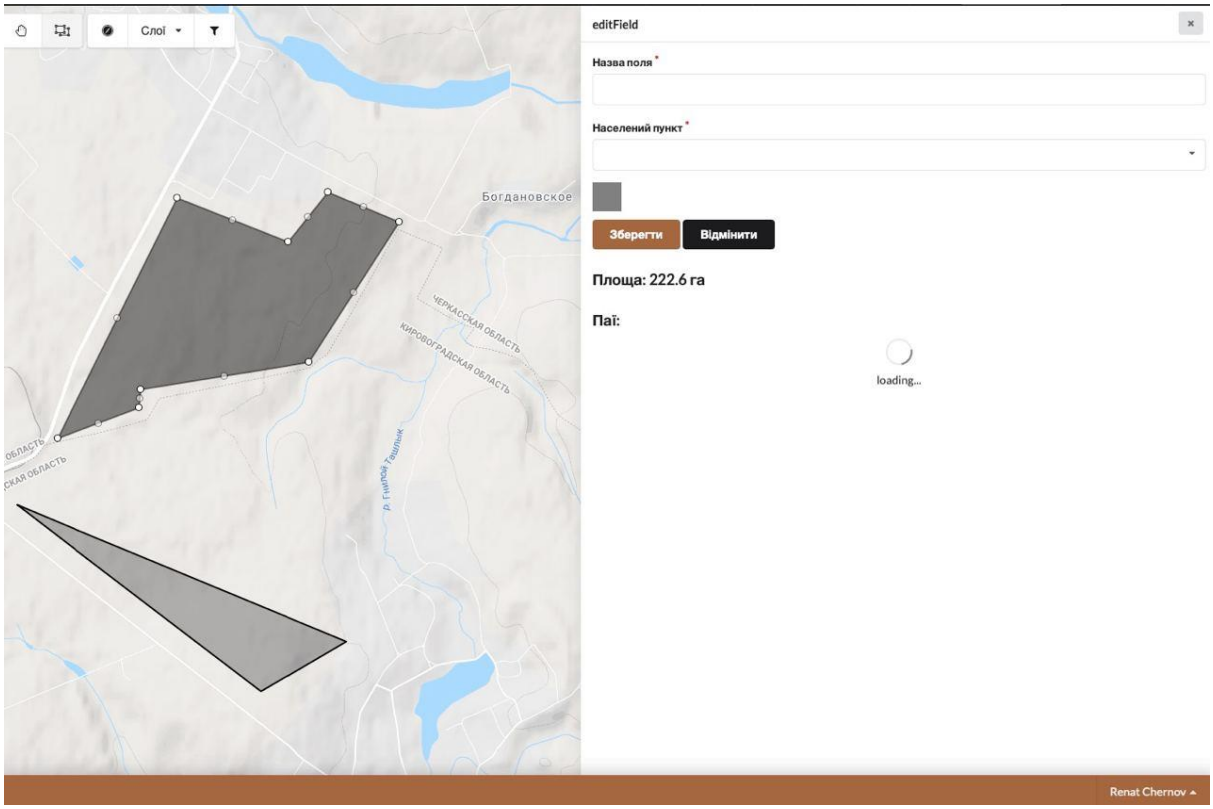


Рис. 3.10. Сторінка editField / карта місцевості

Важливою сторінкою у системі є Сторінка ділянок, у якій можна виконати пошук ділянки, яка вже є у системі, за кадастровим номером земельного паю. А також, маємо змогу створити цю ділянку.

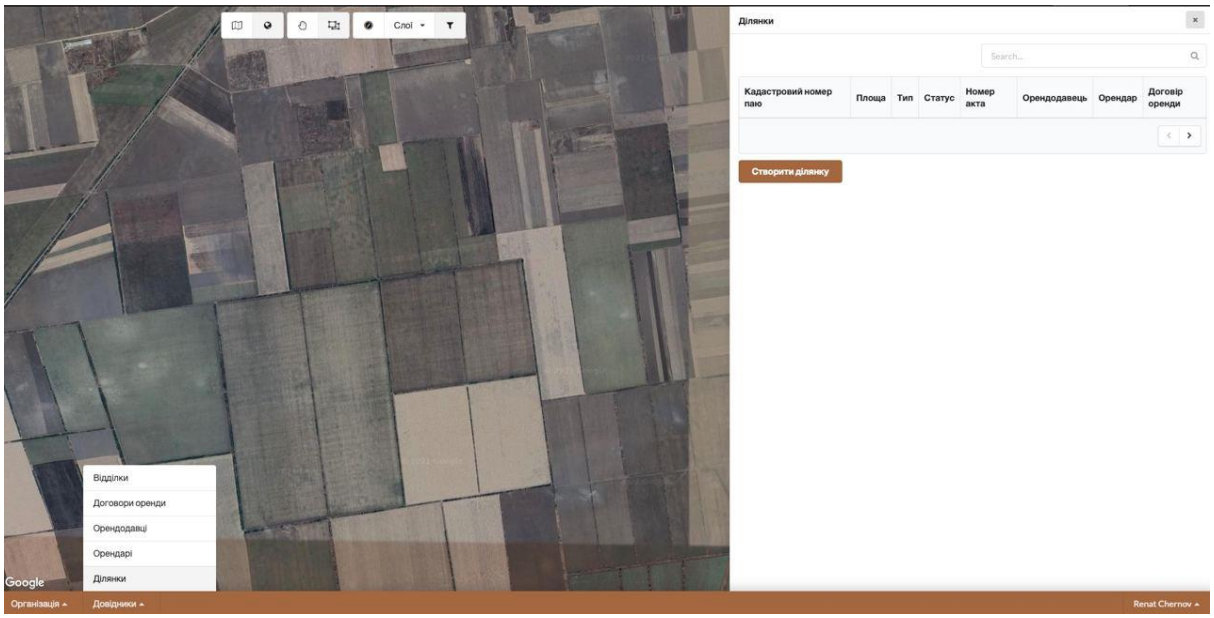


Рис. 3.11. Сторінка Ділянки

При створенні ділянки ми можемо обирати тип ділянки(пай, ділянка, не визначена ділянка), Статус (зареєстрований, на реєстрації) та якщо вони вже присутні в базі, Орендодавець, Орендарі, Договір оренди. Або створити їх, перейшовши на сторінки створення натиснувши кнопку «+».

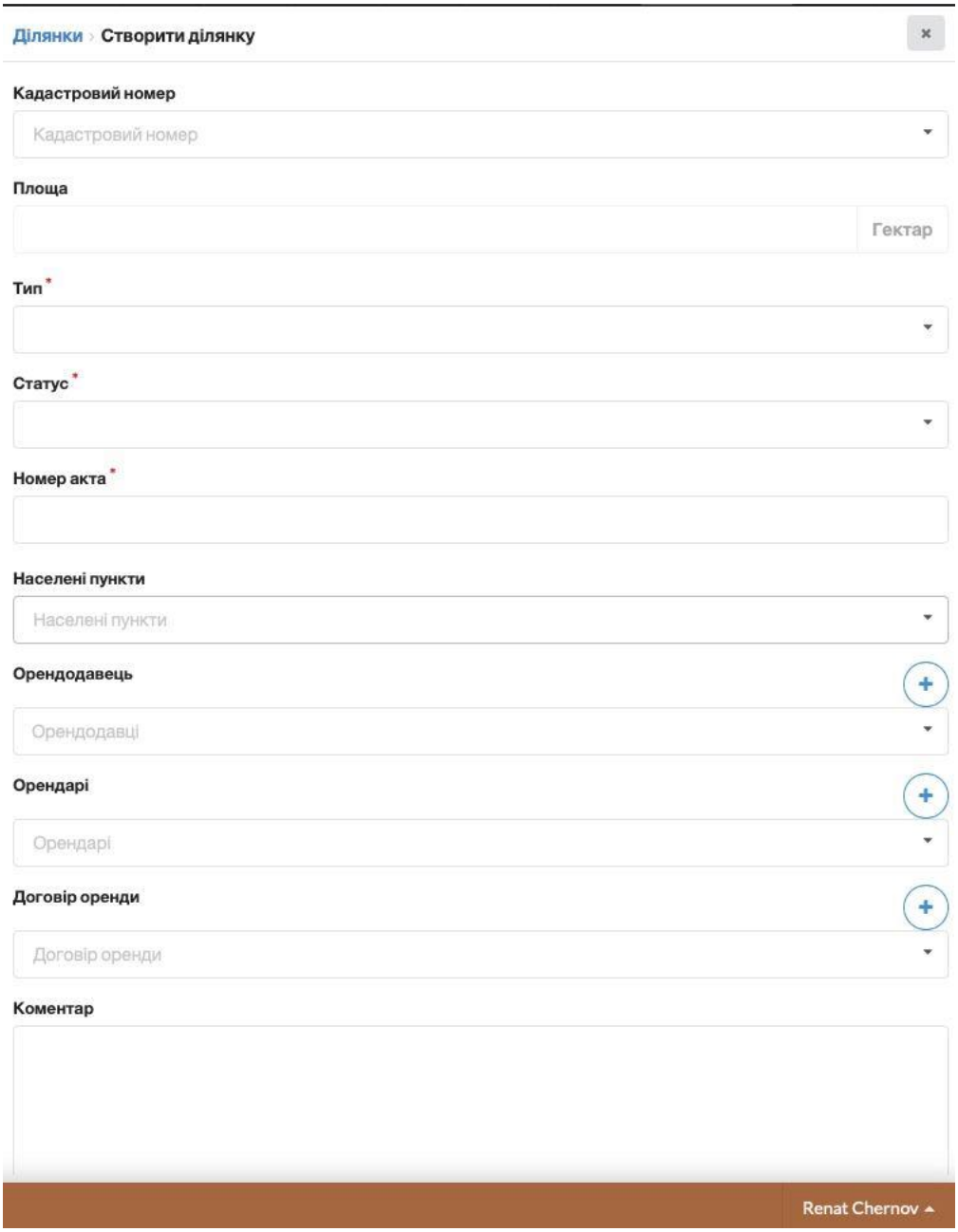


Рис. 3.12. Сторінка Ділянки / Створити ділянку

Під статусом при додаванні орендодавців ми можемо обрати статус (живий / мертвий).

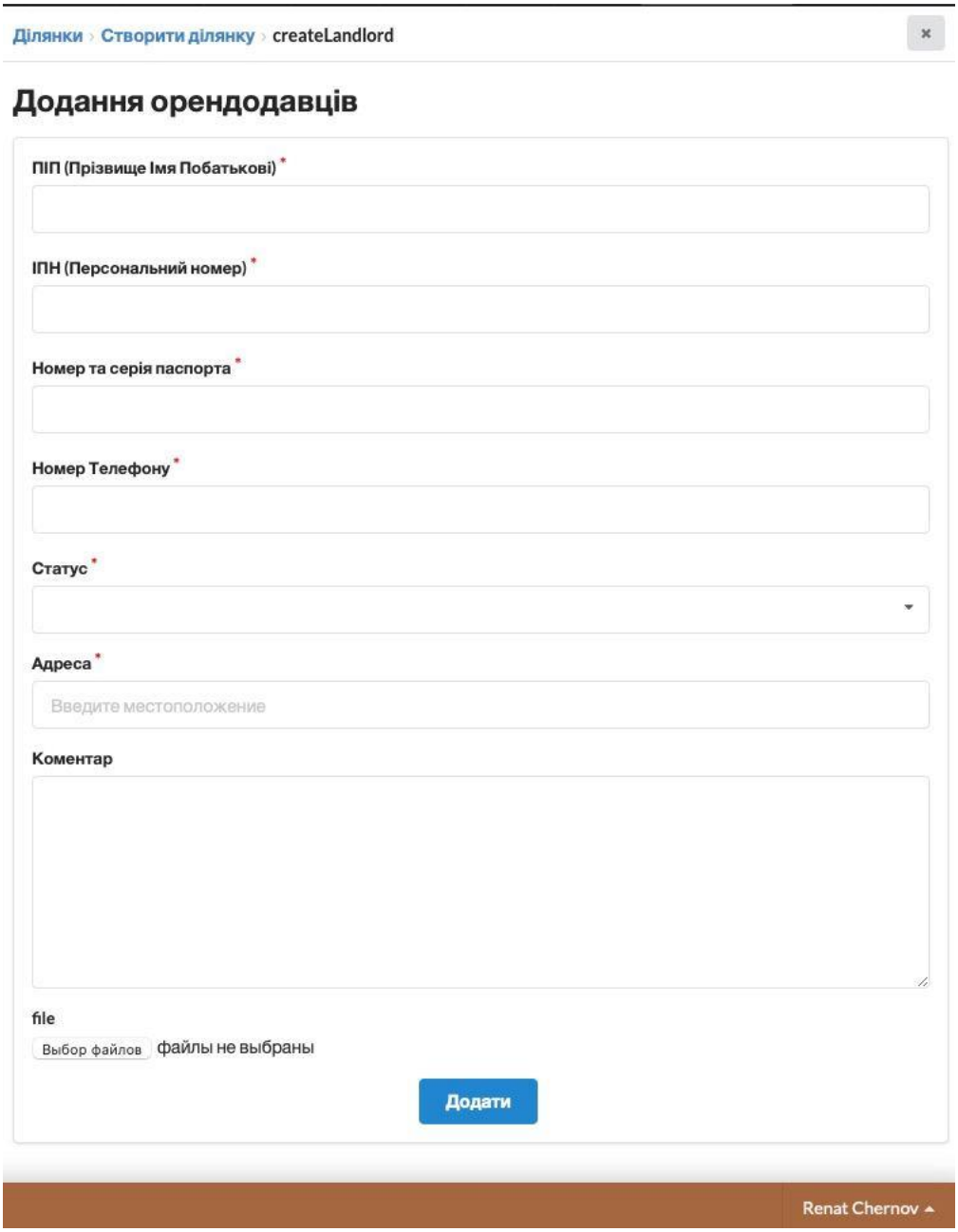


Рис. 3.13. Сторінка Ділянки / createLandlord

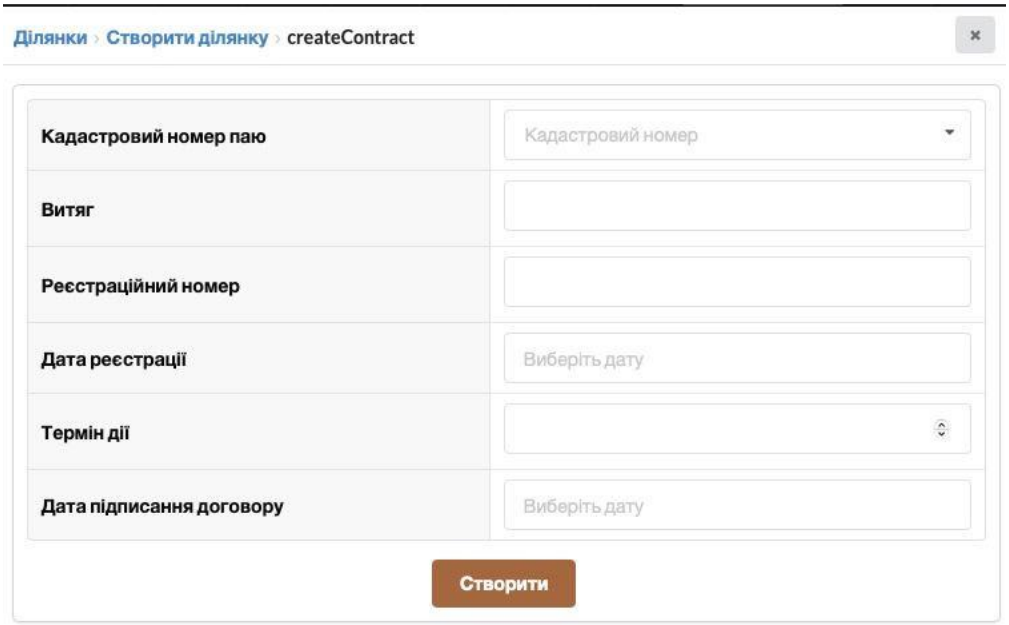


Рис. 3.14. Сторінка Ділянки / createContract

Також є можливість фільтрування списку орендодавців (за статусом живий/мертвий) та пошуку.

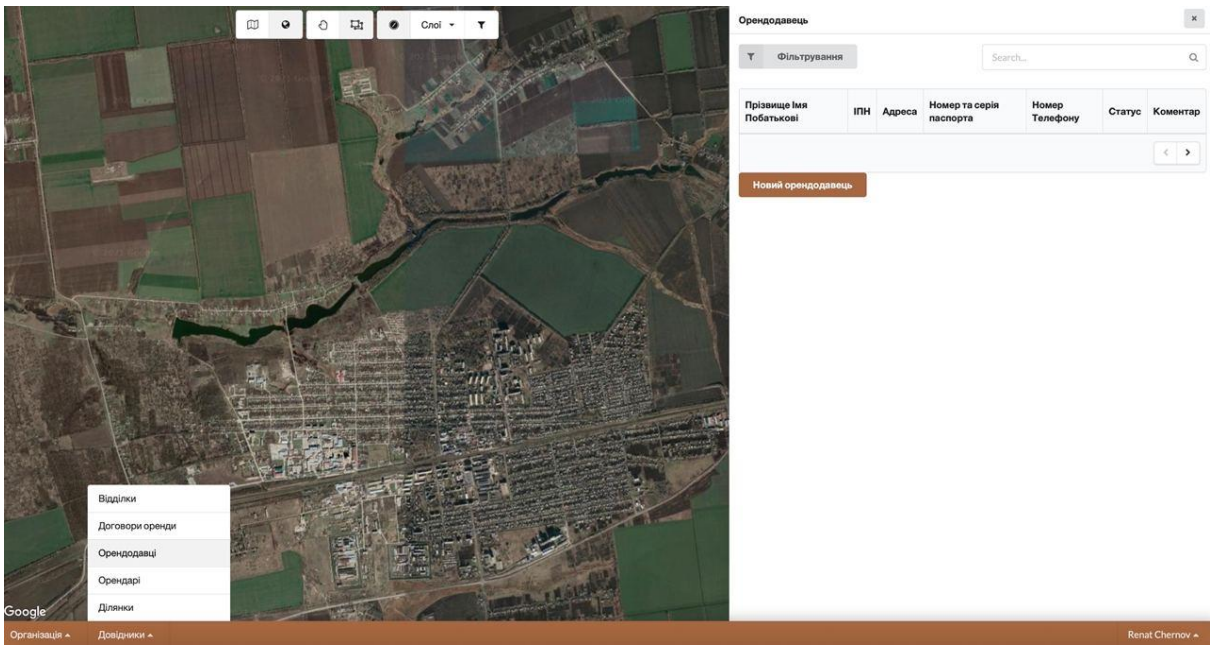


Рис. 3.15. Сторінка Орендодавець

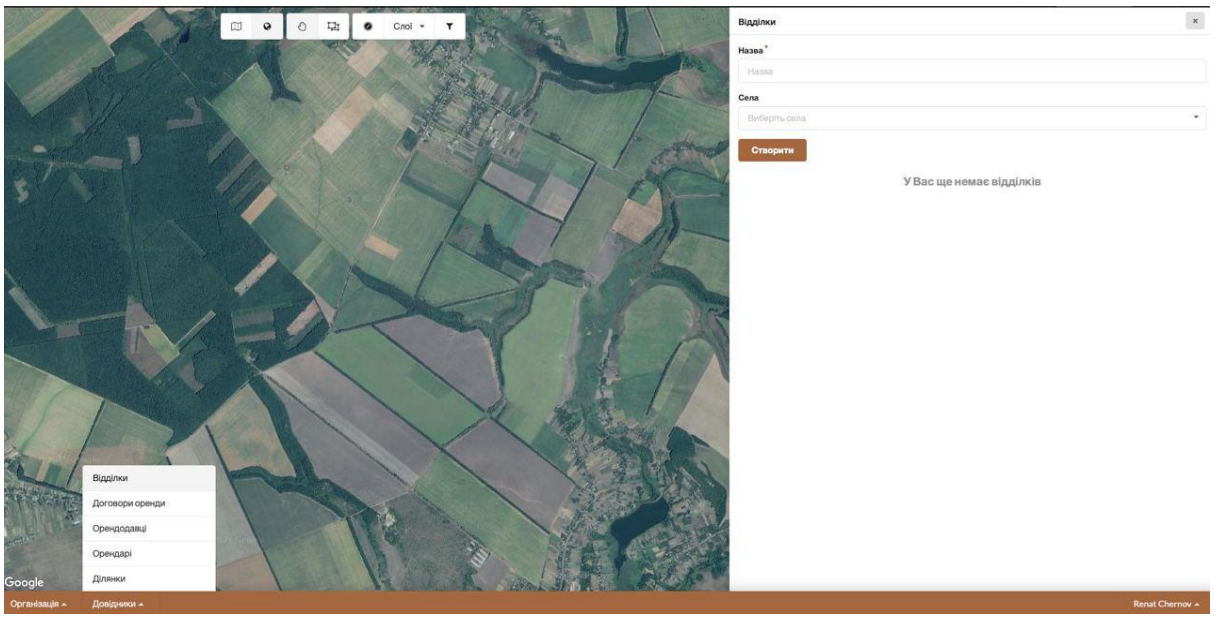


Рис. 3.16. Сторінка Відділки

Сортування можна обрати (без сортування / договори що закінчуються).

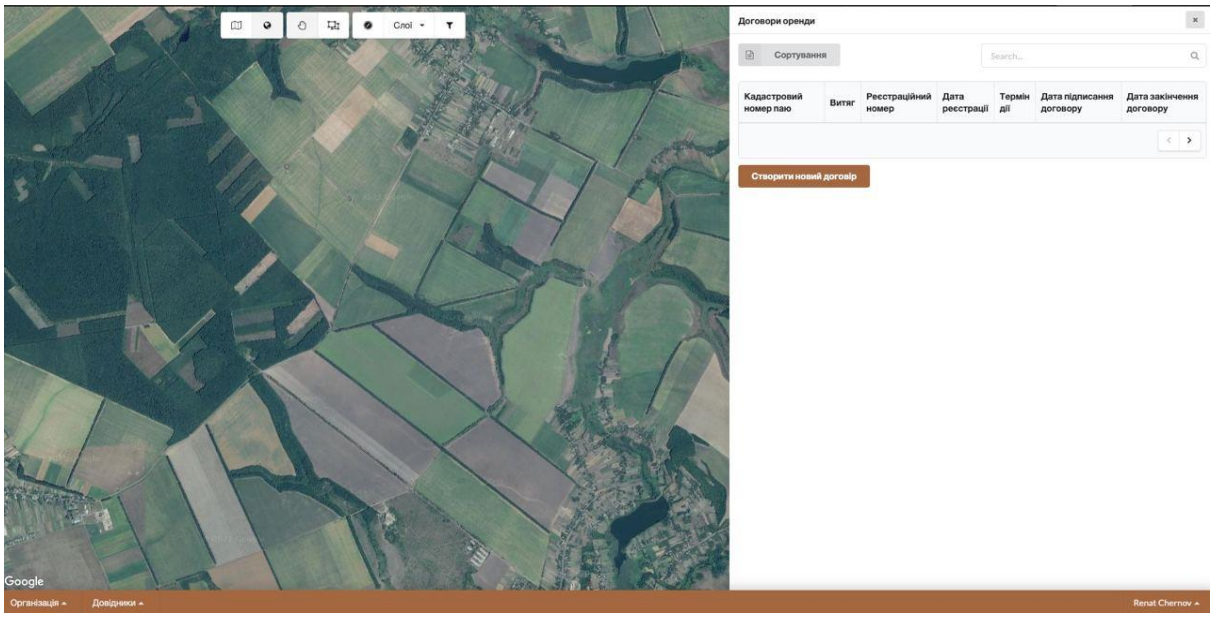


Рис. 3.17. Сторінка Договори оренди

Висновки до розділу

В даній роботі було розроблено систему для обліку та менеджменту полів на сільськогосподарському підприємстві ЛАН.

Система спрощує роботу фермерам та їх бухгалтерам, дозволяє швидко занести в базу потрібну інформацію, відобразити інформацію о стану ділянок, орендарів, орендодавців, контрактів та автоматично генерувати необхідні документи на підставі існуючої інформації.

ВИСНОВКИ

Досліджено можливості застосування систем обліку та маркетингу для аграрного підприємства. Запропонована власна система, що забезпечить невисоку вартість продукту, що являє собою головним фактором, при впровадженні системи, виконання специфічних завдань, зручний інтерфейс, що задовольняє вимогам даної підприємства і, звичайно, виконання програмою її основних функцій.

Вирішено питання автоматизації діяльності фермерів шляхом впровадження системи. До результатів відноситься:

– автоматичний розрахунок обробленої площі поля;

– облік земельного банку, реєстр договорів оренди;

– підвищення оперативності та актуальності інформації;

–скорочення термінів вирішення окремих завдань та прийняття управлінських рішень;

– підвищення якості інформації, її точність, детальність;

– зниження кількості часу, який витрачається на підготовку документів, швидкість видачі вихідних документів;

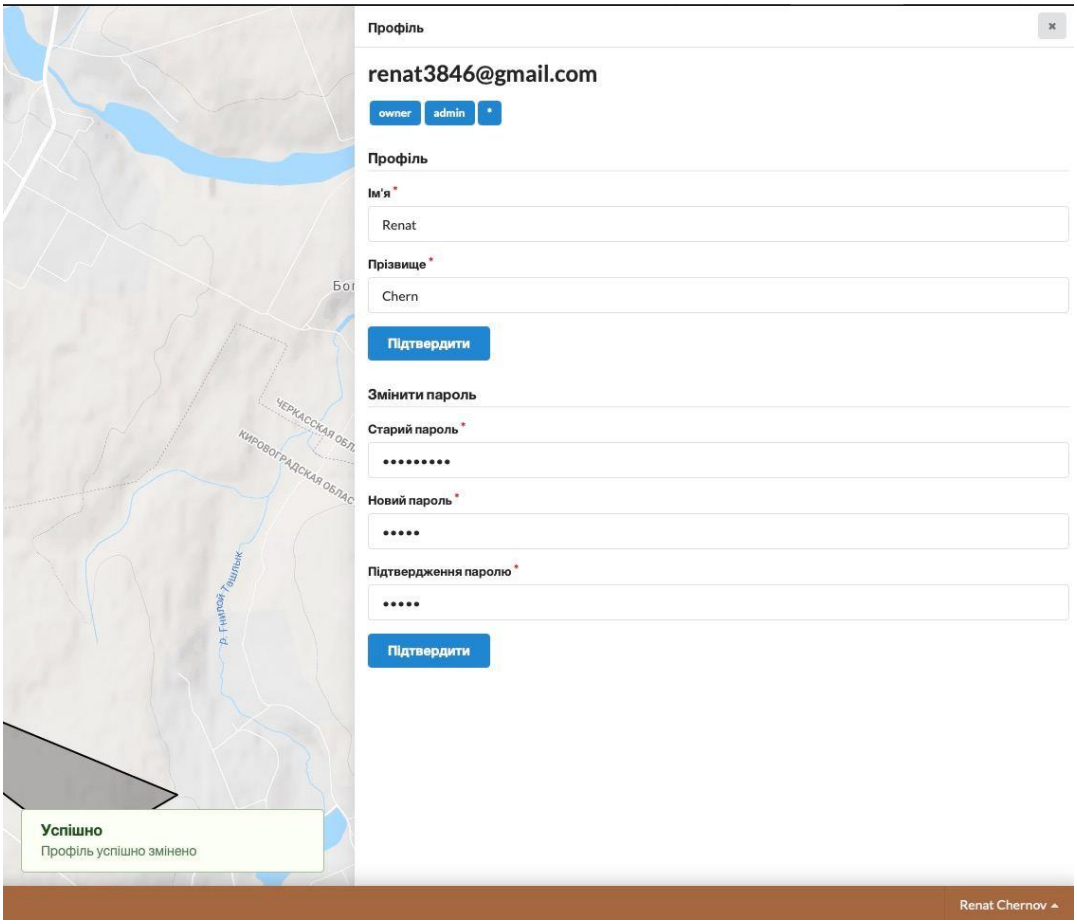
– посилене контролювання, запобігання зловживанню;

– підвищення якості праці за рахунок скорочення рутинних операцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://uhbdp.org/ua/news/innovatsiji-v-apk/
2. https://aggeek.net/ru-blog/5-sposobov-avtomatizirovat-vash-agrobiznes
3. https://www.geomir.ru/publikatsii/elektronnye-karty-poley/
4. http://agroportal.ua/
5. https://kolosok.info
6. https://www.tandfonline.com/doi/pdf/10.1080/17474230802618722
7. https://aggeek.net/ru-blog/5-sposobov-avtomatizirovat-vash-agrobiznes
8. https://latifundist.com/blog/read/

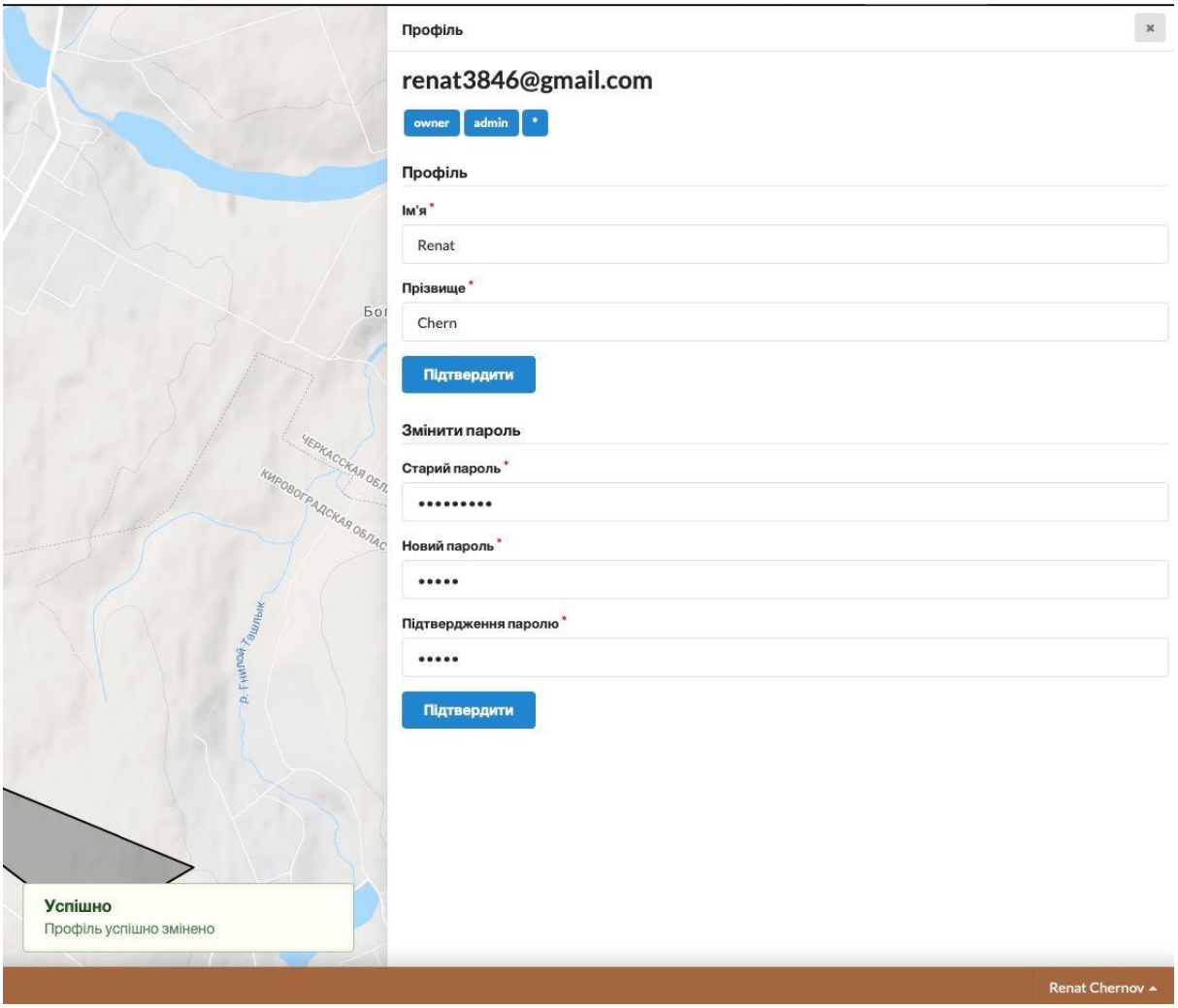
ДОДАТКИ



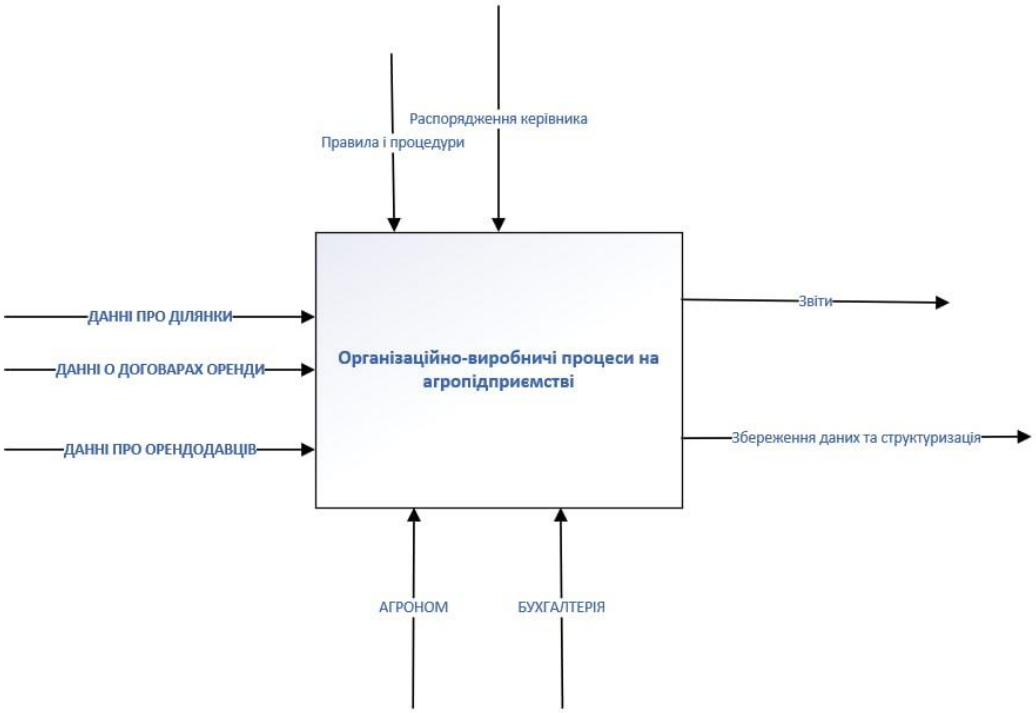
Додаток А.


_id: ObjectId("60957d09fe68eaf8802d9abd")
roles: Array
inviteTokens: Array
username: "qwerty@qwerty.com"
profile: Object
isApproved: true
hash: "16e505e6320e69b3ca48350d71fea7b3d240020d87f967c254c475b4cd74cd11886f8b..."
salt: "c0448d32e7109f84efa615e53d2d876f6444871221aa0f08903a0c380a95d6e8"
__v: 0
_id: ObjectId("60957e15fe68eaf8802d9abe")
roles: Array
inviteTokens: Array
username: "teleginvadim@icloud.com"
profile: Object
isApproved: true
hash: "bae006b1c97b7bb2b913cc5e2f01022beb386c500365755bffff7a4a0eed9ed36fc17a..."
salt: "b7f27e2c6db9ab782ab61338643b33083dd52e6bb2e412f3d2a8fde173535141"
__v: 0
_id: ObjectId("60b57fa8644369eb7a179c3d")
roles: Array
inviteTokens: Array
username: "renat3846@gmail.com"
profile: Object
isApproved: true
hash: "d4694bee3b4707e862408f022e2882de25794c338fcc84232324d966f92d365846575c..."
salt: "253d10a155f89e3b845be857412fa7e32881780ac539cfef1a9c54b0a9f25543"
__v: 0

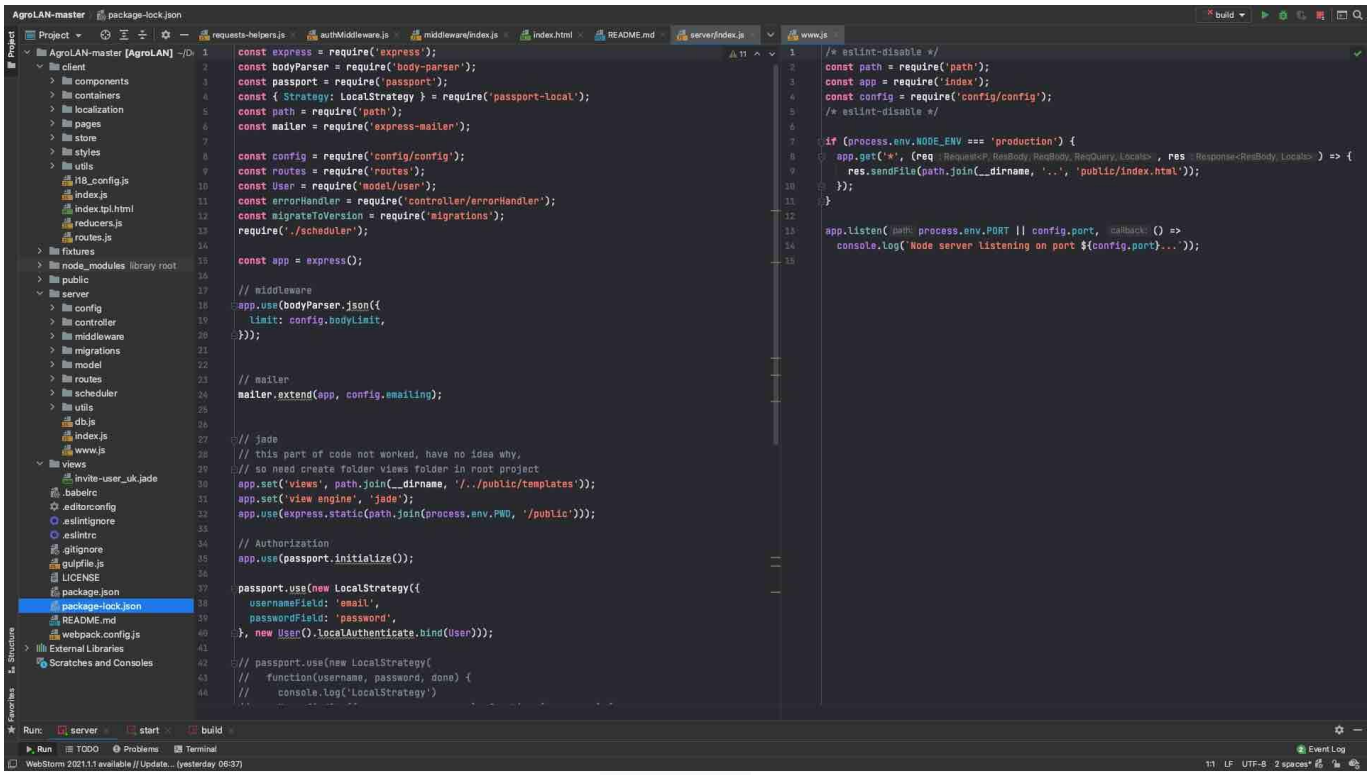
Додаток Б.


Профіль
renat3846@gmail.com
owner
admin
*
Профіль
Ім'я *
Renat
Прізвище *
Chern
Підтвердити
Змінити пароль
Старий пароль *
Новий пароль *
Підтвердження паролю *
Підтвердити
Успішно
Профіль успішно змінено
Renat Chernov

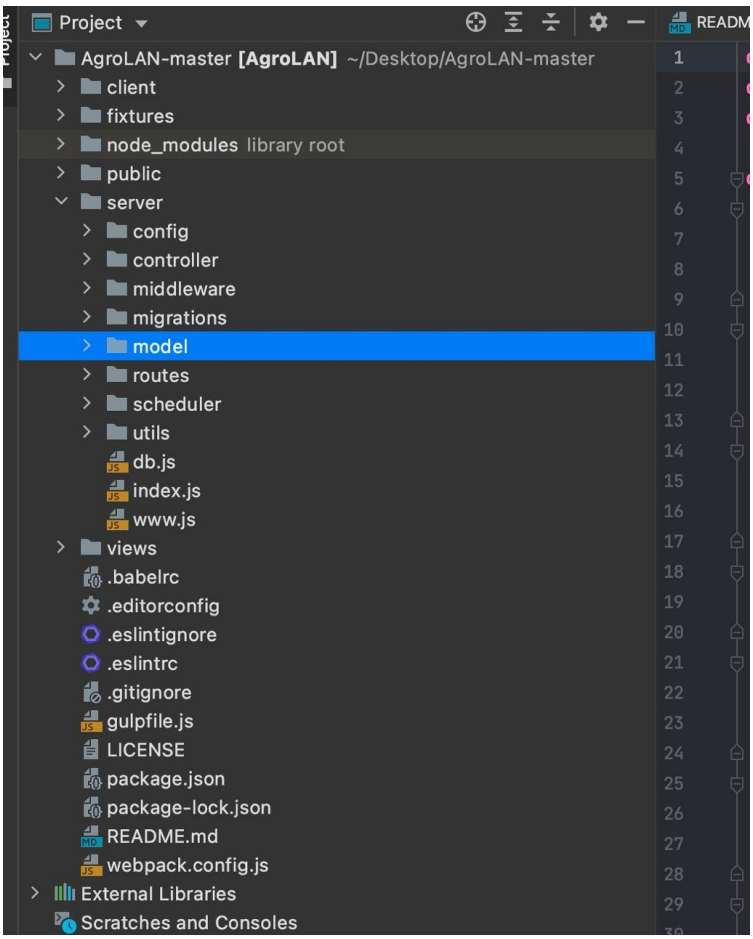
Додаток В


Правила і процедури
Распорядження керівника
ДАННІ ПРО ДІЛЯНКИ
ДАННІ О ДОГОВАРАХ ОРЕНДИ
ДАННІ ПРО ОРЕНДОДАВЦІВ
Організаційно-виробничі процеси на агропідприємстві
Звіти
Збереження даних та структуризація
АГРОНОМ
БУХГАЛТЕРІЯ

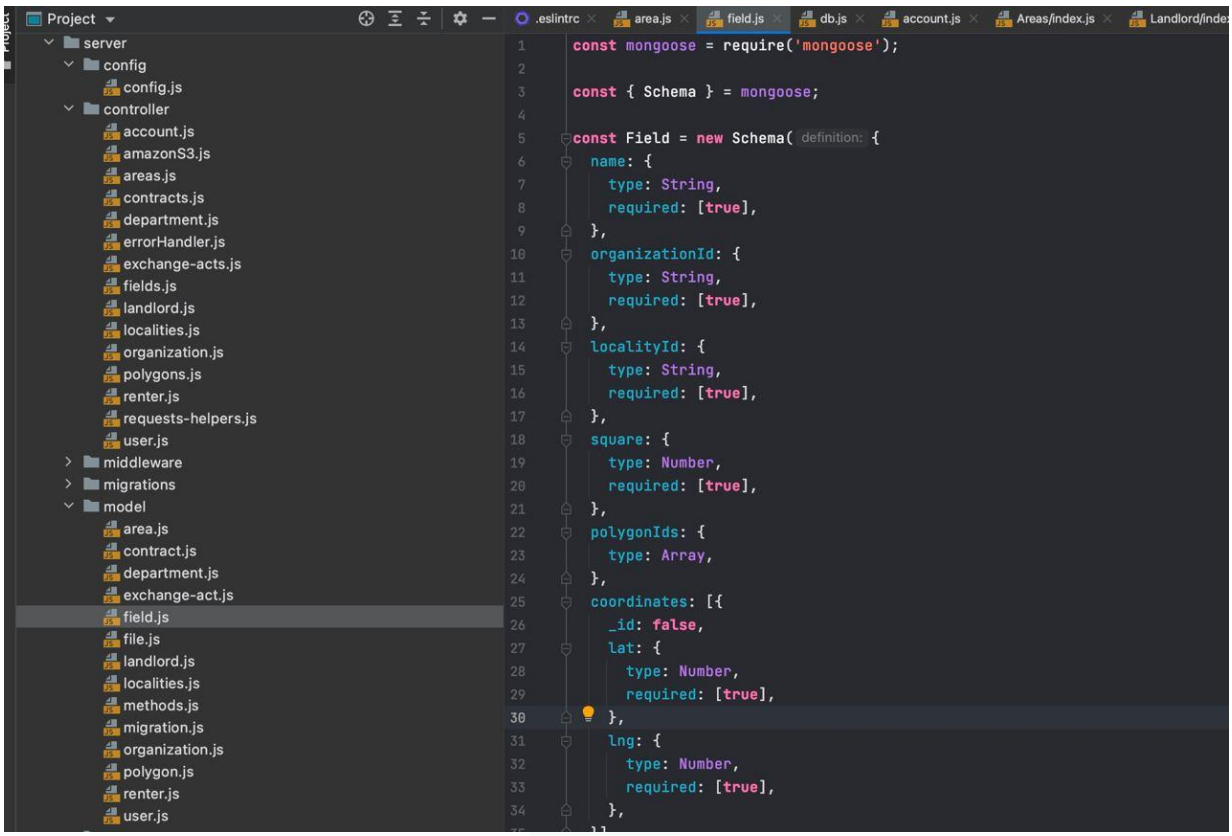
Додаток Д


AgroLAN-master
package-lock.json
Project
requests-helpers.js
authMiddleware.js
middleware/index.js
index.html
README.md
server/index.js
www.js
build
AgroLAN-master [AgroLAN]
client
components
containers
localization
pages
store
styles
utils
i18_config.js
index.js
index.tpl.html
reducers.js
routes.js
fixtures
node_modules library root
public
server
config
controller
middleware
migrations
model
routes
scheduler
utils
db.js
index.js
www.js
views
invite-user_uk.jade
.babelrc
.editorconfig
.eslintignore
.eslintrc
.gitignore
gulpfile.js
LICENSE
package.json
package-lock.json
README.md
webpack.config.js
External Libraries
Scratches and Consoles
const express = require('express');
const bodyParser = require('body-parser');
const passport = require('passport');
const { Strategy: LocalStrategy } = require('passport-local');
const path = require('path');
const mailer = require('express-mailer');
const config = require('config/config');
const routes = require('routes');
const User = require('model/user');
const errorHandler = require('controller/errorHandler');
const migrateToVersion = require('migrations');
require('./scheduler');
const app = express();
// middleware
app.use(bodyParser.json({
limit: config.bodyLimit,
}));
// mailer
mailer.extend(app, config.emailing);
// jade
// this part of code not worked, have no idea why,
// so need create folder views folder in root project
app.set('views', path.join(__dirname, '/../public/templates'));
app.set('view engine', 'jade');
app.use(express.static(path.join(process.env.PWD, '/public')));
// Authorization
app.use(passport.initialize());
passport.use(new LocalStrategy({
usernameField: 'email',
passwordField: 'password',
}, new User().localAuthenticate.bind(User)));
// passport.use(new LocalStrategy(
// function(username, password, done) {
// console.log('LocalStrategy')
/* eslint-disable */
const path = require('path');
const app = require('index');
const config = require('config/config');
/* eslint-disable */
if (process.env.NODE_ENV === 'production') {
app.get('*', (req, res) => {
res.sendFile(path.join(__dirname, '..', 'public/index.html'));
});
}
app.listen(process.env.PORT || config.port, () =>
console.log(`Node server listening on port ${config.port}...`));
Run:
server
start
build
Run
TODO
Problems
Terminal
Event Log
WebStorm 2021.1.1 available // Update... (yesterday 06:37)
1:1
LF
UTF-8
2 spaces*

Додаток Ґ


Project
AgroLAN-master [AgroLAN] ~/Desktop/AgroLAN-master
client
fixtures
node_modules library root
public
server
config
controller
middleware
migrations
model
routes
scheduler
utils
db.js
index.js
www.js
views
.babelrc
.editorconfig
.eslintignore
.eslintrc
.gitignore
gulpfile.js
LICENSE
package.json
package-lock.json
README.md
webpack.config.js
External Libraries
Scratches and Consoles

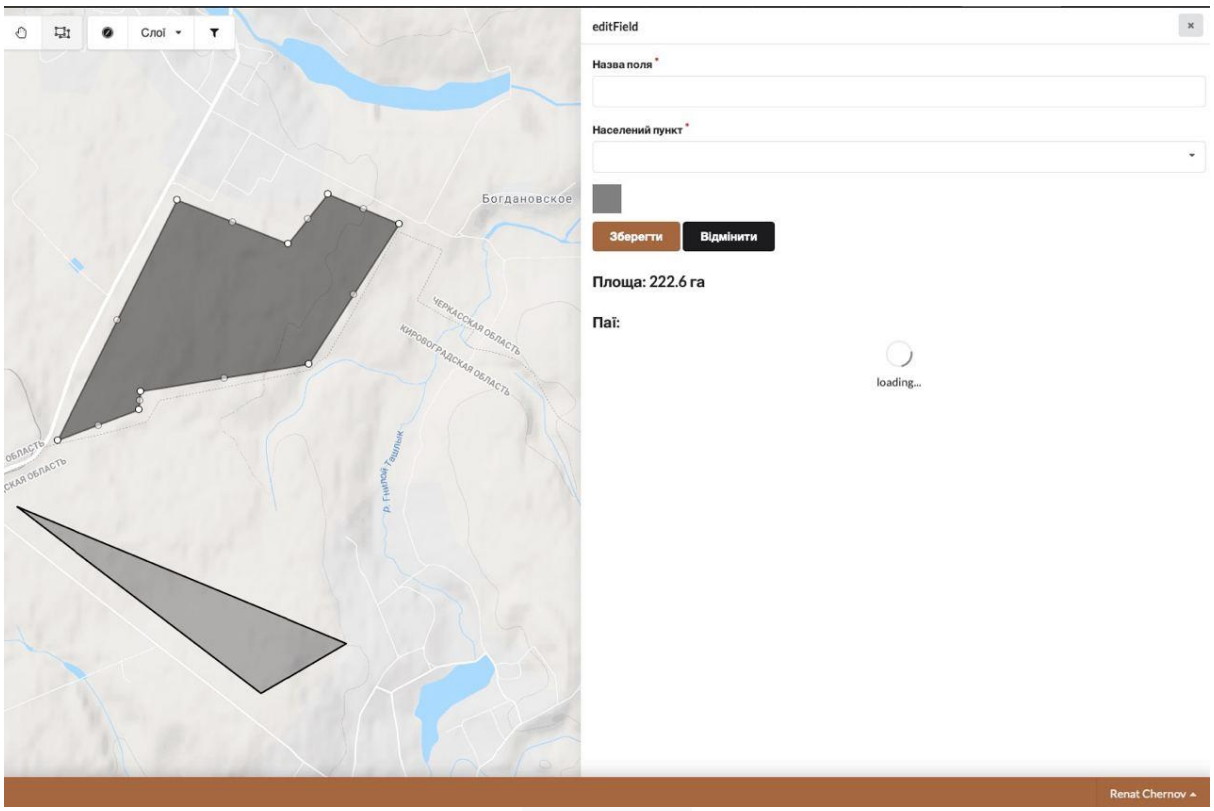
Додаток Д


Project
server
config
config.js
controller
account.js
amazonS3.js
areas.js
contracts.js
department.js
errorHandler.js
exchange-acts.js
fields.js
landlord.js
localities.js
organization.js
polygons.js
renter.js
requests-helpers.js
user.js
middleware
migrations
model
area.js
contract.js
department.js
exchange-act.js
field.js
file.js
landlord.js
localities.js
methods.js
migration.js
organization.js
polygon.js
renter.js
user.js
.eslintrc
area.js
field.js
db.js
account.js
Areas/index.js
const mongoose = require('mongoose');
const { Schema } = mongoose;
const Field = new Schema(definition: {
name: {
type: String,
required: [true],
},
organizationId: {
type: String,
required: [true],
},
localityId: {
type: String,
required: [true],
},
square: {
type: Number,
required: [true],
},
polygonIds: {
type: Array,
},
coordinates: [{
_id: false,
lat: {
type: Number,
required: [true],
},
lng: {
type: Number,
required: [true],
},

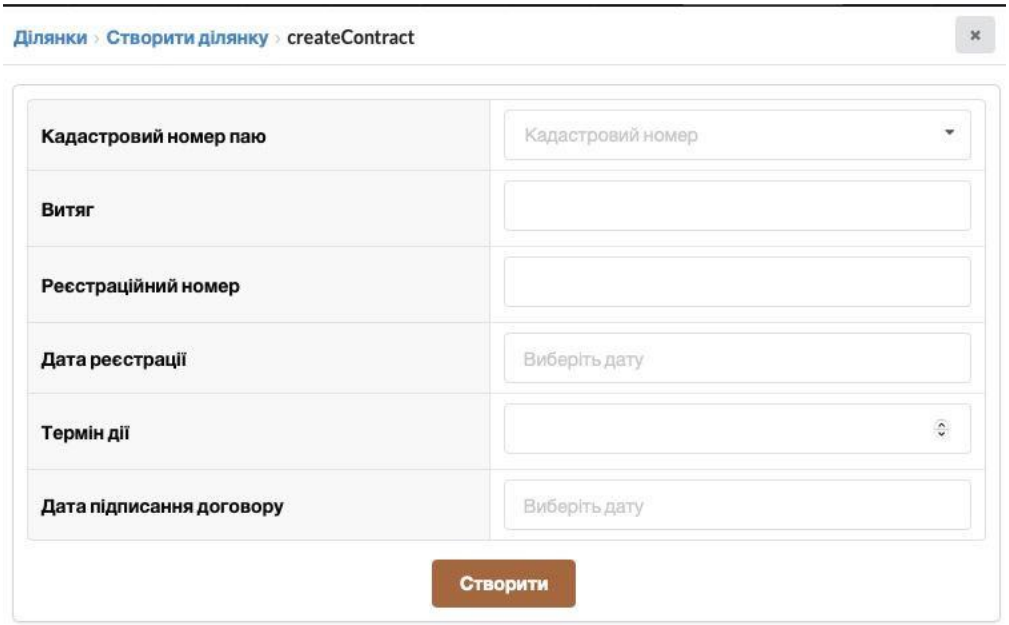
Додаток Е

```jsx
import React, { PropTypes } from 'react';
import { Button, Form, Container, Message } from 'semantic-ui-react';
import { Translate, I18n } from 'react-i18nify';
import FormWarning from '../General/FormWarning';
import './style.scss';

const Login = ({ onLogin, onRegisterButtonClick, formState }) =>
  <div>
    <Container className="sign-in-form">
      <div className="sign-in-form">
        <h3><Translate value="authorization" /></h3>
        <Form onSubmit={onLogin} loading={formState.loading} error={!!formState.error}>
          <Form.Field>
            <label><Translate value="email_address" /></label>
            <Form.Input name="email" placeholder={I18n.t('email_address')} />
            <FormWarning isVisible={formState.warning.name === 'email'} />
          </Form.Field>
          <Form.Field>
            <label><Translate value="password" /></label>
            <Form.Input name="password" placeholder={I18n.t('password')} type="password" />
            <FormWarning isVisible={formState.warning.name === 'password'} />
          </Form.Field>

          <Button primary type="submit">
            <Translate value="log_in" />
          </Button>
          <Button onClick={onRegisterButtonClick}>
            <Translate value="register" />
          </Button>

          <Message
            error
            header={I18n.t('error')}
            content={formState.error}
          />
        </Form>
      </div>
    </Container>
  </div>;
```

Додаток Ж


Слої
Богдановское
ЧЕРКАССКАЯ ОБЛАСТЬ
КИРОВОГРАДСКАЯ ОБЛАСТЬ
editField
Назва поля *
Населений пункт *
Зберегти
Відмінити
Площа: 222.6 га
Паї:
loading...
Renat Chernov

Додаток З


Ділянки › Створити ділянку › createContract
Кадастровий номер паю
Кадастровий номер
Витяг
Реєстраційний номер
Дата реєстрації
Виберіть дату
Термін дії
Дата підписання договору
Виберіть дату
Створити

Додаток З