

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій

(повне найменування факультету)

Кафедра комп'ютерних систем та технологій

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

**«Розробка апаратно-програмного комплексу для системи
дистанційного бездротового керування динамічними зображеннями»**

(тема кваліфікаційної роботи українською мовою)

**«Development of a complex unit for a remote wireless controlling system
for dynamic images»**

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач денної форми навчання
спеціальності 123 – Комп'ютерна інженерія

(код, назва спеціальності)

Освітня програма «Комп'ютерна інженерія»

(назва)

Боровик Артем Ігорович

(прізвище, ім'я, по-батькові здобувача)

Керівник к. ф.-м. н. Чепок А.О.

(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент проф. Приходько С.Б.

(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри

№ від . . 20 р.

Завідувач(ка) кафедри

(підпис)

Гунченко Ю.

(прізвище, ім'я)

Захищено на засіданні ЕК №
протокол № від . . 20 р.

Оцінка / /
(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК

(підпис)

(прізвище, ім'я)

Одеса 2024

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

Кафедра комп'ютерних систем та технологій

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 123 – Комп'ютерна інженерія

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

проф. Юрій ГУНЧЕНКО

“ ”

2024 року

З А В Д А Н Н Я
на кваліфікаційну роботу студенту

Боровик Артема Ігоровича

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи

Розробка апаратно-програмного комплексу для системи дистанційного бездротового керування динамічними зображеннями

керівник кваліфікаційної роботи

к. ф.-м. н. Чепок А.О.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи 7 червня 2024 р.

3. Вихідні дані роботи

Літературні джерела відповідні до тематики роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ, 1. Теоретична Частина, 2. Постановка Задачі, 2.1 Задачі Застосунка, 3. Вибір Програмного Забезпечення, 4. Проектування Програмного Застосунку, 5. Інформаційне Моделювання, 6. Програмна Реалізація Інтерфейсу, 7. Створення Серверу, 8. Принцип Роботи Застосунку, 9. Інструкція Користувача, 9.1 Програмний Інтерфейс, 9.2 Серверний Інтерфейс, 10. Перспективи, Висновок, Список Використаних Джерел, Додаток А, Додаток Б.

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

1. Актуальність, 2.Авторський застосунок-Навіщо?, 3. Чому саме цей застосунок?, 4.Аналоги до авторського застосунка, 5. Ціль створення застосунка, 6.Зміст завдання, 7. Висновок.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Проведення аналітичних тестів	к. ф.-м. н. Чепок А.О.		

7. Дата видачі завдання 25 січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Ознайомлення з завданням до кваліфікаційної роботи	25.01.2024	<i>Виконано</i>
2	Підбір та аналіз джерел по темі роботи	26.01.2024 – 15.02.2024	<i>Виконано</i>
3	Розробка та затвердження плану дипломного проекту	16.02.2024 – 09.03.2024	<i>Виконано</i>
4	Аналітичний огляд і постановка задачі	10.03.2024 – 21.03.2024	<i>Виконано</i>
5	Розробка бази даних застосунку	22.03.2024 – 30.03.2024	<i>Виконано</i>
6	Розробка серверної частини застосунку	31.03.2024 – 15.05.2024	<i>Виконано</i>
7	Розробка клієнтського інтерфейсу	16.05.2024 – 21.05.2024	<i>Виконано</i>
8	Оформлення кваліфікаційної роботи	22.05.2024 – 02.06.2024	<i>Виконано</i>
9	Перевірка на плагіат		

Студент _____ (ім'я, прізвище)

Керівник проекту (роботи) _____ (ім'я, прізвище)

АНОТАЦІЯ

Мета даної дипломної роботи полягає у проєктуванні та реалізації набору інструментів для динамічної зміни зображень. Розроблений застосунок створений як для кінцевих користувачів, так і для адміністраторів, надаючи їм широкі можливості для редагування зображень у реальному часі.

Реалізація програмного застосунку виконана за допомогою мови програмування Java у зв'язці з фреймворком JavaFX та інструментарієм Minecraft Development. Такий вибір технологій забезпечує високу продуктивність, стабільність та зручність у розробці. Архітектура системи відповідає шаблону MVC (Model-View-Controller) та використовує дворівневу архітектуру, що дозволяє ефективно розподіляти функціональні обов'язки між компонентами системи.

Розроблений програмний застосунок дозволяє редагувати зображення, використовуючи певні вбудовані модулі, замість внутрішнього програмного редагування. Це забезпечує більшу гнучкість і розширюваність, оскільки користувачі можуть додавати нові модулі або замінювати існуючі, не змінюючи основного коду програми. Крім того, це значно знижує навантаження на систему та покращує загальну продуктивність.

В серверній частині системи реалізовано захист від несанкціонованого доступу до серверної консолі. Це забезпечує високий рівень безпеки, захищаючи конфіденційні дані та запобігаючи можливим атакам. Для цього використовуються сучасні методи аутентифікації та авторизації, що гарантують надійний захист системи.

Результатом дипломної роботи є потужний застосунок з можливістю дистанційного бездротового редагування динамічних зображень.

ANNOTATION

The purpose of this diploma work is to design and implement a set of tools for dynamic image editing. The developed application is designed for end-users as well as administrators, providing them with extensive capabilities for real-time image editing.

The implementation of the software application is done using the Java programming language in conjunction with the JavaFX framework and the Minecraft Development toolkit. Such technology choices ensure high productivity, stability, and convenience in development. The system architecture follows the MVC (Model-View-Controller) pattern and employs a two-tier architecture, allowing for efficient distribution of functional responsibilities among system components.

The developed software application allows for image editing using specific built-in modules instead of internal programmatic editing. This provides greater flexibility and extensibility since users can add new modules or replace existing ones without altering the core code of the program. Additionally, this significantly reduces system load and enhances overall performance.

In the server-side part of the system, protection against unauthorized access to the server console is implemented. This ensures a high level of security by safeguarding confidential data and preventing potential attacks. Modern authentication and authorization methods are employed to guarantee reliable system protection.

The result of the diploma work is a powerful application capable of remotely editing dynamic images wirelessly.

ЗМІСТ

ВСТУП	7
1 ТЕОРЕТИЧНА ЧАСТИНА	8
2. ПОСТАНОВКА ЗАДАЧІ.....	13
2.1 Задачі застосунка.....	13
3 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	14
4. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСТАСУНКУ	20
5 ІНФОРМАЦІЙНЕ МОДЕЛЮВАННЯ.....	24
6 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ	26
7 СТВОРЕННЯ СЕРВЕРУ.....	27
8. ПРИНЦИП РОБОТИ ЗАСТАСУНКУ	38
9. ІНСТРУКЦІЯ КОРИСТУВАЧА	40
9.1 Програмний інтерфейс	40
9.2 Серверний інтерфейс	43
10. ПЕРСПЕКТИВИ.....	45
ВИСНОВОК.....	46
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	48
ДОДАТОК А Створення візуального інтерфейсу застосунка.....	49
ДОДАТОК Б Функції та змінні програмної реалізації застосунку	51

ВСТУП

Велику частину сучасного світу займає обробка зображень. Це одна з ІТ-галузей, що найбільш динамічно розвивається, яка привертає мільйони людей по всьому світу: обробка зображень – від звичайної зміни кольорового спектру до стилю самого зображення – є дуже важливою частиною культури та технології сучасного суспільства.

Цей проект спрямований на створення програми яка надає можливість зміну всіх характеристик пікселей в зображенні, що обробляється в реальному часі. По суті, метою даної роботи є створення авторської програми для зручного керування/модифікації певного «потокowego» зображенням.

Для досягнення цієї мети передбачено вирішення ряду важливих завдань:

- виявити, що саме можна змінювати в зображенні;
- виявити, як саме можна змінювати зображення;
- обрання архітектури та шаблону проектування для застосунка;
- вибір технологій та засобів розробки застосунка;
- розроблення зовнішнього інтерфейсу для користувачів;
- забезпечення безпеки серверної частини застосунка;
- тестування та виправлення помилок отриманих в результаті тестування.

1 ТЕОРЕТИЧНА ЧАСТИНА

Обробка зображення – це процес маніпуляції зображеннями за допомогою комп'ютерних алгоритмів для покращення їх якості, виділення певних особливостей або витягання корисної інформації. Цей процес охоплює широкий спектр технік, таких як фільтрація, сегментація, трансформація, реконструкція та інші методи аналізу зображень.

Основні етапи обробки зображення включають:

- отримання зображення з камери, сканера або файлу;
- очищення зображення від шумів, корекція контрасту, усунення дефектів тощо;
- використання фільтрів для підвищення якості зображення, наприклад, згладжування або підсилення країв;
- виділення окремих об'єктів або областей на зображенні;
- вимірювання характеристик, таких як площа, периметр, форма об'єктів;
- ідентифікація та класифікація об'єктів на зображенні;
- відображення отриманих результатів у зручному для користувача вигляді.

Обробка зображень має численні застосування в різних галузях, починаючи з медичної (обробка медичних зображень), астрономічної, промислово автоматичної закінчивши художньої.

Згадуючи обробку зображень треба ще згадати поняття графіка. Якою вона буває та в чому різниця між різними типами графіки.

Графіка — це спосіб створення, зберігання та обробки зображень за допомогою комп'ютерних технологій. Існує декілька видів графіки, які можна класифікувати за різними ознаками:

1) Растрова (піксельна) графіка представляє зображення як сукупність пікселів, де кожен піксель має свій колір. Це найбільш поширений вид графіки для фотографій та складних зображень. Основною перевагою растрової

графіки є висока деталізація та реалістичне відображення. Однак є втрати якості при масштабуванні та розмір файлу займає більше місця ніж інші.

2) Векторна графіка використовує математичні формули для опису зображень у вигляді ліній, кривих і багатокутників. Векторні зображення можуть бути масштабовані без втрати якості. Основною перевагою є можливість без втрати якості масштабування, менший розмір файлу для простих зображень. А недоліками векторної графіки є не підходимість для складних зображень з багатьма дрібними деталями.

3) Фрактальна графіка базується на математичних фракталах — самоподібних структурах, які повторюються на різних масштабах. Використовується для створення складних природних форм, таких як ландшафти або текстури. Перевагою такого типу графіки є висока деталізація при збереженні малого розміру файлу. Але такий тип графіки обмеж по застосування та важко створити.

4) Тривимірна графіка створює зображення в тривимірному просторі за допомогою моделей, які мають ширину, висоту та глибину. Використовується для створення реалістичних зображень і анімацій. Зображення такого типу реалістично відображає об'єкта також є можливість додати анімацію та візуалізацію з різних ракурсів. Але такі зображення складно створювати та не кожен комп'ютер зможе їх обробити.

5) Комп'ютерна анімація включає як 2D, так і 3D анімацію, де створюються рухомі зображення за допомогою комп'ютерної графіки. Перевагою є можливість створення реалістичних або фантастичних рухомих зображень. Недоліком є потреба в значних обчислювальних ресурсах, складність процесу.

Майже для кожної з цих галузей є свій застосунок для обробки зображення. Кожен з цих застосунків мають десь схожий набір інструментів, але всі вони мають унікальні можливості. Основними застосунками для той чи іншої галузі:

- **Adobe Photoshop:** редактор зображень з безліччю функцій для редагування, ретуші, корекції кольору та створення композицій. Підтримує плагіни та скрипти для розширення функціональності.

- **Adobe Lightroom:** спеціалізований інструмент для професійних фотографів для організації, обробки та редагування великих колекцій фотографій. Включає інструменти для корекції кольору, експозиції та інших параметрів.

- **GIMP:** доволі гнучкий редактор зображень з відкритим кодом. Підтримує безліч форматів файлів та має розширювану архітектуру за допомогою плагінів та скриптів.

- **MATLAB:** потужна платформа для обробки зображень, часто використовується в наукових дослідженнях та інженерії. Включає Image Processing Toolbox з багатьма алгоритмами та функціями для аналізу та обробки зображень.

- **OpenCV (Open Source Computer Vision Library)** Відкрита бібліотека для комп'ютерного зору та обробки зображень. Підтримує C++, Python та інші мови програмування, включає багато алгоритмів для обробки зображень та машинного навчання.

- **RawTherapee:** інструмент для обробки RAW-файлів з великою кількістю налаштувань для покращення якості зображень. Підтримує багато моделей фото- і відео-камер та форматів файлів.

Основні елементи обробки зображення включають різні методи та техніки, які дозволяють покращити якість зображення, виділити важливі деталі або витягти корисну інформацію. Ось основні елементи обробки зображення:

1) Попередня обробка зображення включає методи, які підготовлюють зображення для подальшого аналізу, покращують його якість та усувають небажані артефакти. Ось декілька таких прикладів:

а) Фільтрація шуму: Використання фільтрів для зменшення шуму.

b) Корекція контрасту: Регулювання яскравості та контрасту для покращення видимості деталей.

c) Нормалізація: Приведення зображення до стандартної форми або діапазону значень.

2) Фільтрація дозволяє виділити або згладити певні частини зображення, підкреслити важливі елементи або усунути небажані. Ось декілька таких прикладів:

a) Просторові фільтри: Застосовуються безпосередньо до пікселів зображення.

b) Частотні фільтри: Використовуються для обробки зображення в частотній області.

3) Сегментація ділить зображення на кілька значущих частин або областей для спрощення аналізу. Ось декілька таких прикладів:

a) Порогова сегментація: Виділення областей на основі порогових значень яскравості.

b) Регіональна сегментація: Виділення областей на основі однорідності пікселів.

c) Кластеризація: Розбиття пікселів на групи.

4) Трансформація включає зміну розмірів, форми або орієнтації зображення. Ось декілька таких прикладів:

a) Зміна розміру: Масштабування зображення до потрібного розміру.

b) Поворот і відображення: Зміна орієнтації зображення.

c) Перетворення зсуву і обертання: Зміна геометрії зображення для виправлення перспективи або вирівнювання.

5) Виділення важливих деталей, які можуть бути використані для подальшого аналізу або класифікації. Ось декілька таких прикладів:

a) Виділення країв: Використання методів, таких як оператори Собеля, методика Кенні, щоб знайти межі об'єктів.

b) Виділення контурів: Визначення контурів об'єктів на зображенні.

с) Виділення кутів і точок інтересу: Використання методів, таких як детектор Харріса, для знаходження ключових точок.

б) Методи аналізу та розпізнавання об'єктів і шаблонів на зображенні.

Ось декілька таких прикладів:

а) Розпізнавання образів: Використання алгоритмів машинного навчання для ідентифікації об'єктів.

б) Аналіз текстур: Визначення властивостей поверхні об'єктів на основі їх текстурних характеристик.

с) Аналіз форми: Визначення геометричних характеристик об'єктів, таких як площа, периметр, форма.

7) Реконструкція зображення включає відновлення зображень з дефектами або покращення якості зображення. Ось декілька таких прикладів:

а) Інтерполяція: Відновлення втрачених або пошкоджених частин зображення.

б) Суперрозширення: Збільшення роздільної здатності зображення за допомогою алгоритмів машинного навчання.

8) Колірна корекція включає зміну кольорових характеристик зображення для досягнення потрібного візуального ефекту або відповідності стандартам. Ось декілька таких прикладів:

а) Баланс білого: Корекція кольорових відтінків для досягнення природних кольорів.

б) Корекція гамми: Зміна яскравості середніх тонів зображення.

с) Перетворення кольорових моделей: Зміна кольорових моделей.

Ці елементи обробки зображення допомагають досягти потрібного результату в різних сферах застосування, від фотографії до медичної діагностики і комп'ютерного зору.

2. ПОСТАНОВКА ЗАДАЧІ

2.1 Задачі застосунка

До основних задач інформаційної системи відносяться наступні:

- **Обробка зображень у реальному часі:** Забезпечення користувачів можливістю редагувати зображення миттєво, з відображенням змін у режимі реального часу. Це включає в себе застосування фільтрів, корекцію кольорів, зміну розмірів та інші операції, що забезпечують процес редагування.

- **Додання нових інструментів та фільтрів для редагування зображень:** Розширення функціоналу застосунку шляхом інтеграції нових інструментів, таких як інструменти для малювання, роботи з шарами, спеціальні фільтри та ефекти. Це дозволяє користувачам мати доступ до більшого набору можливостей для досягнення бажаних результатів при редагуванні зображень.

- **Обмін файлами між користувачами та адміністраторами:** Забезпечення безпечного та швидкого обміну зображеннями та іншими файлами між різними користувачами системи, включаючи адміністраторів. Це включає в себе функції завантаження, збереження та спільного використання файлів, а також забезпечення захисту даних під час передачі.

- **Створення та оновлення динамічних зображень з використанням вбудованих модулів:** Надання користувачам можливості створювати та оновлювати зображення, використовуючи вбудовані модулі, які замінюють необхідність внутрішнього програмного редагування. Це дозволяє користувачам легко додавати нові елементи, змінювати існуючі та оновлювати зображення без необхідності глибоких технічних знань.

Ці задачі забезпечують всебічну функціональність інформаційної системи, що дозволяє користувачам і адміністраторам ефективно працювати з зображеннями, забезпечуючи високу продуктивність, зручність та безпеку використання.

3 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для створення сервера був обраний сервіс для віртуального хостингу Aternos (рис. 3.1), який дозволяє перетворювати локальний сервер на віртуальний. За допомогою цього сервісу можна надати доступ до локального сервера користувачам з зовнішньої мережі, що робить його зручним для багатьох застосувань, таких як онлайн-ігри, спільна робота або навчальні проекти.

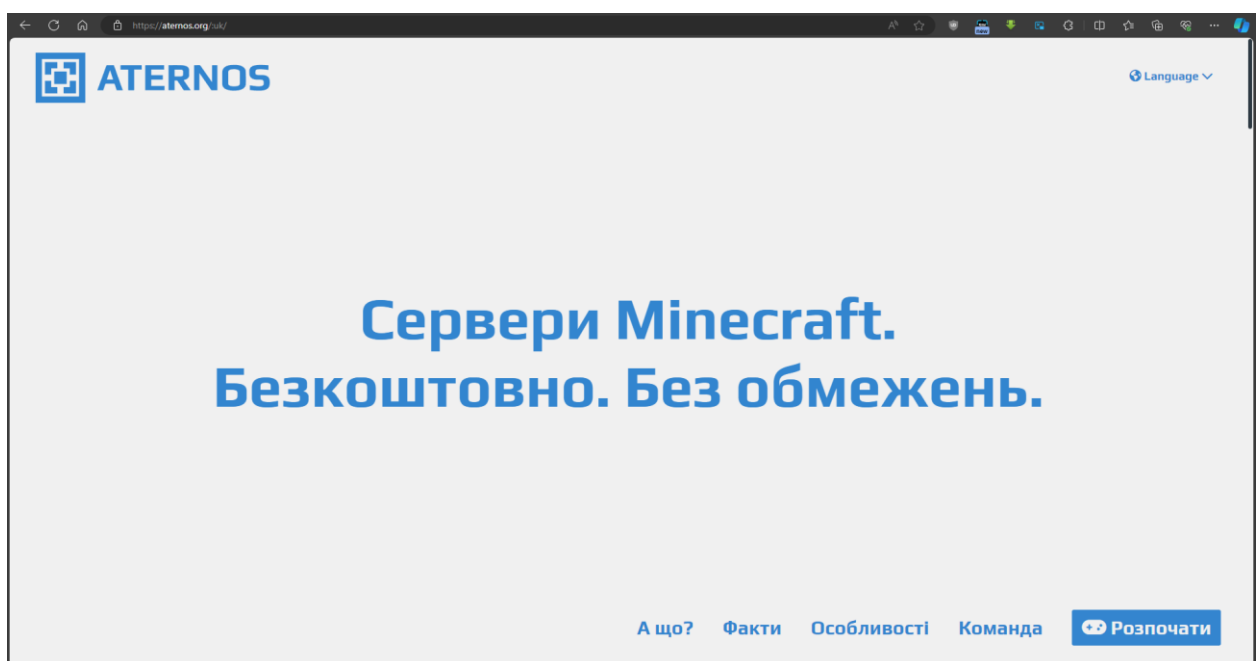


Рисунок 3.1 – Головна сторінка сервісу Aternos

Aternos надає чудові можливості як для хостингу, так і для налаштування самого сервера та доступу до нього. На сайті Aternos користувачі можуть обирати конфігурацію сервера, визначати, чи буде це звичайний сервер Minecraft, чи сервер з різними модифікаціями та можливостями. Це дозволяє гнучко налаштовувати сервер відповідно до потреб користувача, додаючи або видаляючи модифікації, встановлюючи плагіни та інші компоненти для покращення функціональності серверу.

До основних переваг використання Aternos відносяться:

- **Безкоштовність:** Aternos надає послуги хостингу безкоштовно, що робить його доступним для широкого кола користувачів.

- **Простота у використанні:** Інтерфейс сервісу інтуїтивно зрозумілий і не потребує глибоких технічних знань для налаштування та запуску серверу.

- **Гнучкість конфігурації:** Можливість налаштування сервера під конкретні потреби, включаючи вибір модифікацій, плагінів та інших компонентів.

- **Доступність з будь-якої точки світу:** Сервери, хостовані на Aternos, доступні з будь-якої точки світу, де є інтернет-з'єднання.

- **Постійне оновлення та підтримка:** Aternos регулярно оновлює свої сервіси, забезпечуючи підтримку нових версій Minecraft та модифікацій [5].

Для створення звичайного локального сервера не потрібно нічого особливого, окрім встановлення Minecraft Server на комп'ютер користувача. Однак, для перенесення локального сервера на віртуальний хостинг потрібна мова програмування Java. Java є основною мовою для розробки та налаштування Minecraft серверів, оскільки саме на ній написано більшість серверних модифікацій та плагінів[6].

Для налаштування сервера на Java необхідна бібліотека Minecraft Development (рис. 3.2) та ядро Paper, яке надає додаткові можливості для налаштування та оптимізації серверу. Ядро Paper є покращеною версією стандартного Minecraft сервера, яка забезпечує кращу продуктивність, стабільність та розширені можливості для адміністрування серверу.

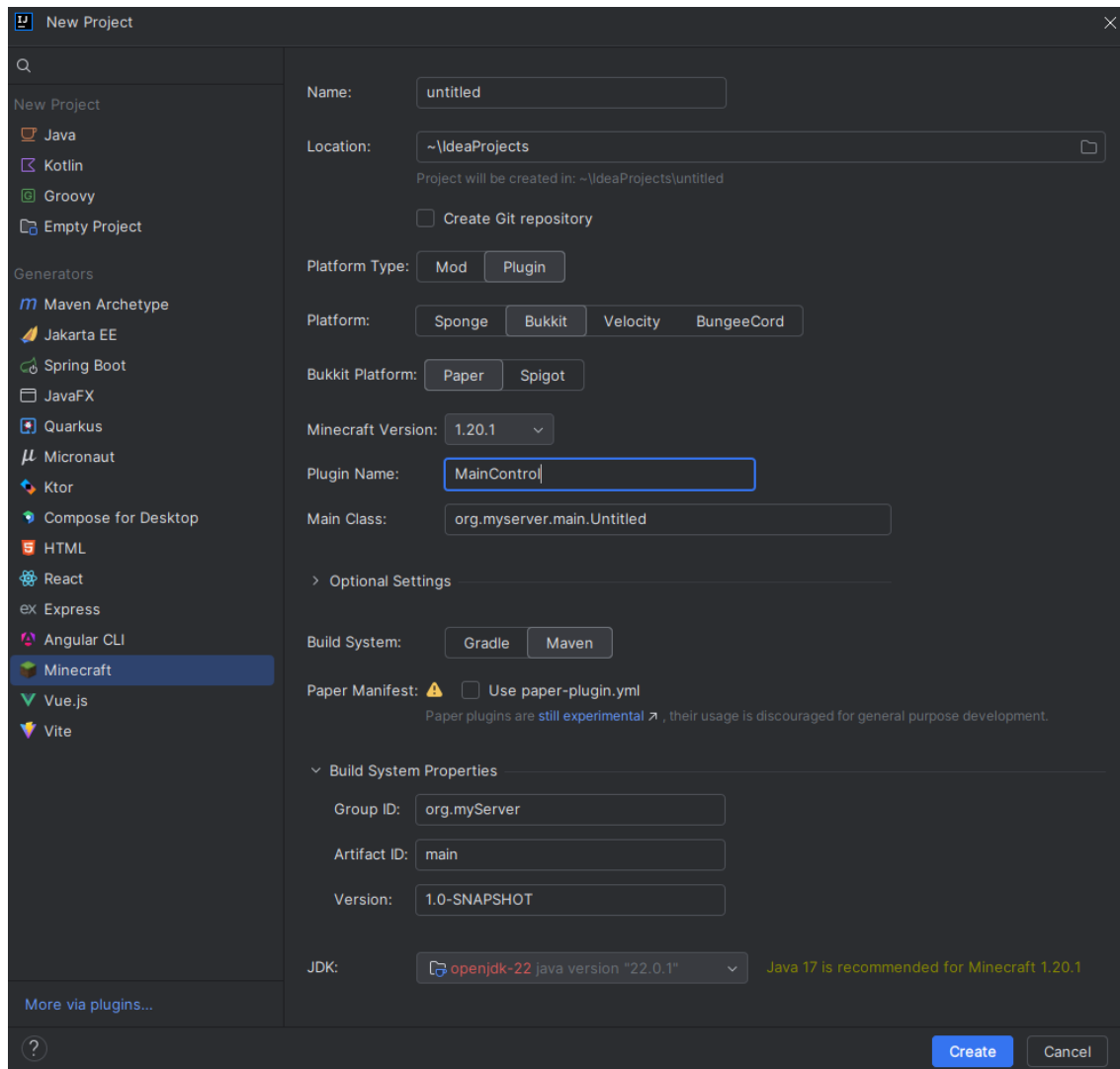


Рисунок 3.2 – Панель створення проекту на мові програмування Java

Для створення проекту на Java використовується середовище розробки, яке підтримує інтеграцію з бібліотекою Minecraft Development. Ця бібліотека включає інструменти та ресурси, які полегшують процес створення модифікацій та плагінів для Minecraft. Використовуючи ці інструменти, розробники можуть створювати складні та багатофункціональні сервери, які відповідають конкретним потребам користувачів[3].

Одним з ключових компонентів для створення локального сервера є ядро Paper, яке можна завантажити з офіційного сайту (рис. 3.3).

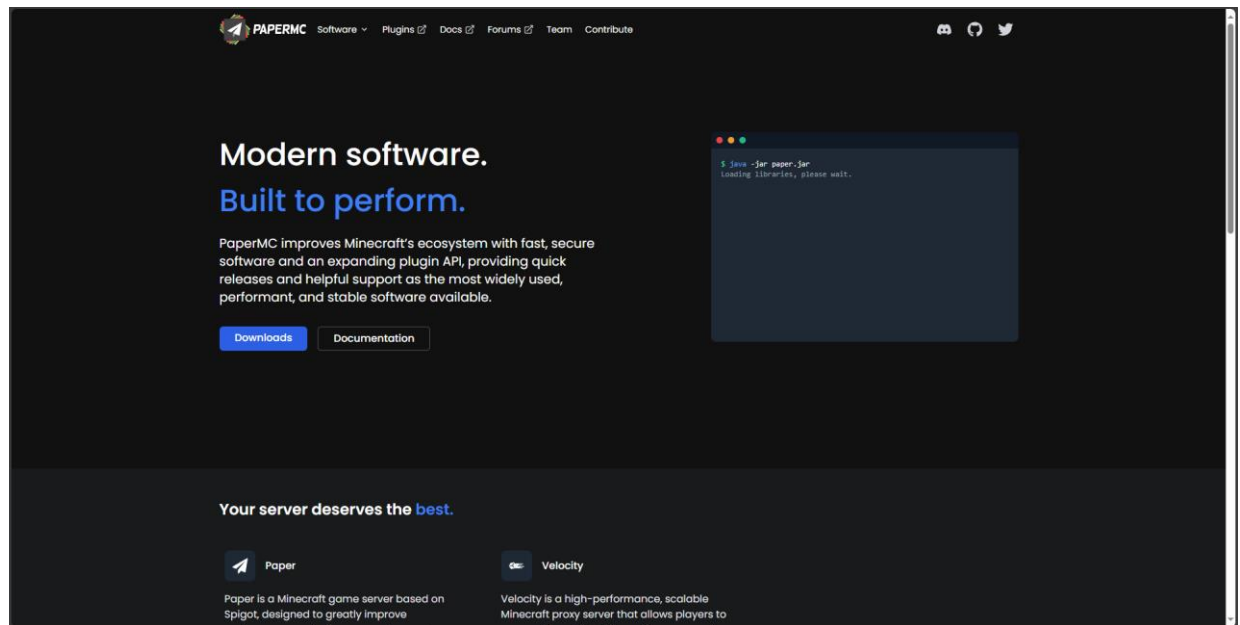


Рисунок 3.3 – Офіційний сайт ядра Paper

Ядро Paper необхідне для створення локального сервера, який не залежить від дій користувача і забезпечує стабільну роботу серверу навіть при великій кількості підключень та високому навантаженні. Це ядро забезпечує поліпшену продуктивність та додаткові можливості для адміністрування серверу, що робить його ідеальним вибором для більшості проектів.

До основних переваг використання ядра Paper відносяться:

- **Підвищена продуктивність:** Оптимізовані алгоритми обробки, які забезпечують швидшу і стабільнішу роботу сервера.

- **Можливість розширення:** Підтримка великої кількості плагінів та модифікацій, що дозволяє додавати нові функції без значних зусиль.

- **Покращена стабільність:** Висока стійкість до збоїв та підтримка безперебійної роботи навіть при високому навантаженні.

- **Широкі можливості налаштування:** Гнучкі налаштування дозволяють адміністраторам детально конфігурувати сервер відповідно до специфічних вимог.

- **Активна спільнота:** Велика спільнота користувачів та розробників, яка постійно доповнює та покращує можливості ядра [4].

Таким чином, використання сервісу Aternos та ядра Paper забезпечує потужний і гнучкий інструмент для створення та адміністрування серверів Minecraft, надаючи користувачам всі необхідні ресурси для реалізації їхніх проектів.

Для створення зовнішнього інтерфейсу була обрана бібліотека JavaFX рис. 3.4, яка надає зручний спосіб розробки графічних інтерфейсів на мові програмування Java. JavaFX дозволяє створювати сучасні та інтерактивні користувацькі інтерфейси, які забезпечують комфортну роботу користувачів з сервером. Завдяки широкому набору інструментів та компонентів, JavaFX дозволяє розробникам створювати привабливі та функціональні інтерфейси, що значно полегшує взаємодію користувачів з сервером.

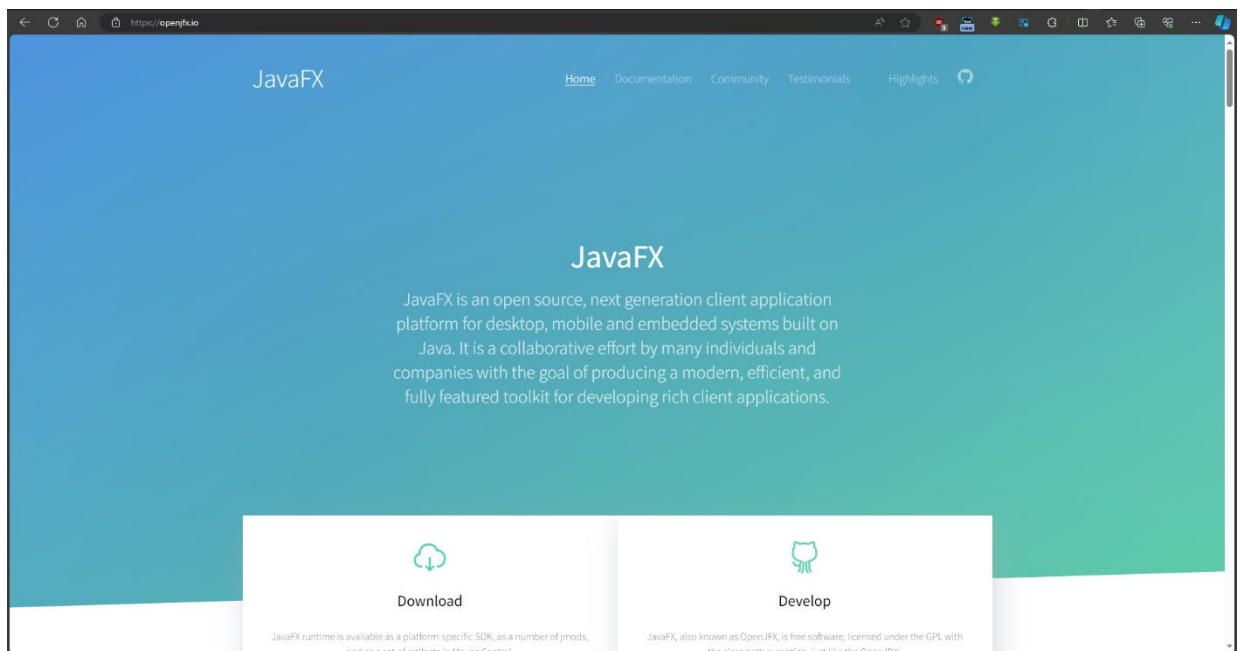


Рисунок 3.4 – Офіційний сайт JavaFX

JavaFX є потужним інструментом для створення сучасних графічних інтерфейсів користувача (GUI) на платформі Java. Він пропонує низку переваг, які роблять його популярним серед розробників:

- **Висока продуктивність:** JavaFX використовує апаратне прискорення для рендерингу графіки, що забезпечує високу продуктивність навіть для складних графічних елементів.

- **Широкі можливості для створення GUI:** JavaFX підтримує різноманітні компоненти користувацького інтерфейсу, включаючи кнопки, текстові поля, таблиці, графіки та інші елементи, що дозволяє створювати інтуїтивно зрозумілі і зручні інтерфейси.

- **CSS та FXML:** JavaFX підтримує CSS для стилізації інтерфейсу, що дозволяє легко змінювати зовнішній вигляд додатку. FXML, XML-подібна мова розмітки, дозволяє розробникам описувати інтерфейс користувача окремо від логіки програми, що полегшує розробку і підтримку коду.

- **Модульність:** JavaFX побудовано на модульній архітектурі, що дозволяє розробникам використовувати тільки ті компоненти, які їм потрібні, зменшуючи розмір кінцевого додатку та підвищуючи його ефективність.

Ці переваги роблять JavaFX потужним інструментом для розробки сучасних, продуктивних та кросплатформених графічних інтерфейсів користувача, що задовольняють потреби як розробників, так і кінцевих користувачів [2].

4. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСТАСУНКУ

Основною архітектурою для системи обрана дворівнева архітектура клієнт-сервер рис. 4.1

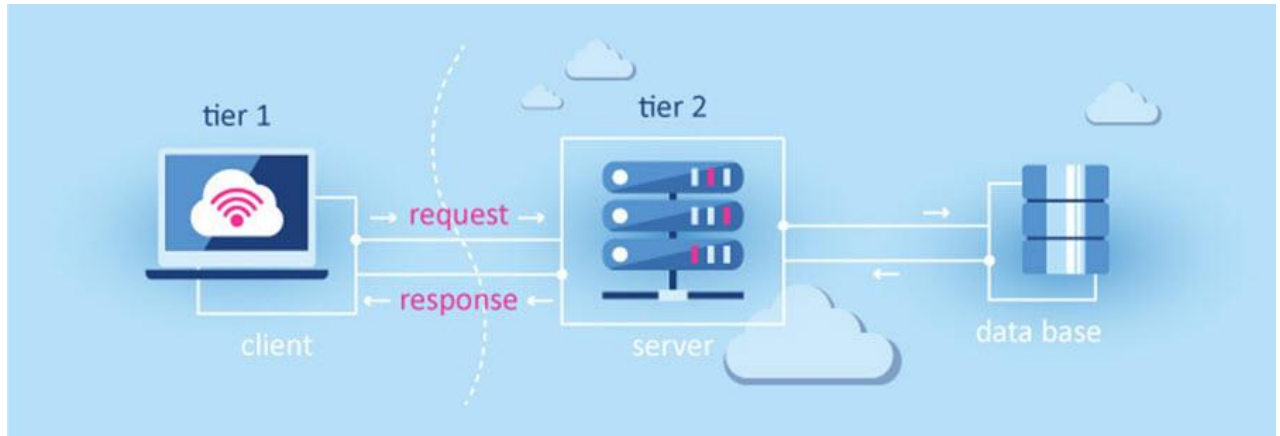


Рисунок 4.1 – дворівнева архітектура [1]

У комп'ютерних технологіях дворівнева архітектура без бази даних передбачає наявність двох основних компонентів: клієнтського додатка та сервера додатків. Ця архітектура спрощує процес розгортання і підтримки системи, оскільки виключає необхідність окремого сервера баз даних.

Термінал у такій дворівневій архітектурі виконує роль інтерфейсу користувача (зазвичай графічного) та представляє перший рівень — додаток для кінцевого користувача. Перший рівень обробляє взаємодію з користувачем та може містити певну бізнес-логіку. В основному клієнтський додаток відповідає за відображення даних користувачу, отримання введення від нього, а також передачу цього введення на сервер додатків для подальшої обробки. Прикладом клієнтського додатка можуть бути веб-додатки, мобільні додатки або desktop-додатки, які взаємодіють із сервером додатків через мережевий інтерфейс.

Сервер додатків, який забезпечує виконання бізнес-логіки та зберігання даних у пам'яті або файлової системі, розташовується на другому рівні. У цьому випадку сервер додатків виконує роль центрального компоненту, який обробляє запити від клієнтських додатків і забезпечує зберігання даних без

використання традиційних баз даних. Він може використовувати різні технології для зберігання даних, такі як XML-файли, JSON-файли або інші формати файлів. Сервер додатків також може мати вбудовані механізми кешування для підвищення продуктивності.

Основними перевагами дворівневої архітектури:

1) Простота розгортання і підтримки. У дворівневій архітектурі менше компонентів, тому простіше розгорнути і підтримувати додаток.

2) Прямий доступ до даних. Клієнтський додаток може напряму звертатися до сервера бази даних, що може спростувати розробку та зменшувати затримки у взаємодії.

3) Зниження витрат на інфраструктуру. Відсутність проміжного сервера додатків зменшує витрати на обладнання та адміністрування.

Найбільш частими недоліками дворівневої архітектури являються:

1) Знижена цілісність даних. Оскільки клієнтський додаток має прямий доступ до бази даних, виникає більший ризик пошкодження даних ненадійними клієнтськими програмами.

2) Гірша безпека. Клієнтський додаток має прямий доступ до бази даних, що ускладнює контроль доступу і збільшує ризик несанкціонованого доступу до даних.

3) Менша модульність. Зміна одного рівня (наприклад, бази даних) може вимагати значних змін у клієнтському додатку, що знижує гнучкість системи.

Дворівнева архітектура може бути корисною у випадках, коли необхідно швидко розгорнути прототип або невеликий додаток з обмеженим обсягом даних. Вона також може бути застосована в умовах, де вимоги до безпеки і цілісності даних не є критичними. Такий підхід часто використовується для локальних додатків, внутрішніх інструментів компаній або навчальних проєктів, де простота і швидкість розробки мають пріоритет над масштабованістю і безпекою.

Шаблон проектування додатку обрано за моделлю MVC (рис. 4.2).

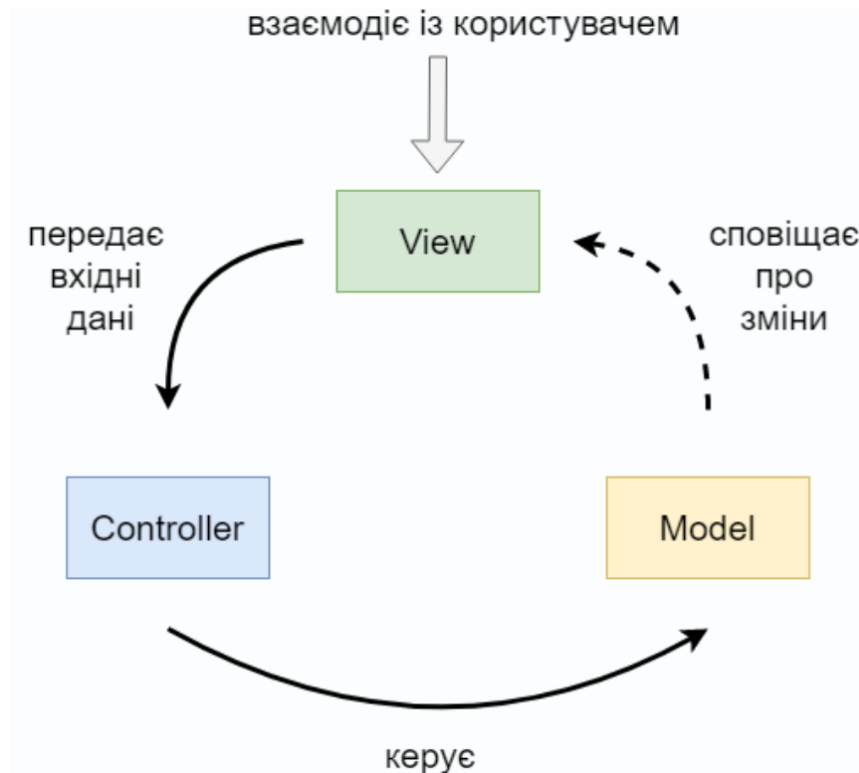


Рисунок 4.2 – Шаблон MVC

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, представлення та модуль керування. Він застосовується для відокремлення даних (моделі) від інтерфейсу користувача (представлення), що мінімізує вплив змін інтерфейсу користувача на роботу з даними, та дозволяє змінювати модель даних без необхідності змінювати інтерфейс користувача.

Мета використання шаблону MVC – створити гнучкий дизайн програмного забезпечення, який полегшує внесення подальших змін або розширень програми, а також надає можливість повторного використання окремих компонентів. Використання цього шаблону у великих системах сприяє впорядкованості структури та зменшує їхню складність, роблячи їх більш зрозумілими.

У рамках архітектурного шаблону модель-представлення-контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами:

- Модель (Model) відповідає за зберігання даних та їхню структуру;

- Представлення (View) відповідає за відображення цих даних користувачеві, тобто за інтерфейс програми;

- Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакцій на дії користувача і передає дані у модель;

Переваги використання шаблону MVC:

1) Відокремлення інтерфейсу користувача від бізнес-логіки. MVC підвищує гнучкість, зручність обслуговування та масштабованість, оскільки бізнес-логіка відокремлена від логіки доступу до даних;

2) Повторне використання компонентів. Наприклад, якщо потрібно замінити інтерфейс веб-браузера на інтерфейс користувача Windows, це можна зробити швидко і просто, замінивши лише компонент інтерфейсу користувача. Усі інші компоненти, такі як бізнес-логіка, доступ до даних і база даних, залишаються незмінними;

3) Незалежне розгортання компонентів. У MVC можна створювати різні компоненти програми, які можна незалежно розгортати, підтримувати та оновлювати в різний час;

4) Тестування компонентів. Шаблон MVC дозволяє тестувати компоненти незалежно один від одного, що полегшує виявлення та виправлення помилок [7].

5 ІНФОРМАЦІЙНЕ МОДЕЛЮВАННЯ

Термінал у цій системі має інтерфейс, який дозволяє користувачам взаємодіяти з ним шляхом натискання на кнопки. Ці кнопки відправляють запити до сервера, який, у свою чергу, змінює зображення або середовище відповідно до отриманих команд. Запити містять набір команд, які були визначені заздалегідь та зазначені в терміналі.

Термінал є графічним інтерфейсом, що надає користувачам можливість вибирати різні команди через кнопки або інші елементи управління. Кожна команда відповідає певній дії, яку потрібно виконати на сервері. Наприклад, це можуть бути команди для зміни кольору зображення, додавання нових об'єктів у середовище або зміни положення існуючих об'єктів. Користувач, натискаючи на кнопки терміналу, надсилає ці команди до сервера для обробки.

Сервер розташовується на другому рівні системи, тим самим він є багатокористувацькою точкою доступу з унікальною IP-адресою, що дозволяє підключатися до нього різними клієнтами одночасно. Сервер симулює зображення (середовище) в реальному часі, приймаючи команди від клієнтів і виконуючи їх для зміни стану зображення (середовище).

Коли сервер отримує запит від клієнта, він розшифровує команду та виконує відповідні дії для зміни зображення (середовища). Наприклад, якщо клієнт відправляє команду для зміни фону зображення, сервер обробляє цю команду і змінює колір фону відповідно.

Після виконання команди сервер надсилає відповідь до клієнта, яка полягає у зміні стану зображення (середовища). Ця відповідь буде відображена на терміналі сервера та у вигляді оновленого зображення в терміналі. Таким чином, користувачі можуть бачити результати своїх дій у реальному часі, що створює інтерактивний та динамічний досвід взаємодії з системою.

Розглянемо приклад взаємодії користувача з системою. Користувач відкриває програму і бачить набір кнопок, кожна з яких відповідає певній

команді. Натискаючи на кнопку «Змінити фон на синій», термінал надсилає відповідний запит на сервер. Сервер отримує запит, розпізнає команду і змінює фон зображення на синій колір. Після цього сервер надсилає відповідь до терміналу, який відображає змінений фон. Користувач бачить, що фон змінився на синій, і може продовжувати взаємодію з терміналом для виконання інших дій.

6 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ

Для того, щоб користувачі мали змогу редагувати зображення, необхідно розробити користувацький інтерфейс, а також відповідний серверний код, який буде обробляти запити користувачів, здійснювати оновлення зображень та повертати результати назад користувачам. У наведеному прикладі показано реалізацію форми для зміни зображення за допомогою мови розмітки FXML, яка використовується в JavaFX для створення графічних інтерфейсів.

У лістингу 6.1 наведений приклад реалізації форми для зміни зображення а саме часу:

```

                                <Text    layoutX="280.0"    layoutY="50.0"
strokeType="OUTSIDE"           strokeWidth="0.0"           text="Час"
wrappingWidth="73.0" />
                                <ChoiceBox fx:id="time" layoutX="340.0"
layoutY="30.0" prefWidth="170.0" />

```

Лістинг 6.1 – Приклад створення приєму порамерта часу

Після створення інтерфейсу необхідно забезпечити обробку вхідних значень та відображення тих, що вже існують. Для того щоб інтерфейс не був просто набором написів з полями, реалізується передача даних з шаблону до сервера.

```

public void getTime(ActionEvent actionEvent) {
    System.out.println("/time set " + RutoEn(time.getValue(), times,
timesEn));
}

```

Лістинг 6.2 – Приклад передічі порамерту часу в программі

Дана функція оброблює інформацію з форми та передає її на сервер.

7 СТВОРЕННЯ СЕРВЕРУ

Для створення сервера на надійному хостингу важливо спочатку налаштувати його у грі. Цей процес починається зі входу в гру та вибору режиму "Гра наодинці" рис. 7.1. Ця дія відкриє вікно з усіма існуючими світами, створеними користувачем протягом гри. Після вибору цього режиму, користувач отримує доступ до можливостей створення нового світу або вибору існуючого для подальших налаштувань.



Рисунок 7.1 – Головне меню

Після вибору режиму "Гра наодинці", система автоматично відкриває вікно, де відображаються всі раніше створені світи. Для створення нового світу необхідно скористатися опцією "Створити новий світ", яка знаходиться в меню або на екрані, як показано на рис. 7.2.

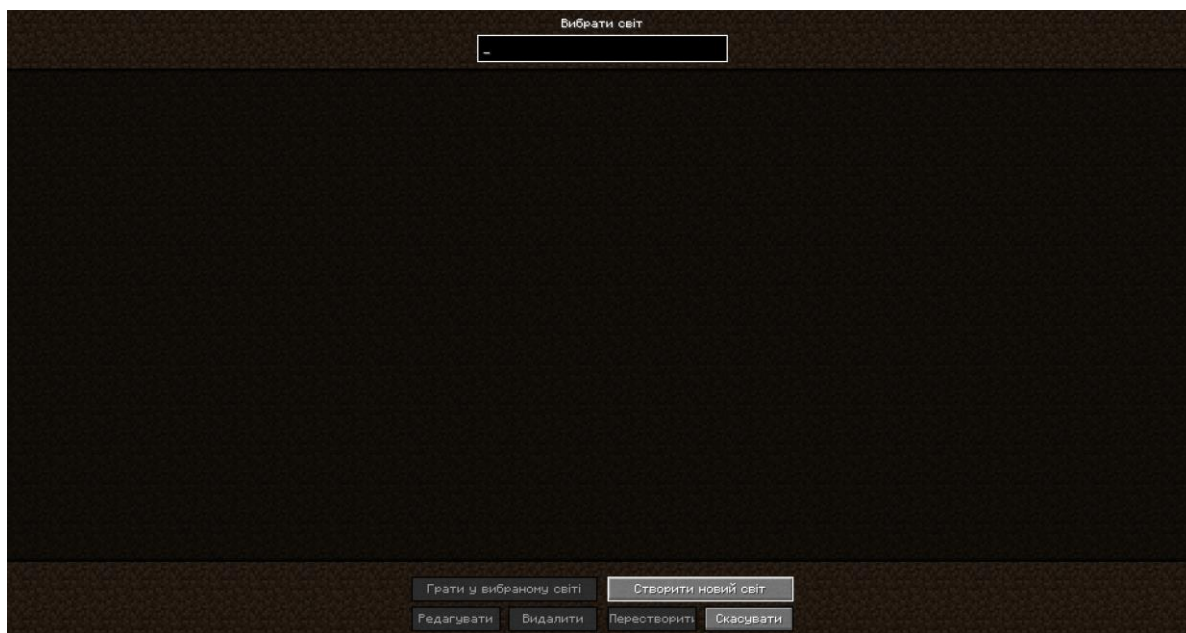


Рисунок 7.2 – Меню вибору світу

Після натискання кнопки "Створити новий світ" відкриється нова вкладка з налаштуваннями для нового світу, який збираєтеся створити, згідно з рис. 7.3. В цьому вікні мають можливість налаштувати різні параметри світу, такі як тип території, рівень складності тощо. Однак, не потрібно змінювати жодні налаштування і потрібно створити стандартний світ, просто натиснути кнопку "Створити новий світ" без будь-яких додаткових змін.

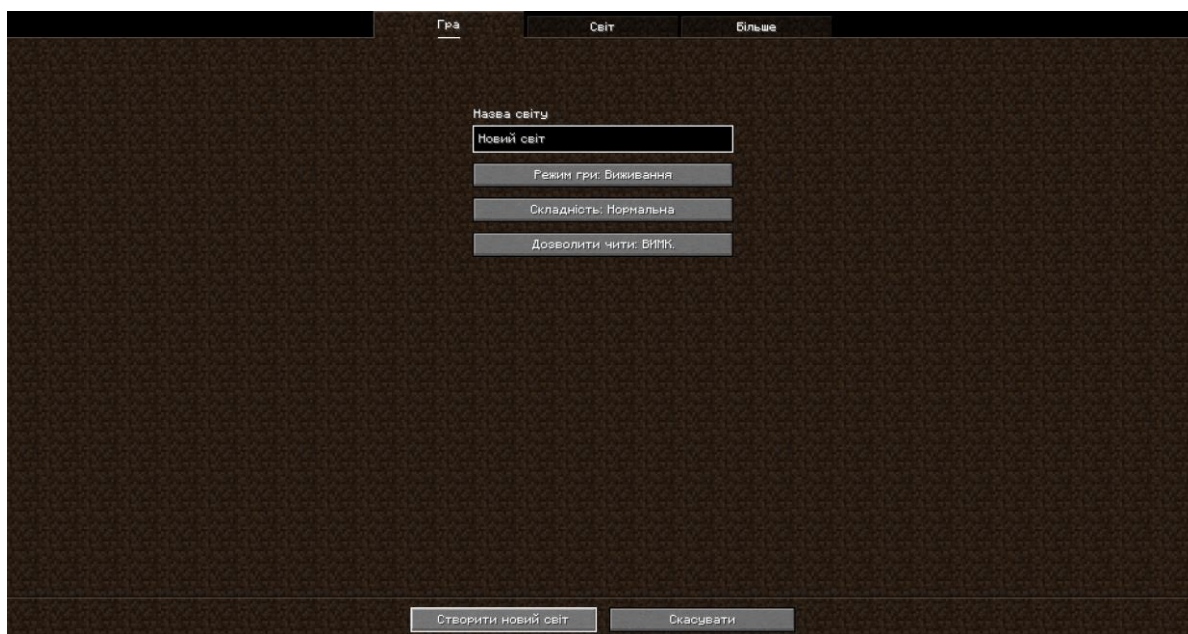


Рисунок 7.3 – Меню створення світу

Після успішного створення нового світу потрібно вийти з гри та перейти до наступного кроку - створення локального серверу. Для цього потрібно відвідати офіційний веб-сайт Paper та завантажити ядро сервера, яке відповідає версії вашого Minecraft, згідно з рис. 7.4.

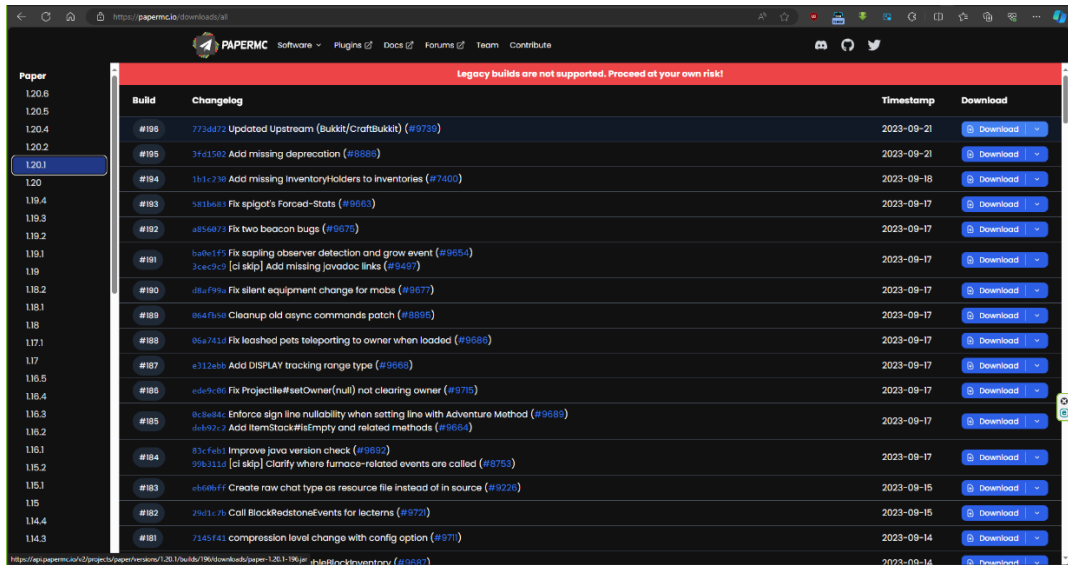


Рисунок 7.4 – Сторінка сайту Paper з завантаженням ядра

Після успішного завантаження ядра сервера необхідно перемістити його у кореневу папку світу, який вже був створен для створення сервера. Це можна зробити, як показано на рисунку 7.5.

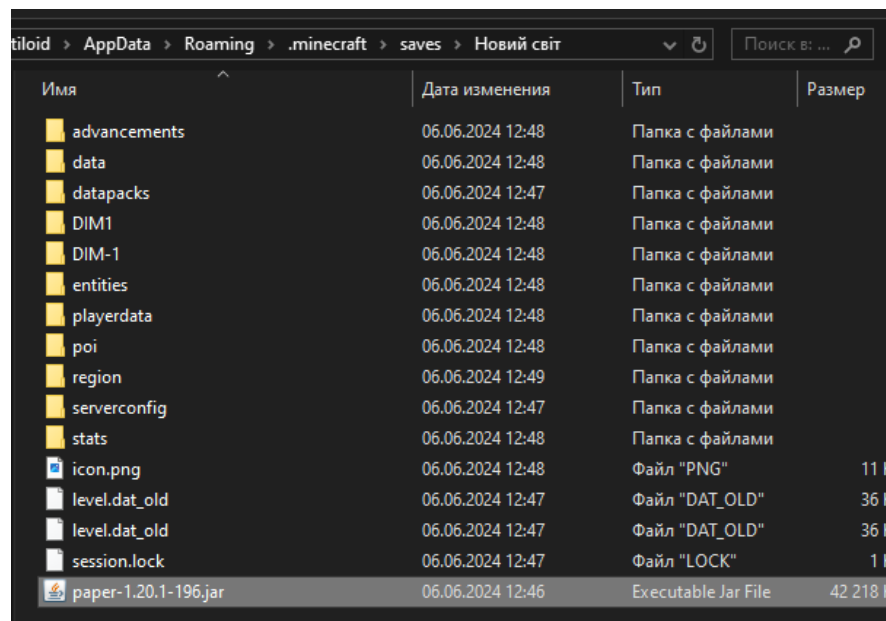
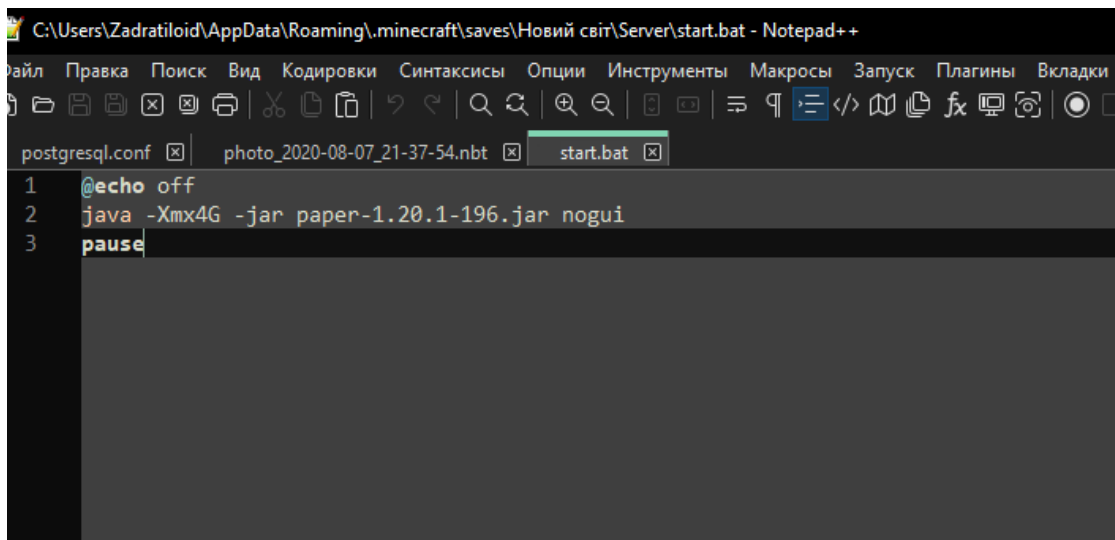


Рисунок 7.5 – Об'єднання не активованого ядра та світу

Після завантаження ядра сервера у кореневу папку світу, для того щоб змінити світ у локальний сервер, потрібно створити файл з назвою "start.bat" та вказати у ньому кількість ресурсів, які система повинна виділити під сервер. Цей файл "start.bat" служитиме для запуску сервера. Для цього можна відкрити будь-який текстовий редактор, такий як Notepad, та створити новий файл з назвою "start.bat". Після цього потрібно відкрити цей файл і ввести в нього необхідні команди, щоб запустити сервер. Наприклад, команда "java -Xmx4G -jar server.jar" (рис. 7.6) встановлює початковий обсяг оперативної пам'яті для сервера максимальний обсяг на 4 гігабайт. Після створення файлу "start.bat" можемо виконати його, щоб запустити сервер.



```
C:\Users\Zadratiloid\AppData\Roaming\.minecraft\saves\Новий світ\Server\start.bat - Notepad++
Файл  Правка  Поиск  Вид  Кодировки  Синтаксисы  Опции  Инструменты  Макросы  Запуск  Плагины  Вкладки
postgres.conf  photo_2020-08-07_21-37-54.nbt  start.bat
1 @echo off
2 java -Xmx4G -jar paper-1.20.1-196.jar nogui
3 pause
```

Рисунок 7.6 – Файл start.bat з налаштуваннями

Після завершення всіх налаштувань та запуску файлу з налаштуваннями, необхідно дочекатися завершення процесу перетворення світу у сервер. Цей процес може зайняти декілько хвилин. Після завершення процесу можна переглянути нові файли (див. рис. 7.7), які підтверджують, що локальний сервер був успішно створений.

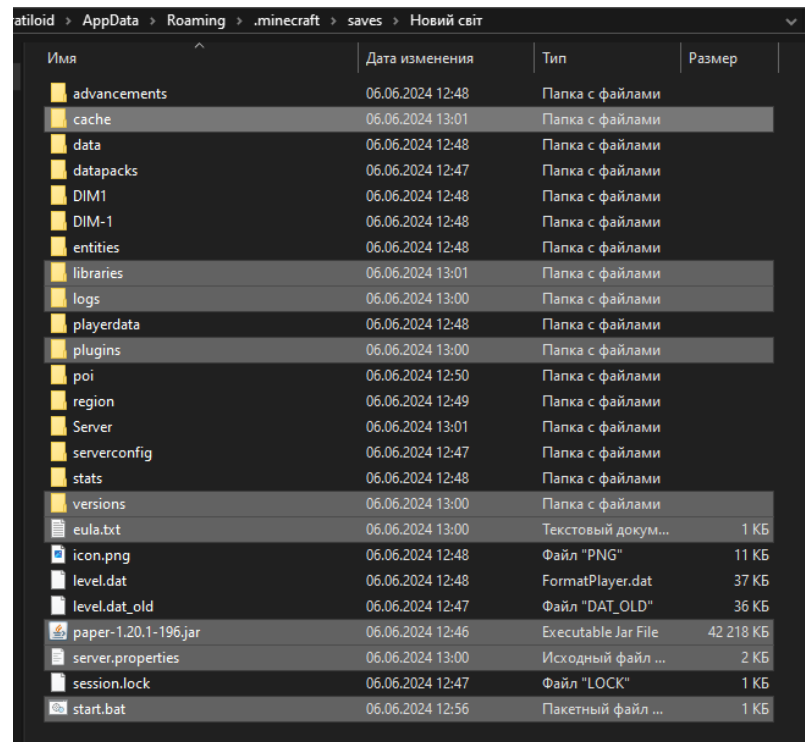


Рисунок 7.7 – Провідник з переробленим світом на сервер

Після завершення процесу створення сервера важливо перевірити його доступність і коректну роботу. Для цього заходимо в гру та переходимо до меню "Гра в мережі", щоб переглянути список усіх доступних серверів (рис. 7.8). Це дозволить переконатися, що новий сервер з'явився у списку та можна з ним підключитися. Це важливий крок, який підтверджує успішне створення та готовність сервера до використання.



Рисунок 7.8 – Меню мережевих ігор

Для початку, натисніть кнопку "Додати" на екрані, що відкриється, після чого з'явиться нове вікно з можливістю введення назви та адреси сервера. Назва сервера необов'язкова для зміни і може залишитися без змін. У полі адреси сервера введіть IP-адресу "127.0.0.1", яка вказує на той самий комп'ютер, на якому запущен сервер (див. рис. 7.9). Така адреса дозволяє підключитися до локального сервера, який працює на комп'ютері.

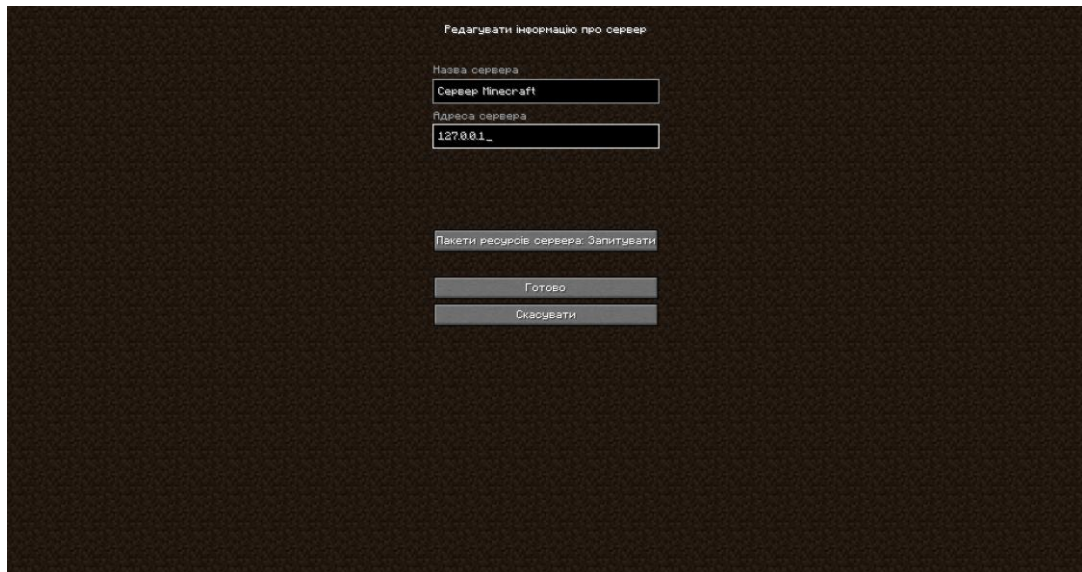


Рисунок 7.9 – Додавання нового серверу

Після успішного додавання сервера він автоматично з'явиться у списку всіх доступних серверів. Підключитися до нього можна подвійним натисканням на його назву або значок. Після цього відкриється вікно, яке дозволить вам увійти на цей сервер (див. рис. 7.10).



Рисунок 7.10 – Підключення до серверу

Можемо побачити що успішно зайшли на сервер. Після того як був створен локальний сервер переносимо його до віртуального хостинга. Для перенесення на віртуальний хостинг переходимо до сайту Aternus рис. 7.11.

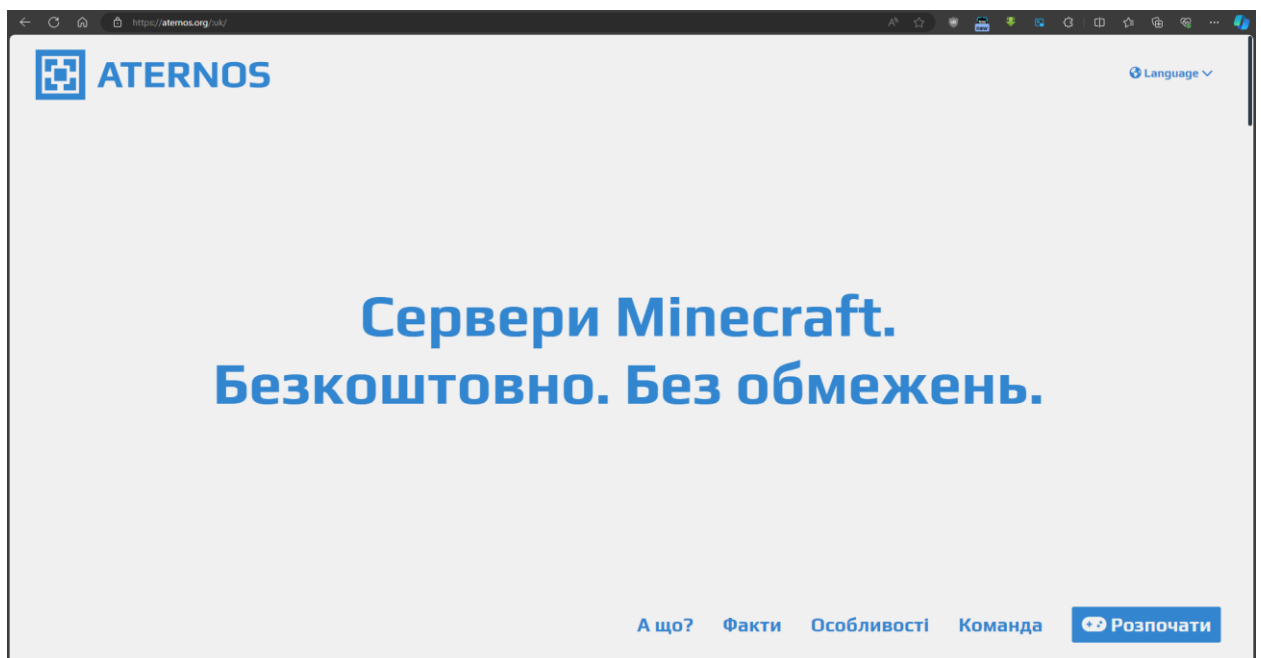


Рисунок 7.11 – Головна сторінка сайту Aternus

Перед початком перенесення сервера на віртуальний хостинг, необхідно авторизуватися на веб-сайт рис 7.12. Цей крок передбачає введення облікових

даних, таких як ім'я користувача та пароль, для входу до облікового запису на платформі Aternos. Це не лише забезпечить безпеку облікового запису, але й дозволить отримати доступ до функцій управління серверами та іншими користувацькими опціями.

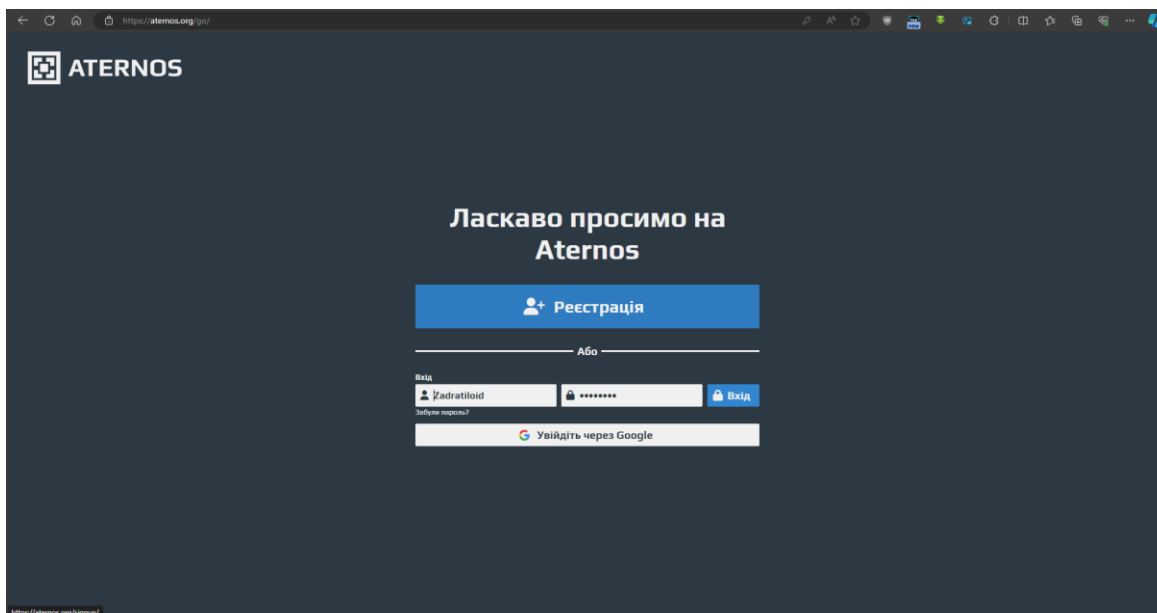


Рисунок 7.12 – Авторизація на сайті

Після успішної авторизації на сайті Aternos з'явиться вікно, в якому будуть перелічені всі сервери, доступні для редагування та керування рис 7.13. Це вікно надасть вам зручний інтерфейс для вибору конкретного сервера.

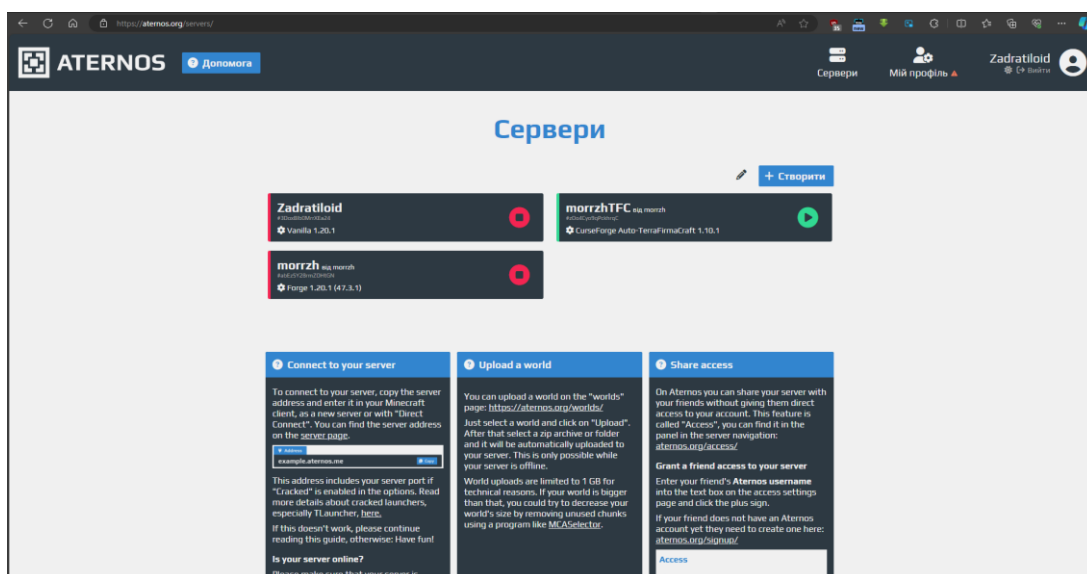


Рисунок 7.13 – Сторінка зі всіма серверами

Після натискання кнопки "Створити" з'явиться нова вкладка з назвою "Світ" рис 7.14. На цій вкладці буде запропоновано додати світ, який буде використовуватися на сервері. Натисніть кнопку "Додати", щоб вибрати потрібний світ для сервера.

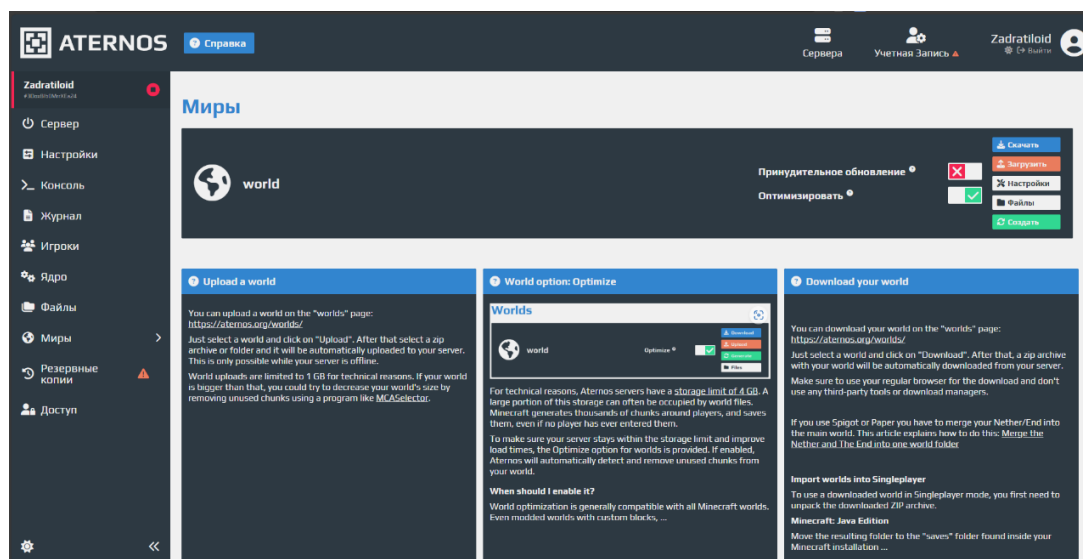


Рисунок 7.14 – Сайт Aternus меню створення світу

Після того, як натиснути на кнопку "Додати", відкриється провідник файлів, де потрібно буде визначити кореневу папку сервера який вже був створен рис 7.15. Після вибору цієї папки буде надана можливість завантажити сервер на віртуальний хостинг.

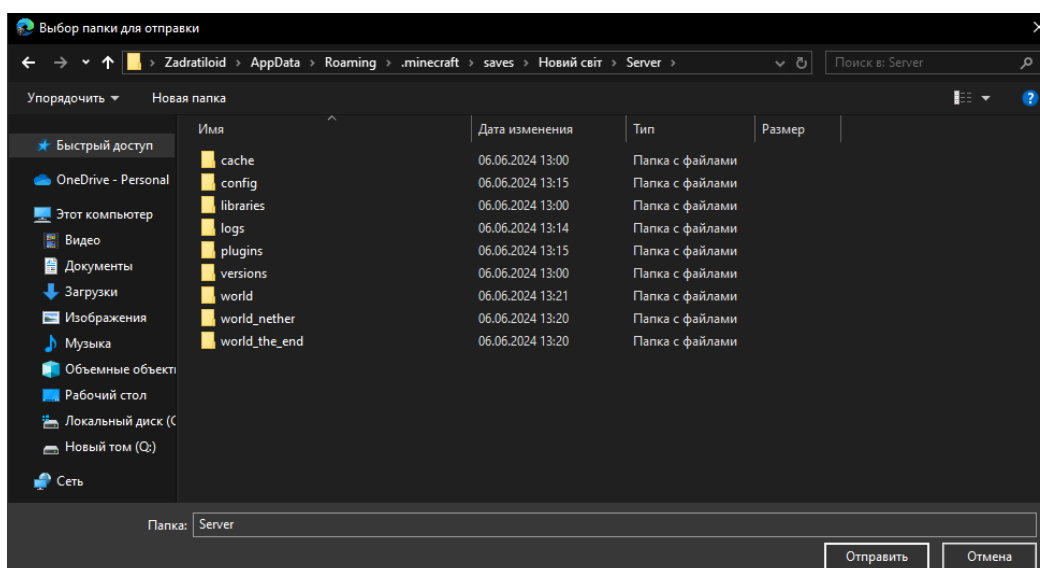


Рисунок 7.15 – Вибір світу для сервера

Після завершення процесу завантаження сервера на віртуальний хостинг, важливо переконатися, що все налаштовано належним чином перед запуском. Після цього можемо ініціювати його запуск, щоб сервер був доступний для користувачів рис 7.16.

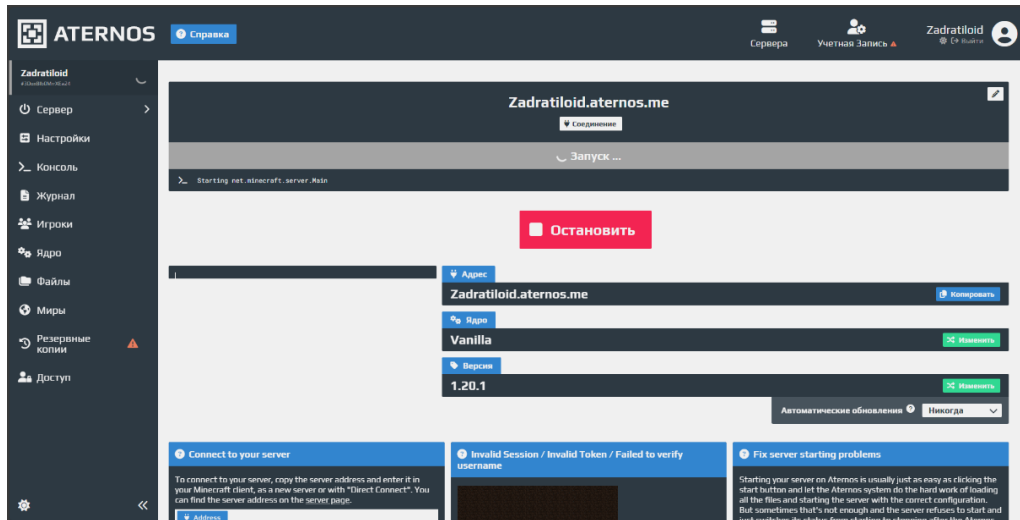


Рисунок 7.16 – Запуск сервера

Щоб додати сервер у гру, спочатку потрібно відкрити гру та перейти до меню для додавання сервера. Після натискання кнопки "Додати", у відкритому вікні у поле "Адреса" вводимо адресу сервера, яку отримали при запуску сервера раніше рис. 7.17. Ця адреса зазвичай представляє собою числове значення, таке як "Zadratiloid.aternos.me", що вказує на локальний сервер.

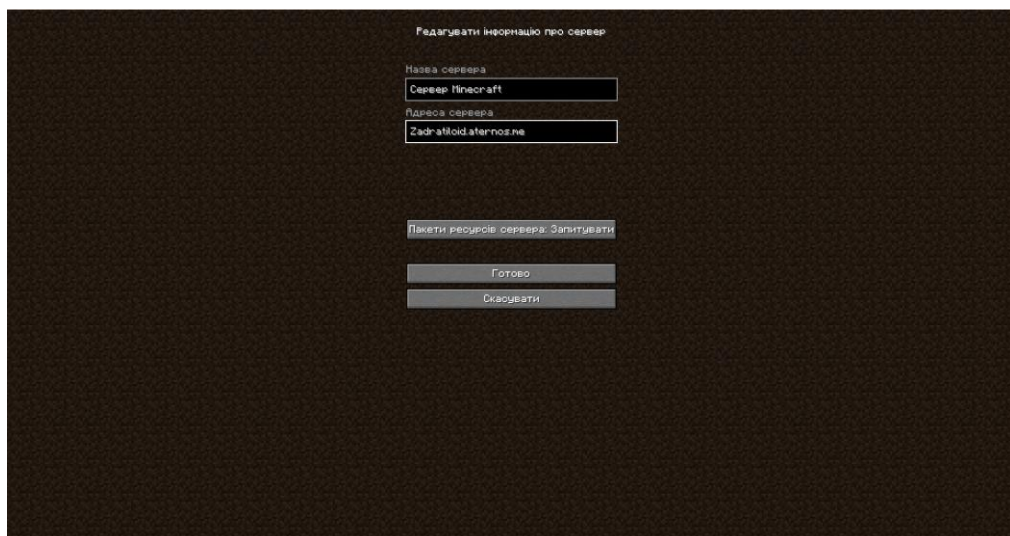


Рисунок 7.17 – Додавання нового серверу

Після успішного підключення до сервера (див. рис. 7.18), можна впевнитися, що це той самий світ, який був створений раніше. Це важливо, оскільки підтверджує, що налаштування підключення здійснено правильно, і маємо доступ до необхідного сервера. Таке підтвердження дозволяють продовжувати роботу з сервером з впевненістю в його ідентифікації та доступності.

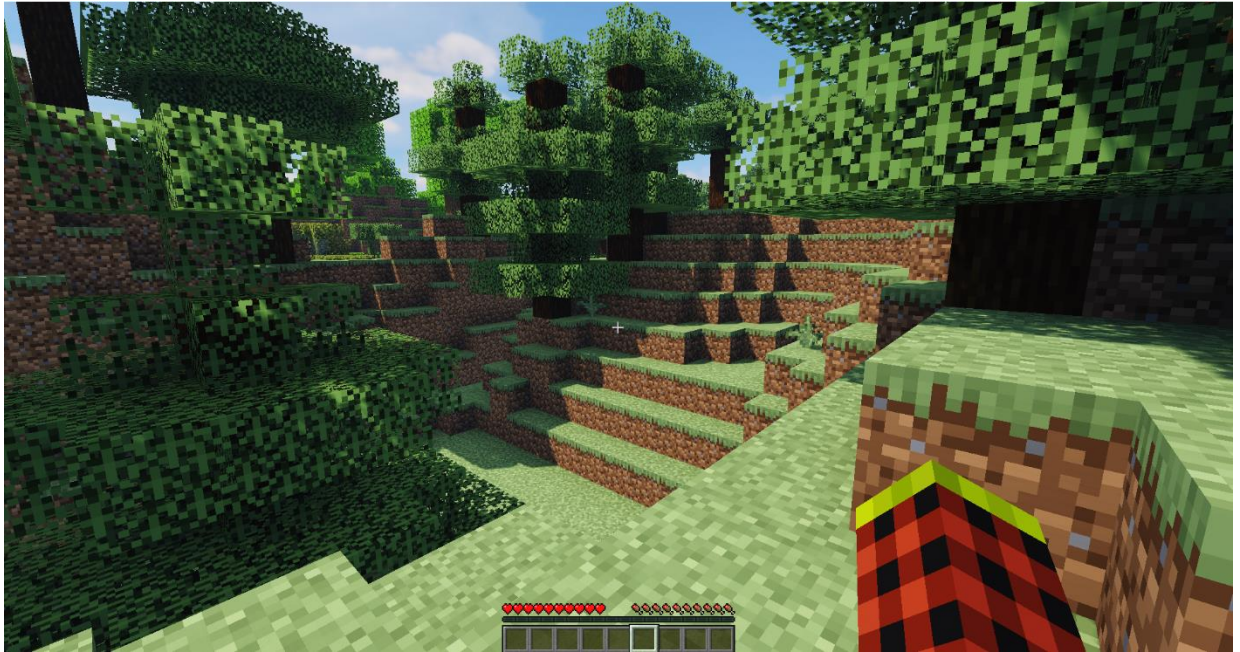


Рисунок 7.18 – Підключення до серверу

Так, в результаті всіх цих кроків отримали загальнодоступний сервер, до якого можуть під'єднуватися інші користувачі. Цей сервер може бути використаний для спільної гри, обміну ресурсами або будь-яких інших спільних дій в мережі. Цей процес надає можливість створення власного гравального оточення або спільного робочого середовища.

8. ПРИНЦИП РОБОТИ ЗАСТОСУНКУ

Перший крок в обробці зображень на сервері - це завантаження зображення користувачем. Воно робиться через веб-інтерфейс або мобільний додаток. Користувач вибирає зображення зі свого пристрою або надає URL-адресу зображення, яке потрібно обробити.

Процес:

- Користувач відправляє HTTP-запит на сервер з даними зображення (зазвичай в форматі multipart/form-data). Це може бути реалізовано через форму завантаження на веб-сторінці або через API-запит з мобільного додатка.

- Сервер приймає запит і зберігає зображення у тимчасовій директорії для подальшої обробки. Це відбувається за допомогою серверного програмного забезпечення, яке приймає та зберігає файли.

Після отримання зображення сервер починає процес його обробки. Обробка зображення може включати в себе різні етапи в залежності від вимог: зміна розміру, обрізка, фільтри, аналіз зображення, розпізнавання об'єктів тощо.

- Сервер викликає відповідні функції для обробки зображення. Це може бути реалізовано за допомогою мов програмування, таких як Java з використанням бібліотек.

Після завершення обробки зображення сервер зберігає результат у визначеному місці. Це бути відображення зображення в реальному часі в грі.

- Оброблене зображення зберігається разом з метаданими, такими як дата обробки, параметри обробки та інформація про користувача. Це дозволяє легко відслідковувати історію обробки та повертатися до попередніх версій зображень.

Після завершення обробки та збереження результатів сервертранслює оновлене зображення користувачю.

- Користувач отримує результат і може завантажити оброблене зображення на свій пристрій або переглянути його у веб-інтерфейсі. Це

дозволяє користувачам легко використовувати оброблені зображення для подальшого використання або зберігання.

Приклад серверного коду, розроблений на основі JavaFX, обробляє запити від користувача та здійснює необхідні зміни в зображеннях. Сервер приймає запити, обробляє їх і відправляє змінені зображення назад до клієнта.

```
app = Flask(__name__)

@app.route('/upload', methods=['POST'])
def upload_image():
    if 'file' not in request.files:
        return "No file part", 400
    file = request.files['file']
    if file.filename == '':
        return "No selected file", 400
    if file:
        image = Image.open(file.stream)
        processed_image = image.filter(ImageFilter.BLUR)
        img_io = io.BytesIO()
        processed_image.save(img_io, 'JPEG')
        img_io.seek(0)
        return send_file(img_io, mimetype='image/jpeg')

if __name__ == '__main__':
    app.run(debug=True)
```

Лістинг 8.1 – передача даних з програми до сервера

Цей приклад показує, як завантажити зображення, обробити його (застосувавши фільтр розмиття), і повернути користувачу оброблене зображення.

9. ІНСТРУКЦІЯ КОРИСТУВАЧА

В цьому розділі розказуються як різні користувачі використовують різні види інтерфейсів.

9.1 Програмний інтерфейс

Програмний інтерфейс являє собою декілька випадаючих списків, де кожен список співвідноситься певній команді рис. 9.1.

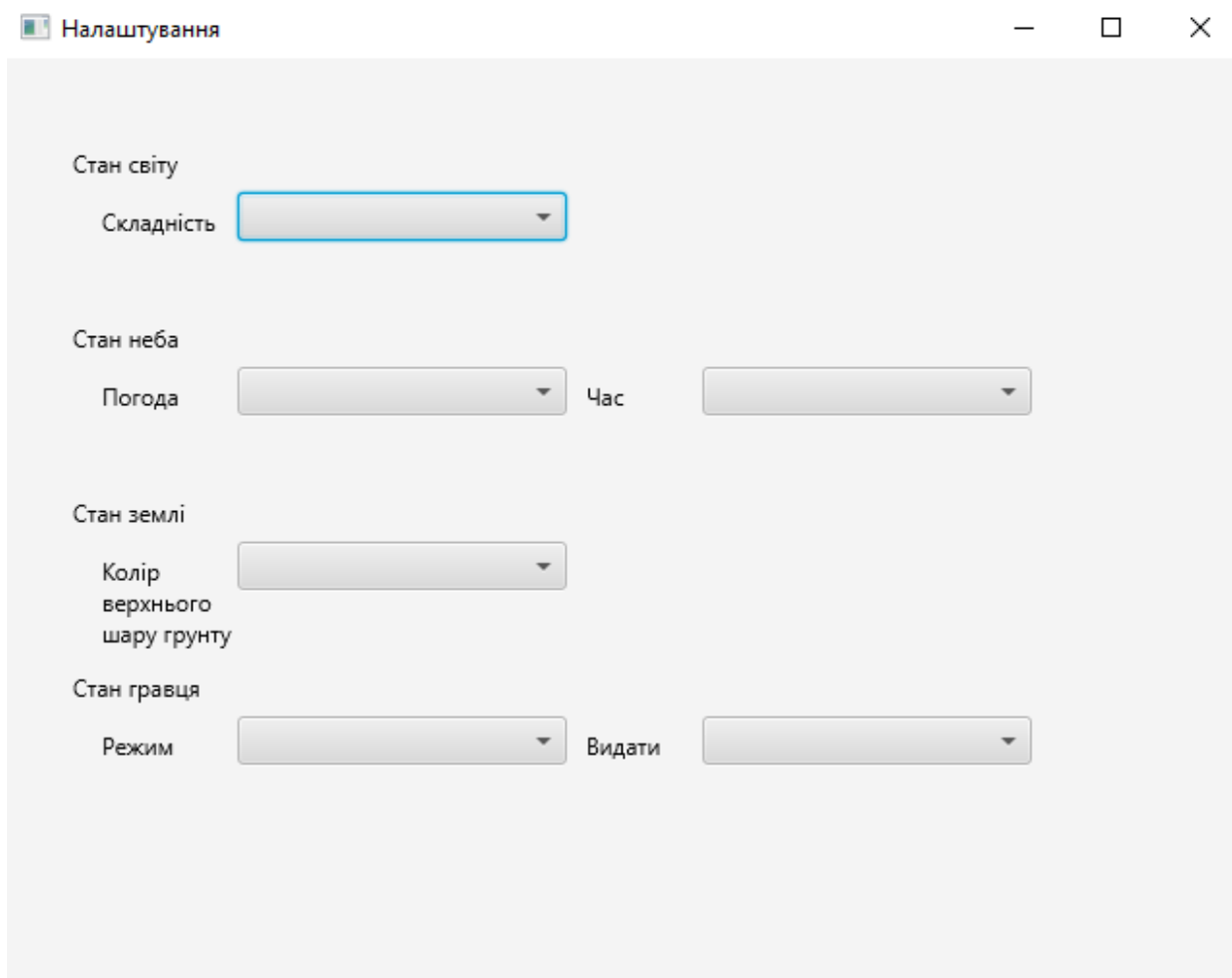


Рисунок 9.1 – Інтерфейс програми

Коли користувач вибирає щось з випадаючого списку надсилається запит до сервера на зміну середовища. Наприклад розглянемо зміну неба а

саме час. Як можна здогадатися час відповідає за положення сонця та освітленість. Коли виставлена день сонце лише сходить

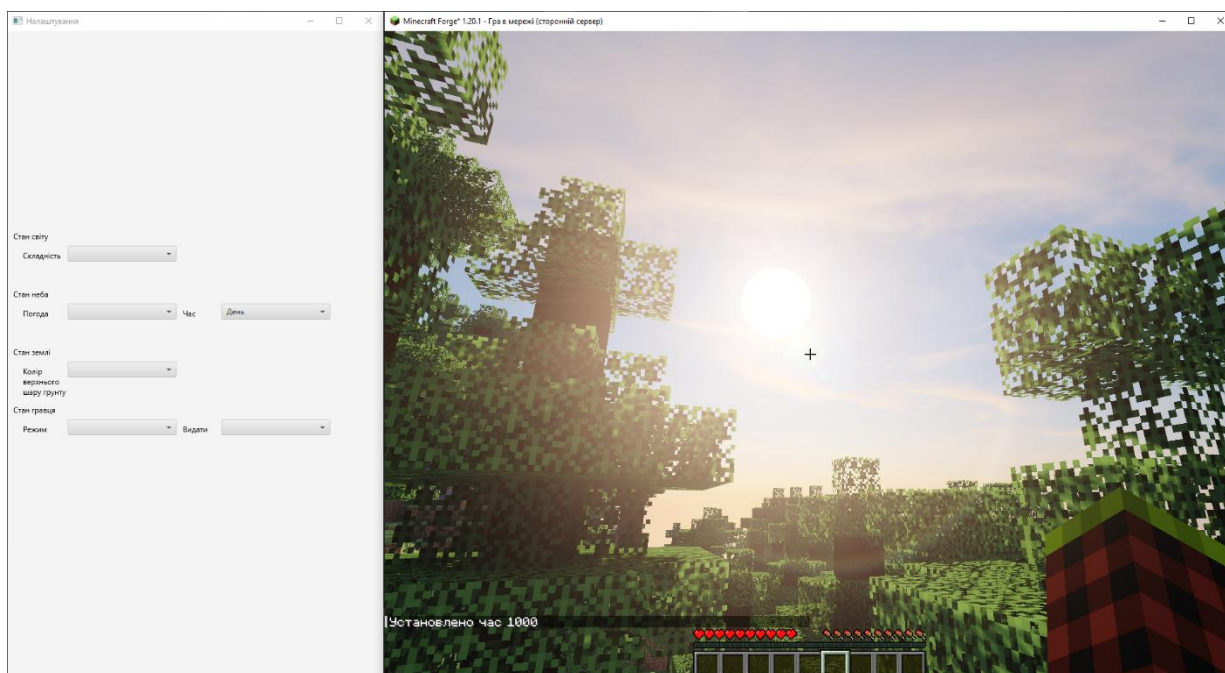


Рисунок 9.2 – Інтерфейс програми стан день

Коли вибраний полудень сонце знаходиться у зеніті.

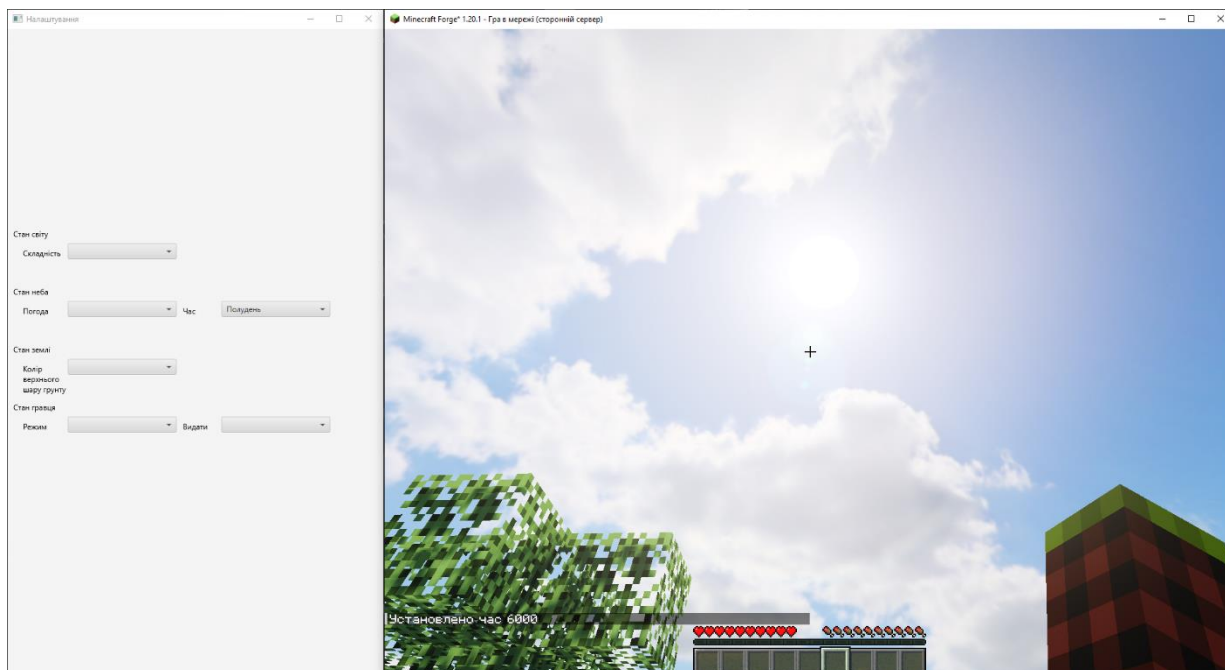


Рисунок 9.3 – Інтерфейс програми стан полудень

Коли вибраний ніч сонце взагалі немає але є місяць який тільки з'явився.

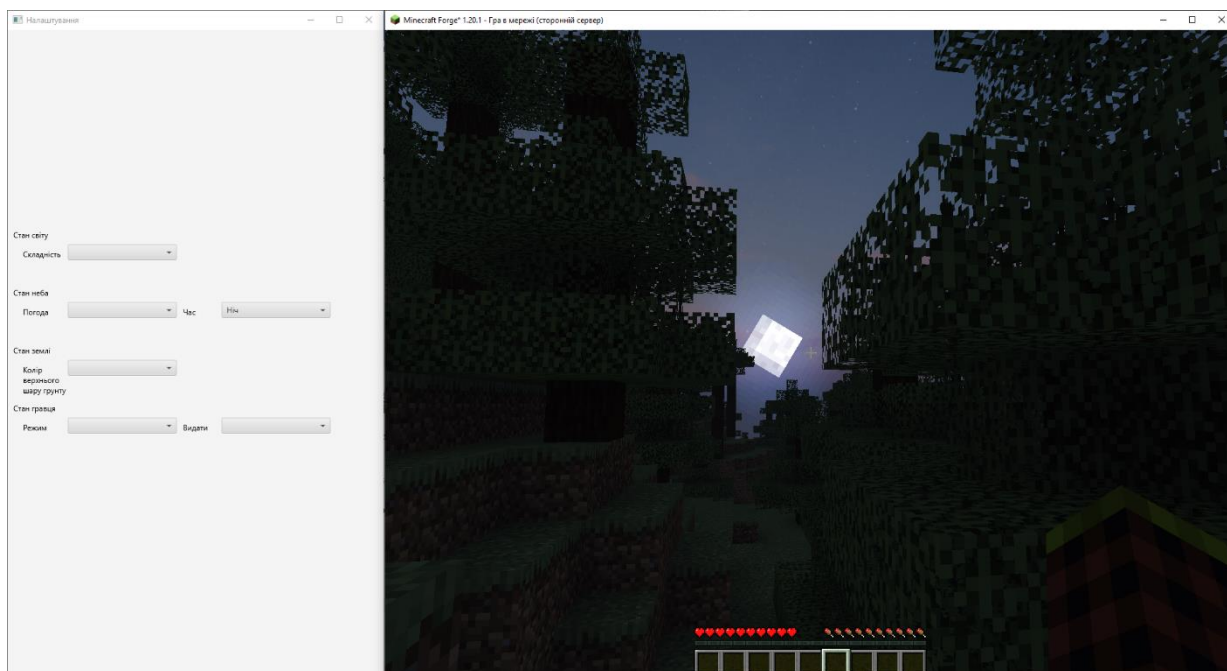


Рисунок 9.4 – Інтерфейс програми стан ніч

На останок залишилась лише опівніч, сонця також немає та місяць знаходиться у зеніті.

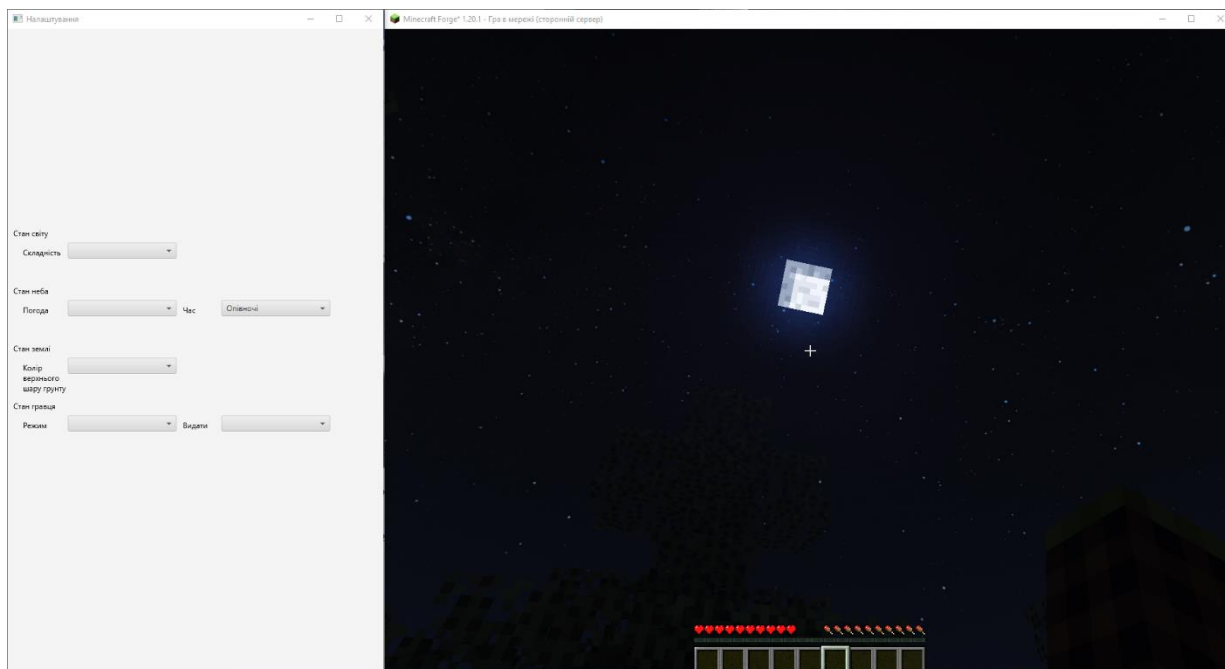


Рисунок 9.5 – Інтерфейс програми стан опівніч

9.2 Серверний інтерфейс

Серверний інтерфейс на багато складніший ніж програмний але він спрямован та адміністрування сервером. Серед всіх вкладок най більш цікавими є вкладки з консоллю та журналом.

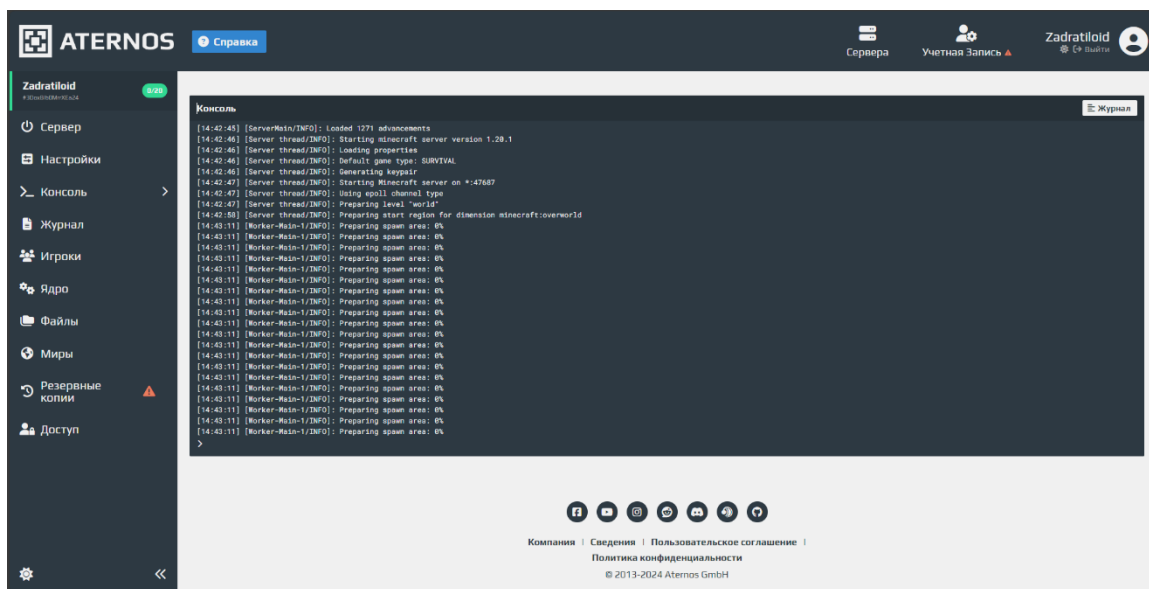


Рисунок 9.6 – Интерфейс консоли

На цій вкладці відображаються всі використані команди для зміни середовища в момент запуску сервера.

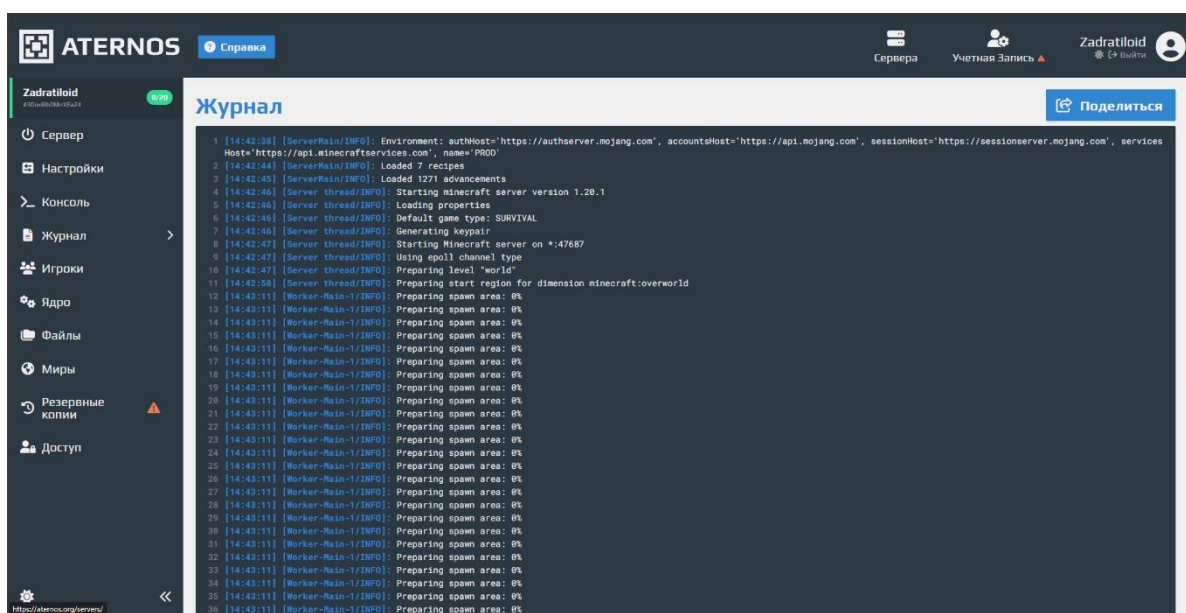


Рисунок 9.7 – Интерфейс журнала

Основною різницею між консоллю та журналом являється те що журнал записує все що відбувалося на сервері.

Також Aternos має зручний інтерфейс налаштування сервера таких як кількість людей та список людей які можуть приєднатися до сервера рис. 9.8.

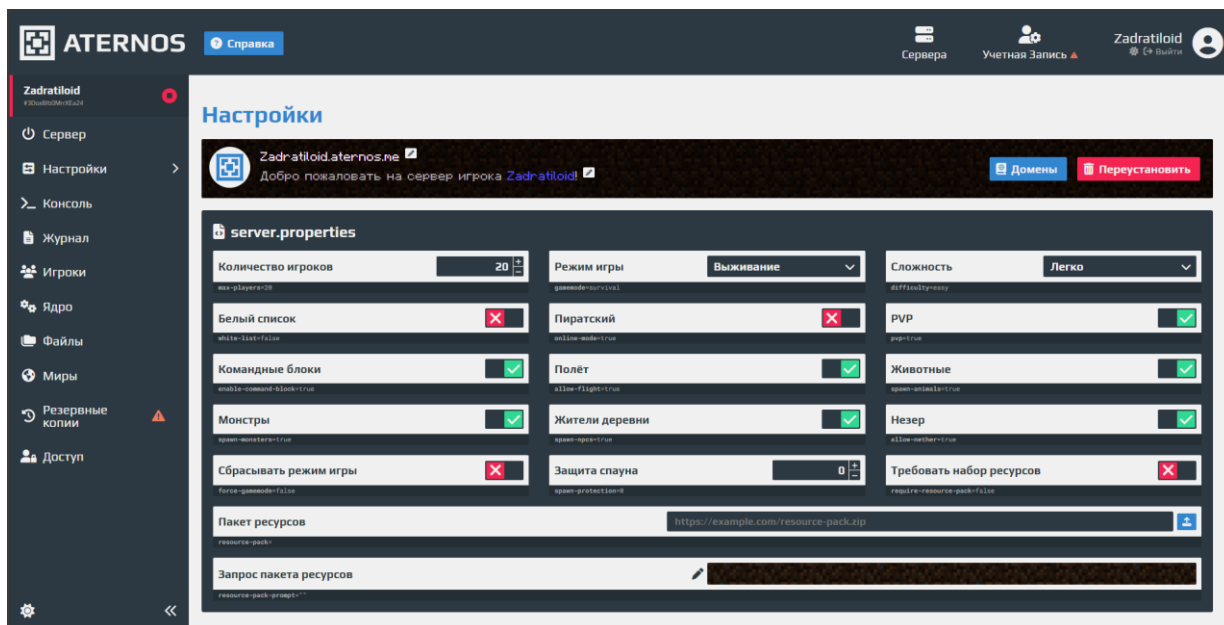


Рисунок 9.8 – Интерфейс Налаштування серверу

10. ПЕРСПЕКТИВИ

Найближчими перспективами є додавання нових можливостей та виправлення недоліків у вже створеному застосунку для редагування зображень у реальному часі. Перш за все, планується розширення функціоналу, щоб користувачі могли використовувати ще більше інструментів для редагування. Це може включати такі можливості, як розширений набір фільтрів, інструменти для роботи з шарами, можливості для малювання та багато іншого. Виправлення недоліків також буде одним з основних пріоритетів, щоб забезпечити максимальну стабільність та зручність використання застосунку.

На майбутнє перспективи стають ще більш амбітними. Однією з головних ідей є інтеграція розробленого застосунку в саму гру. Це дозволить гравцям редагувати зображення та графіку безпосередньо під час гри, що може значно підвищити їхній досвід і додати нові рівні творчості. Така інтеграція потребуватиме тісної співпраці з розробниками ігор та адаптації застосунку до специфічних вимог ігрового середовища.

Додатково, планується додавання підтримки додаткових команд від різних модифікацій. Це означає, що застосунок буде сумісним з широким спектром модифікацій, дозволяючи користувачам використовувати спеціальні команди та інструменти, які можуть бути розроблені третіми сторонами. Така відкритість до модифікацій забезпечить ще більшу гнучкість і можливості для кастомізації, що, без сумніву, буде оцінено користувачами.

ВИСНОВОК

Метою даної роботи було проєктування та реалізація набору інструментів для динамічної зміни зображень. В результаті створено потужний застосунок, який надає широкі можливості для редагування зображень у реальному часі як для кінцевих користувачів, так і для адміністраторів.

Реалізація програмного застосунку виконана за допомогою мови програмування Java у поєднанні з фреймворком JavaFX та інструментарієм Minecraft Development. Такий вибір технологій забезпечив високу продуктивність, стабільність та зручність у розробці. Архітектура системи відповідає шаблону MVC (Model-View-Controller) та використовує дворівневу архітектуру, що дозволило ефективно розподілити функціональні обов'язки між компонентами системи.

Застосунок дозволяє редагувати зображення, використовуючи вбудовані модулі, що забезпечує більшу гнучкість і розширюваність. Користувачі можуть додавати нові модулі або замінювати існуючі без змін основного коду програми, що знижує навантаження на систему та покращує продуктивність. В серверній частині системи реалізовано захист від несанкціонованого доступу до серверної консолі, що забезпечує високий рівень безпеки та захист конфіденційних даних за допомогою сучасних методів аутентифікації та авторизації.

Результатом роботи став потужний застосунок з можливістю дистанційного бездротового редагування динамічних зображень, що відкриває нові горизонти для творчості та співпраці, дозволяючи користувачам ефективно працювати з зображеннями з будь-якої точки з доступом до інтернету.

Майбутні перспективи розвитку цього застосунку передбачають додавання нових можливостей та виправлення недоліків:

- **Розширення функціоналу:** Додавання нових інструментів для редагування зображень, таких як малювання, робота з шарами, нові фільтри та ефекти.
- **Інтеграція з іграми:** Інтеграція застосунку в ігрові платформи, що дозволить гравцям редагувати зображення та графіку під час гри.
- **Підтримка модифікацій:** Додавання підтримки спеціальних команд від різних модифікацій, що забезпечить сумісність з широким спектром модифікацій та додаткових інструментів.
- **Покращення інтерфейсу:** Оптимізація та вдосконалення користувацького інтерфейсу для підвищення зручності та інтуїтивності використання.

Ці майбутні реалізації сприятимуть подальшому вдосконаленню застосунку, надаючи користувачам нові функції та можливості для творчої діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Клієнт-серверна архітектура. – Режим доступу: [Клієнт-серверна архітектура. \(qatestlab.com\)](http://qatestlab.com)
2. Документація по JavaFX. – Режим доступу: [Getting Started with JavaFX \(openjfx.io\)](http://openjfx.io)
3. Форум по бібліотеці Minecraft Development. – Режим доступу: [Issues · minecraft-dev/MinecraftDev · GitHub](https://github.com/minecraft-dev/MinecraftDev/issues)
4. Документція по ядру Paper. – Режим доступу [Home | PaperMC Docs](#)
5. Сайт створення серверів Aternos. – Режим доступу: [Сервера | Aternos | Бесплатный сервер Minecraft](#)
6. Документація по мові програмування Java. – Режим доступу: [Java Documentation - Get Started \(oracle.com\)](http://docs.oracle.com/javase/8/docs/guide/)
7. Стаття по особливості використання MVC моделей [MVC Framework Introduction - GeeksforGeeks](#)
8. Школа по вивченню мови програмування Java. – Режим доступу: [Java для початківців - курс програмування, навчання основ Джава з нуля, уроки на itProger – курси українською](#)

ДОДАТОК А

Створення візуального інтерфейсу застосунка

```

<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Text?>
<?import javafx.scene.control.ChoiceBox?>
<VBox alignment="CENTER" spacing="20.0"
xmlns:fx="http://javafx.com/fxml"

fx:controller="com.example.controller.HelloController">
    <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity"
minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0"
prefWidth="600.0" xmlns="http://javafx.com/javafx/21"
xmlns:fx="http://javafx.com/fxml/1">
        <children>
            <AnchorPane layoutX="0.0" prefHeight="90.0"
prefWidth="600.0">
                <children>
                    <Text layoutX="15.0" layoutY="20.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Стан світу"
wrappingWidth="73.0" />
                    <Text layoutX="30.0" layoutY="50.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Складність"
wrappingWidth="73.0" />
                    <ChoiceBox fx:id="Dif" layoutX="100.0"
layoutY="30.0" prefWidth="170.0" />
                </children>
            </AnchorPane>
            <AnchorPane layoutX="0.0" layoutY="90.0"
prefHeight="90.0" prefWidth="600.0">
                <children>
                    <Text layoutX="15.0" layoutY="20.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Стан неба"
wrappingWidth="73.0" />
                    <Text layoutX="30.0" layoutY="50.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Погода"
wrappingWidth="73.0" />
                    <ChoiceBox fx:id="weather"
layoutX="100.0" layoutY="30.0" prefWidth="170.0" />
                    <Text layoutX="280.0" layoutY="50.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Час"
wrappingWidth="73.0" />
                    <ChoiceBox fx:id="time" layoutX="340.0"
layoutY="30.0" prefWidth="170.0" />
                </children>
            </AnchorPane>
        </children>
    </AnchorPane>
</VBox>

```

Лістинг А.1 – Форми для зміни зображення

```

        </children>
    </AnchorPane>
    <AnchorPane layoutY="180.0" prefHeight="90.0"
prefWidth="600.0">
        <children>
            <Text layoutX="15.0" layoutY="20.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Стан землі"
wrappingWidth="73.0" />
            <Text layoutX="30.0" layoutY="50.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Колір верхнього шару
грунту" wrappingWidth="73.0" />
            <ChoiceBox fx:id="Biom" layoutX="100.0"
layoutY="30.0" prefWidth="170.0" />
        </children>
    </AnchorPane>
    <AnchorPane layoutY="270.0" prefHeight="90.0"
prefWidth="600.0">
        <children>
            <Text layoutX="15.0" layoutY="20.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Стан гравця"
wrappingWidth="73.0" />
            <Text layoutX="30.0" layoutY="50.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Режим"
wrappingWidth="73.0" />
            <ChoiceBox style="-fx-padding: var(10)"
fx:id="gamemode" layoutX="100.0" layoutY="30.0" prefWidth="170.0"
/>
            <Text layoutX="280.0" layoutY="50.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Видати"
wrappingWidth="73.0" />
            <ChoiceBox fx:id="give" layoutX="340.0"
layoutY="30.0" prefWidth="170.0" />
        </children>
    </AnchorPane>

```

Лістинг А.1, Лист 2

ДОДАТОК Б

Функції та змінні програмної реалізації застосунку

```

package com.example.controller;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.ChoiceBox;

import java.net.URL;
import java.util.ResourceBundle;

app = Flask(__name__)
login_manager = LoginManager()
login_manager.login_view = 'login'
login_manager.init_app(app)
app.secret_key = "fafwafwafawfawfwa"
class User(UserMixin):
    def __init__(self, clientno, full_name, phoneno, mail,
passwords, type):
        self.id = clientno
        self.full_name = full_name
        self.phoneno = phoneno
        self.mail = mail
        self.passwords = passwords
        self.type = type
def is_authenticated(self):
    return True

@app.route('/add_praise')
@login_required
def add_praise():
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f'update guns set Rent_praise = Rent_praise * 1.1 where
serial_number in (select serial_number from(SELECT Guns.Name,
Guns.Weapon_type, Guns.Caliber, Guns.Rent_praise,
Booking.Serial_number, COUNT(*) AS Popular FROM Booking INNER
JOIN Guns on Guns.Serial_number = Booking.Serial_number GROUP BY
Booking.Serial_number, Guns.Name, Guns.Weapon_type,

```

Лістинг Б.1 – передача даних з шаблон до сервера

```

Guns.Caliber, Guns.Rent_prise ORDER BY COUNT(*) DESC) as poT
where Popular = (SELECT max(Top.Popular) FROM (SELECT
Guns.Name, Guns.Weapon_type, Guns.Caliber, Guns.Rent_prise,
Booking.Serial_number, COUNT(*) AS Popular FROM Booking INNER
JOIN Guns on Guns.Serial_number = Booking.Serial_number GROUP BY
Booking.Serial_number, Guns.Name, Guns.Weapon_type,
Guns.Caliber, Guns.Rent_prise ORDER BY COUNT(*) DESC) AS Top));
')
    cursor.execute(sql)
    conn.close()
    return redirect('adminpage')

@app.route('/minecraft')
def minecraft():

    return redirect('minecraft')

def get_user(phoneno):
    buf = "ClientNotLogin"

    if is_authenticated(current_user):
        try:
            buf = current_user.type
        except:
            print(3)
    else:
        buf = ''

    conn = psycopg2.connect(log[buf])
    cursor = conn.cursor()
    cursor.execute(f'''SELECT * FROM clients where phoneno =
\ '{phoneno}' \ ''')
    user_data = cursor.fetchone()
    print(user_data)
    curr_user = User(user_data[0], user_data[1], user_data[2],
user_data[3], user_data[4], user_data[5])
    print(curr_user.phoneno)
    return curr_user

def get_user_by_id(user_id):
    buf = "ClientNotLogin"

```

```

if is_authenticated(current_user):
    try:
        buf = current_user.type
    except:
        print(3)
else:
    buf = ''

conn = psycopg2.connect(log[buf])
cursor = conn.cursor()
cursor.execute(f'''SELECT * FROM clients WHERE
clientno=\'{user_id}\';''')
user_data = cursor.fetchone()
user = User(user_data[0], user_data[1], user_data[2],
user_data[3], user_data[4], user_data[5])
return user

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect('login')

@login_manager.unauthorized_handler
def unauthorized():
    return 'Необходимо войти в систему'

@login_manager.user_loader
def load_user(user_id):
    return get_user_by_id(user_id)

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username2 = request.form['PhoneNo']
        password = request.form['Passwords']
        password = hashlib.md5(password.encode('utf-
8')).hexdigest()
        print(username2)

```

```

    print(get_user(username2))
    user = get_user(username2)
    if user and password == user.passwords:
        login_user(user)
        print(user.type)
        return redirect('home')
    else:
        return 'Неверное имя пользователя или пароль'
elif request.method == 'GET':
    return render_template('login.html')
return render_template('login.html')

@app.route('/')
@app.route('/home')
def index():
    buf = "ClientNotLogin"

    if is_authenticated(current_user):
        try:
            buf = current_user.type
        except:
            print(3)
    else:
        buf = ''

    conn = psycopg2.connect(log[buf])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
    сортировки

    sql = ("select worker.full_name from Worker where Job_title
= 'Instructor'")
    cursor.execute(sql)
    data = cursor.fetchall()

    sql = ("SELECT Guns.Name, Guns.Weapon_type, Guns.Caliber,
Guns.Rent_prise, Booking.Serial_number, COUNT(*) AS Popular FROM
Booking INNER JOIN Guns on Guns.Serial_number =
Booking.Serial_number GROUP BY Booking.Serial_number, Guns.Name,
Guns.Weapon_type, Guns.Caliber, Guns.Rent_prise ORDER BY
COUNT(*) DESC LIMIT 5")

```

```

        cursor.execute(sql)
        datag = cursor.fetchall()

        sql = ("select TrackNo, Rental_price from Tracks where
Malfunction = 'properly'")
        cursor.execute(sql)
        datar = cursor.fetchall()

        return render_template("index.html", data=data, datag=datag,
datar=datar, user_type=buf)

@app.route('/guns')
def guns():
    buf = "ClientNotLogin"

    if is_authenticated(current_user):
        try:
            buf = current_user.type
        except:
            print(3)
    else:
        buf = ''

    conn = psycopg2.connect(log[buf])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f'SELECT G.Name, G.Weapon_type, Inventory.Inventory,
Rent_prise, Breaking '
          f'FROM Inventory inner join Guns G on
Inventory.InventoryNo = G.Caliber '
          f''WHERE G.Breaking = 'Whole' ''')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()

    return render_template("guns.html", data=data,
user_type=buf)

@app.route('/guns_worker')

```

```

def guns_worker():
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f'SELECT G.serial_number, G.Name, G.Weapon_type,
Inventory.Inventory, Rent_prise, Breaking, cause_of_failure '
          f'FROM Inventory inner join Guns G on
Inventory.InventoryNo = G.Caliber')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("guns_worker.html", data=data,
user_type=current_user.type)

@app.route('/userspage')
def userspage():
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f'SELECT * '
          f'FROM Clients '
          f"where clientno = '{current_user.id}' ")
    cursor.execute(sql)
    data = cursor.fetchall()

    sql = (f'select booking.bookingno, C.Full_name, W.Full_name,
booking.date_and_time_of_booking_start,
Booking.Date_and_time_weapons_were_issued,
Booking.Date_and_time_receipt_of_weapons,
booking.date_and_time_of_booking_end, Guns.Name,
Booking.Number_of_rounds,
Booking.The_number_of_not_fired_rounds, T.TrackNo,
Booking.Status, NotFull_price, Full_price FROM Booking INNER
JOIN Guns on Guns.Serial_number = Booking.Serial_number inner
join Worker W on W.WorkerNo = Booking.WorkerNo inner join Tracks
T on T.TrackNo = Booking.Track_number inner join Clients C on
C.ClientNo = Booking.ClientNo '
          f"where booking.ClientNo = '{current_user.id}' "
          f'order by booking.date_and_time_of_booking_start

```

```

desc ')
    cursor.execute(sql)
    datar = cursor.fetchall()

    sql = (f''' select * FROM Complaints where clientno =
'{current_user.id}' order by date_and_time_of_application DESC
''')
    cursor.execute(sql)
    datac = cursor.fetchall()

    conn.close()
    return render_template("userspage.html", data=data,
datar=datar, datac=datac, user_type=current_user.type)

@app.route('/users_admin')
def users_admin():
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f'SELECT * '
          f'FROM Clients '
          f'order by clientno ')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("users_admin.html", data=data,
user_type=current_user.type)

@app.route('/users_worker')
def users_worker():
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f'SELECT clientno, full_name, phoneno, mail '
          f'FROM Clients '
          f'order by clientno ')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()

```

```

        return render_template("users_worker.html", data=data,
user_type=current_user.type)

@app.route('/register', methods=['GET', 'POST'])
def registr():
    if request.method == 'POST':
        full_name_c_web = request.form['full_name']
        phoneno_web = request.form['PhoneNo']
        Mail_web = request.form['Mail']
        Passwords_web =
hashlib.md5(request.form['Passwords'].encode('utf-
8')).hexdigest()

        buf = "ClientNotLogin"

        if is_authenticated(current_user):
            try:
                buf = current_user.type
            except:
                print(3)
        else:
            buf = ''

        conn = psycopg2.connect(log[buf])
        cursor = conn.cursor()
        sql = ("INSERT INTO Clients (Full_name, PhoneNo,
Mail, Passwords, title) "
            f'''VALUES ('{full_name_c_web}',
'{{phoneno_web}}', '{{Mail_web}}', '{{Passwords_web}}', 'Client' )'''
        cursor.execute(sql)
        conn.commit()
        conn.close()
        return redirect(url_for('login'))
    else:
        return render_template("register.html")

@app.route('/inventory_worker')
def inventory_worker():
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления

```

```

сортировки
    sql = (f'SELECT * '
          f'FROM Inventory '
          f'order by inventoryno')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("inventory_worker.html", data=data,
user_type=current_user.type)

```

```

@app.route('/inventory')
def inventory():
    buf = "ClientNotLogin"

    if is_authenticated(current_user):
        try:
            buf = current_user.type
        except:
            print(buf)
    else:
        buf = ''

    conn = psycopg2.connect(log[buf])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f'SELECT inventoryno, inventory,
price_per_piece_lease '
          f'FROM Inventory '
          f'order by inventoryno')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()

    return render_template("inventory.html", data=data,
user_type=buf)

```

```

@app.route('/stock_replenishment')
def stock_replenishment():
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления

```

```

сортировки
    sql = (f'select deliveryno, I.inventory,
Stock_replenishment.Quantity, replenishment_date, W.full_name '
        f'from stock_replenishment inner join Worker W on
W.WorkerNo = stock_replenishment.WorkerNo inner join Inventory I
on I.InventoryNo = stock_replenishment.InventoryNo')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("stock_replenishment.html",
data=data, user_type=current_user.type)

@app.route('/add_replenishment', methods=['GET', 'POST'])
def add_replenishment():
    if request.method == 'POST':
        inventoryno_web = request.form['inventoryno']
        quantity_web = request.form['quantity']
        workerno_web = request.form['workerno']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()

        sql = ("insert into Stock_replenishment (InventoryNo,
Quantity, Replenishment_date, WorkerNo) "
            f'''VALUES ('{inventoryno_web}' ,
'{{quantity_web}}' , '{{datetime.now().date():%Y-%m-%d}}',
'{{workerno_web}}' )''')
        cursor.execute(sql)
        conn.commit()
        conn.close()
        return redirect(url_for('stock_replenishment'))
    else:
        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = ("select worker.workerno, worker.full_name from
Worker")
        cursor.execute(sql)
        data = cursor.fetchall()
        sql2 = ("SELECT InventoryNo, inventory FROM Inventory
where InventoryNo >= 3")
        cursor.execute(sql2)
        datain = cursor.fetchall()

```

```

        conn.close()
        return render_template("add_replenishment.html",
data=data, datain=datain, user_type=current_user.type)

@app.route('/workerpage')
def workerpage():
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f"SELECT Clients.Full_name, count(Booking.ClientNo),
rank() OVER (ORDER BY count(Booking.ClientNo) DESC ) as
Rank_count_of_visits,
sum(Booking.The_number_of_not_fired_rounds), rank() OVER (ORDER
BY sum(Booking.The_number_of_not_fired_rounds) DESC ) as
Rank_sum_of_not_fired_rounds, sum(Booking.Full_price), rank()
OVER (ORDER BY sum(Booking.Full_price) DESC ) as
Rank_sum_of_full_price FROM Booking INNER JOIN Clients on
Clients.ClientNo = Booking.ClientNo WHERE current_timestamp -
interval '1 month'<= Date_and_time_of_booking_start GROUP BY
Clients.Full_name ")
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("workerpage.html", data=data,
user_type=current_user.type)

@app.route('/adminpage')
def adminpage():
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f"SELECT Clients.Full_name, count(Booking.ClientNo),
rank() OVER (ORDER BY count(Booking.ClientNo) DESC ) as
Rank_count_of_visits,
sum(Booking.The_number_of_not_fired_rounds), rank() OVER (ORDER
BY sum(Booking.The_number_of_not_fired_rounds) DESC ) as
Rank_sum_of_not_fired_rounds, sum(Booking.Full_price), rank()
OVER (ORDER BY sum(Booking.Full_price) DESC ) as
Rank_sum_of_full_price FROM Booking INNER JOIN Clients on
Clients.ClientNo = Booking.ClientNo WHERE current_timestamp -

```

```

interval '1 month'<= Date_and_time_of_booking_start GROUP BY
Clients.Full_name ")
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("adminpage.html", data=data,
user_type=current_user.type)

@app.route('/booking')
def booking():
    buf = "ClientNotLogin"

    if is_authenticated(current_user):
        try:
            buf = current_user.type
        except:
            print(3)
    else:
        buf = ''

    conn = psycopg2.connect(log[buf])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f''' select booking.bookingno, W.Full_name,
booking.date_and_time_of_booking_start,
booking.date_and_time_of_booking_end, Guns.Name, T.TrackNo FROM
Booking INNER JOIN Guns on Guns.Serial_number =
Booking.Serial_number inner join Worker W on W.WorkerNo =
Booking.WorkerNo inner join Tracks T on T.TrackNo =
Booking.Track_number inner join Clients C on C.ClientNo =
Booking.ClientNo where Status = 'Active' order by
booking.date_and_time_of_booking_start desc ''')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()

    return render_template("booking.html", data=data,
user_type=buf)

@app.route('/rouds_worker')
def rouds_worker():

```

```

    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f''' select * FROM Tracks ''' )
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("rouds_worker.html", data=data,
user_type=current_user.type)

@app.route('/workers_admin')
def workers_admin():
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f''' select * FROM worker order by workerno''' )
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("workers_admin.html", data=data,
user_type=current_user.type)

@app.route('/complaints_admin')
def complaints_admin():
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f'select ComplaintNo,
C.Full_name,date_and_time_of_application, text_of_complaint '
        f'FROM Complaints INNER join Clients C on
Complaints.ClientNo = C.ClientNo '
        f'order by date_and_time_of_application DESC ')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("complaints_admin.html", data=data,
user_type=current_user.type)

```

```

@app.route('/booking_worker', methods=['GET', 'POST'])
def booking_worker():
    if request.method == 'POST':
        full_name_c = request.form['full_name_c']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        # Создайте SQL-запрос с учетом столбца и направления
сортировки
        sql = (f''' select booking.bookingno, C.Full_name,
W.Full_name, booking.date_and_time_of_booking_start,
Booking.Date_and_time_weapons_were_issued,
Booking.Date_and_time_receipt_of_weapons,
booking.date_and_time_of_booking_end, Guns.Name,
Booking.Number_of_rounds,
Booking.The_number_of_not_fired_rounds, T.TrackNo,
Booking.Status, NotFull_price, Full_price FROM Booking INNER
JOIN Guns on Guns.Serial_number = Booking.Serial_number inner
join Worker W on W.WorkerNo = Booking.WorkerNo inner join Tracks
T on T.TrackNo = Booking.Track_number inner join Clients C on
C.ClientNo = Booking.ClientNo order by
booking.date_and_time_of_booking_start desc where C.Full_name
= '{full_name_c}' ''')
        cursor.execute(sql)
        data = cursor.fetchall()
        conn.close()
        return render_template("booking_worker.html", data=data)
    else:
        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        # Создайте SQL-запрос с учетом столбца и направления
сортировки
        sql = (f''' select booking.bookingno, C.Full_name,
W.Full_name, booking.date_and_time_of_booking_start,
Booking.Date_and_time_weapons_were_issued,
Booking.Date_and_time_receipt_of_weapons,
booking.date_and_time_of_booking_end, Guns.Name,
Booking.Number_of_rounds,
Booking.The_number_of_not_fired_rounds, T.TrackNo,
Booking.Status, NotFull_price, Full_price FROM Booking INNER
JOIN Guns on Guns.Serial_number = Booking.Serial_number inner
join Worker W on W.WorkerNo = Booking.WorkerNo inner join Tracks
T on T.TrackNo = Booking.Track_number inner join Clients C on

```

```

C.ClientNo = Booking.ClientNo order by
booking.date_and_time_of_booking_start desc  '')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("booking_worker.html", data=data,
user_type=current_user.type)

@app.route('/booking_book', methods=['GET', 'POST'])
def booking_book():
    if request.method == 'POST':
        full_name_w = request.form['full_name_w']
        booking_start = request.form['booking_start']
        booking_end = request.form['booking_end']
        guns = request.form['guns']
        Rounds = request.form['Rounds']
        roud = request.form['roud']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()

        sql = ("insert into Booking (ClientNo, WorkerNo,
Date_and_time_of_booking_start, Date_and_time_of_booking_end,
Serial_number, Number_of_rounds, the_number_of_not_fired_rounds,
Track_number, Status) "
            f''VALUES ('{current_user.id}' , '{full_name_w}'
, '{booking_start}', '{booking_end}', '{guns}', '{Rounds}',
'0','{roud}', 'Active' )''')
        cursor.execute(sql)
        conn.commit()
        conn.close()

        return redirect(url_for('booking'))
    else:
        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = ("select worker.workerno, worker.full_name from
Worker where Job_title = 'Instructor' ")
        cursor.execute(sql)
        dataw = cursor.fetchall()

        sql3 = ("SELECT Serial_number, Name FROM Guns where

```

```

breaking = 'Whole'")
    cursor.execute(sql3)
    datag = cursor.fetchall()
    sql4 = ("select TrackNo from Tracks where Malfunction =
'properly'")
    cursor.execute(sql4)
    datar = cursor.fetchall()
    conn.close()

    return render_template("booking_book.html", dataw=dataw,
datag=datag, datar=datar, user_type=current_user.type)

@app.route('/editbooking/<id>', methods=['GET', 'POST'])
def editbooking(id):
    if request.method == 'POST':
        gun_start_web = request.form['gun_start']
        gun_end_web = request.form['gun_end']
        round_web = request.form['round']
        status_web = request.form['status']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = (f''' update booking set
Date_and_time_weapons_were_issued = '{gun_start_web}',
Date_and_time_receipt_of_weapons = '{gun_end_web}',
The_number_of_not_fired_rounds = '{round_web}', Status =
'{status_web}' where bookingno = '{id}' ''')
        cursor.execute(sql)
        conn.commit()
        conn.close()
        return redirect(url_for('booking_worker'))
    else:
        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        # Создайте SQL-запрос с учетом столбца и направления
сортировки
        sql = (f''' select booking.bookingno,
Booking.Date_and_time_weapons_were_issued,
Booking.Date_and_time_receipt_of_weapons,
Booking.The_number_of_not_fired_rounds, Booking.Status FROM
Booking WHERE bookingno = '{id}' ''')
        cursor.execute(sql)

```

```

        data = cursor.fetchall()
        conn.close()
        return render_template("editbooking.html",
booking=data[0], user_type=current_user.type)

@app.route('/editguns_worker/<Ser_nam>', methods=['GET',
'POST'])
def editguns_worker(Ser_nam):
    if request.method == 'POST':
        name_web = request.form['name']
        type_web = request.form['types']
        breaking_web = request.form['breaking']
        cause_web = request.form['cause']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = (f''update guns set name = '{name_web}',
Weapon_type = '{type_web}', Breaking = '{breaking_web}',
cause_of_failure = '{cause_web}' WHERE Serial_number =
'{Ser_nam}' '')
        cursor.execute(sql)
        conn.commit()
        conn.close()
        return redirect(url_for('guns_worker'))
    else:
        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        # Создайте SQL-запрос с учетом столбца и направления
сортировки
        sql = (f'' select serial_number, name, Weapon_type,
Breaking, cause_of_failure from guns WHERE serial_number =
'{Ser_nam}' '')
        cursor.execute(sql)
        data = cursor.fetchall()
        conn.close()
        return render_template("editguns_worker.html",
guns=data[0], user_type=current_user.type)

@app.route('/editworker_admin/<id>', methods=['GET', 'POST'])
def editworker_admin(id):
    if request.method == 'POST':
        full_name_web = request.form['name']

```

```

phone_web = request.form['phone']
address_web = request.form['address']
title_web = request.form['title']

conn = psycopg2.connect(log[current_user.type])
cursor = conn.cursor()
sql = (f'''update worker set full_name =
'{full_name_web}', phoneno = '{phone_web}', address =
'{address_web}', job_title = '{title_web}' WHERE workerno =
'{id}' ''')
#sql = (f'''update worker set full_name =
'{full_name_web}' WHERE workerno = '{id}' ''')

cursor.execute(sql)
conn.commit()
conn.close()
return redirect(url_for('workers_admin'))
else:
conn = psycopg2.connect(log[current_user.type])
cursor = conn.cursor()
# Создайте SQL-запрос с учетом столбца и направления
сортировки
sql = (f''' select * from worker WHERE workerno = '{id}'
''')

cursor.execute(sql)
data = cursor.fetchall()
conn.close()
return render_template("editworker_admin.html",
worker=data[0], user_type=current_user.type)

@app.route('/editguns_admin/<Ser_nam>', methods=['GET', 'POST'])
def editguns_admin(Ser_nam):
    if request.method == 'POST':
        name_web = request.form['name']
        type_web = request.form['types']
        rent_prise_web = request.form['rent_prise']
        breaking_web = request.form['breaking']
        cause_web = request.form['cause']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = (f'''update guns set name = '{name_web}',
Weapon_type = '{type_web}', Rent_prise = '{rent_prise_web}',

```

```

Breaking = '{breaking_web}', cause_of_failure = '{cause_web}'
WHERE Serial_number = '{Ser_nam}' ''')
    cursor.execute(sql)
    conn.commit()
    conn.close()
    return redirect(url_for('guns_admin'))
else:
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f''' select serial_number, name, Weapon_type,
Rent_prise, Breaking, cause_of_failure from guns WHERE
serial_number = '{Ser_nam}' ''')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("editguns_admin.html",
guns=data[0], user_type=current_user.type)

@app.route('/editusers_user/<id>', methods=['GET', 'POST'])
def editusers_user(id):
    if request.method == 'POST':
        name_web = request.form['name']
        phone_web = request.form['phone']
        email_web = request.form['email']
        password_web = current_user.passwords

        if request.form['password'] == '':
            pass
        else:
            password_web =
hashlib.md5(request.form['password'].encode('utf-
8')).hexdigest()

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = (f'''update clients set full_name = '{name_web}',
phoneno = '{phone_web}', mail = '{email_web}', passwords =
'{password_web}' WHERE clientno = '{id}' ''')
        cursor.execute(sql)
        conn.commit()

```

```

        conn.close()
        return redirect(url_for('userspage'))
    else:
        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        # Создайте SQL-запрос с учетом столбца и направления
        сортировки
        sql = (f''' select clientno, full_name, phoneno, mail,
passwords from clients WHERE clientno = '{id}' '''
        cursor.execute(sql)
        data = cursor.fetchall()
        conn.close()
        return render_template("editusers_user.html",
user=data[0], user_type=current_user.type)

@app.route('/editusers_admin/<id>', methods=['GET', 'POST'])
def editusers_admin(id):
    if request.method == 'POST':
        name_web = request.form['name']
        phone_web = request.form['phone']
        email_web = request.form['email']
        type_web = request.form['type']
        password_web = current_user.passwords

        if request.form['password'] == '':
            pass
        else:
            password_web =
hashlib.md5(request.form['password'].encode('utf-
8')).hexdigest()

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = (f'''update clients set full_name = '{name_web}',
phoneno = '{phone_web}', mail = '{email_web}', passwords =
'{password_web}', title = '{type_web}' WHERE clientno = '{id}'
''')
        cursor.execute(sql)
        conn.commit()
        conn.close()
        return redirect(url_for('users_admin'))

```

```

else:
    conn = psycopg2.connect(log[current_user.type])
    cursor = conn.cursor()
    # Создайте SQL-запрос с учетом столбца и направления
сортировки
    sql = (f''' select clientno, full_name, phoneno, mail,
passwords, title from clients WHERE clientno = '{id}' ''')
    cursor.execute(sql)
    data = cursor.fetchall()
    conn.close()
    return render_template("editusers_admin.html",
user=data[0], user_type=current_user.type)

@app.route('/editrouds_worker/<Rnom>', methods=['GET', 'POST'])
def editrouds_worker(Rnom):
    if request.method == 'POST':
        malfun_web = request.form['malfun']
        cause_of_malfun_web = request.form['cause_of_malfun']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = (f'''update tracks set malfunction =
'{malfun_web}', cause_of_malfunction = '{cause_of_malfun_web}'
WHERE trackno = '{Rnom}' ''')
        cursor.execute(sql)
        conn.commit()
        conn.close()
        return redirect(url_for('rouds_worker'))
    else:
        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        # Создайте SQL-запрос с учетом столбца и направления
сортировки
        sql = (f''' select * from tracks where trackno =
'{Rnom}' ''')
        cursor.execute(sql)
        data = cursor.fetchall()
        conn.close()
        return render_template("editrouds_worker.html",
rouds=data[0], user_type=current_user.type)

```

```

@app.route('/editrouds_admin/<Rnom>', methods=['GET', 'POST'])
def editrouds_admin(Rnom):
    if request.method == 'POST':
        rent_prises_web = request.form['rent_prises']
        malfun_web = request.form['malfun']
        cause_of_malfun_web = request.form['cause_of_malfun']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = (f'''update tracks set rental_price =
        '{rent_prises_web}', malfunction = '{malfun_web}',
        cause_of_malfunction = '{cause_of_malfun_web}' WHERE trackno =
        '{Rnom}' ''')
        cursor.execute(sql)
        conn.commit()
        conn.close()
        return redirect(url_for('rouds_worker'))
    else:
        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        # Создайте SQL-запрос с учетом столбца и направления
сортировки
        sql = (f''' select * from tracks where trackno =
        '{Rnom}' ''')
        cursor.execute(sql)
        data = cursor.fetchall()
        conn.close()
        return render_template("editrouds_admin.html",
        rouds=data[0], user_type=current_user.type)

@app.route('/editinventory/<Rnom>', methods=['GET', 'POST'])
def editinventory(Rnom):
    if request.method == 'POST':
        inventory_web = request.form['inventory']
        quantity_web = request.form['quantity']
        price_web = request.form['price']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = (f'''update inventory set inventory =
        '{inventory_web}', quantity = '{quantity_web}',
        price_per_piece_lease = '{price_web}' WHERE inventoryno =

```

```

    '{Rnom}' ''')
        cursor.execute(sql)
        conn.commit()
        conn.close()
        return redirect(url_for('inventory_worker'))
    else:
        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        # Создайте SQL-запрос с учетом столбца и направления
сортировки
        sql = (f''' select * from inventory where inventoryno =
    '{Rnom}' ''')
        cursor.execute(sql)
        data = cursor.fetchall()
        conn.close()
        return render_template("editinventory.html",
rouds=data[0], user_type=current_user.type)

@app.route('/newinstructor', methods=['POST', 'GET'])
def newinstructor():
    if request.method == 'POST':
        full_name_web = request.form['full_name']
        passport_data_web = request.form['passport_data']
        phoneno_web = request.form['phoneno']
        address_web = request.form['address']
        job_title_web = request.form['job_title']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = ("INSERT INTO worker (Full_name, Passport_data,
PhoneNo, Address, Date_of_hiring, Job_title) "
            f'''VALUES ('{full_name_web}' ,
    '{passport_data_web}' , '{phoneno_web}' , '{address_web}' ,
    '{datetime.now().date():%Y-%m-%d}', '{job_title_web}' )''')
        cursor.execute(sql)
        conn.commit()
        conn.close()
        return redirect(url_for('workers_admin'))
    else:
        return render_template("newinstructor.html",
user_type=current_user.type)

```

```

@app.route('/newgun', methods=['POST', 'GET'])
def newgun():
    if request.method == 'POST':
        name_web = request.form['name']
        type_web = request.form['Type']
        calibr_web = request.form['Calibr']
        serial_web = request.form['serial']
        rent_web = request.form['rent']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = ("insert into Guns (Name, Weapon_type, Caliber,
Serial_number, Rent_prise,Breaking) "
            f'''VALUES ('{name_web}', '{type_web}',
'{calibr_web}', '{serial_web}', '{rent_web}', 'Whole' )''')
        cursor.execute(sql)
        conn.commit()
        conn.close()
        return redirect(url_for('guns_worker'))
    else:
        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        # Создайте SQL-запрос с учетом столбца и направления
сортировки
        sql = (f' SELECT InventoryNo, inventory FROM Inventory
where InventoryNo >= 3 ')
        cursor.execute(sql)
        data = cursor.fetchall()
        conn.close()
        return render_template("newgun.html", data=data,
user_type=current_user.type)

@app.route('/newrouds', methods=['POST', 'GET'])
def newrouds():
    if request.method == 'POST':
        rent_web = request.form['rent']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = ("insert into Tracks
(Rental_price,Malfunction) "

```

```

        f''values ('{rent_web}', 'properly') '''
    cursor.execute(sql)
    conn.commit()
    conn.close()
    return redirect(url_for('rouds_worker'))
else:
    return render_template("newrouds.html",
user_type=current_user.type)

@app.route('/newcomplaints', methods=['POST', 'GET'])
def newcomplaints():
    if request.method == 'POST':
        compla_web = request.form['compla']
        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = ("insert into Complaints (ClientNo,
Date_and_time_of_application, Text_of_Complaint) "
            f''values ('{current_user.id}',
'{datetime.now().date():%Y-%m-%d}', '{compla_web}') '''
        cursor.execute(sql)
        conn.commit()
        conn.close()
        return redirect(url_for('complaints_admin'))
    else:
        return render_template("newcomplaints.html",
user_type=current_user.type)

@app.route('/newinventory', methods=['POST', 'GET'])
def newinventory():
    if request.method == 'POST':
        inve_web = request.form['inve']
        quan_web = request.form['quan']
        rent_web = request.form['rent']

        conn = psycopg2.connect(log[current_user.type])
        cursor = conn.cursor()
        sql = ("insert into Inventory (Inventory, Quantity,
Price_per_piece_lease) "
            f''values ('{inve_web}', '{quan_web}',
'{rent_web}') '''

        cursor.execute(sql)

```

```
        conn.commit()
        conn.close()
        return redirect(url_for('inventory_worker'))
    else:
        return render_template("newinventory.html",
user_type=current_user.type)

if __name__ == '__main__':
    app.run(debug=True)
```

ЛІСТИНГ Б.1, Лист 26