МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Колесникова Олександра Андрійовича
(прізвище, ім'я, по батькові здобувача)

на тему «Розробка ігрового симулятора управління космічними місіями»
(повна назва теми)

за матеріалами праць провідних спеціалістів з розробки ПЗ та проектування БД
(повна назва бази дослідження)

науковий керівник к.е.н., доц. _____ Ткаліченко С.В.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 11.06.2025 р. № 12
Завідувач кафедри _____
(підпис)
д.т.н., професор Зеленський О.С.
Наук. ступень, вчене звання *Ініціали, прізвище*

Кривий Ріг - 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____ Зеленський О.С.
(підпис) (Прізвище, ініціали)
« 11 » червня 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

1. Тема роботи «Розробка ігрового симулятора управління космічними місіями»

Керівник роботи к.е.н., доц. Ткаліченко С.В.
затверджені наказом закладу вищої освіти від «04» квітня 2025 р. № 222-ст

2. Строк подання здобувачем роботи до «09» червня 2025 р.

3. Зміст кваліфікаційної роботи, об’єкт, предмет та мета дослідження:

Розділ 1. Постановка задачі

Розділ 2. Розробка алгоритму розв’язання задачі

Розділ 3. Організація інформаційного забезпечення

Розділ 4. Розробка програмного забезпечення

Об’єкт дослідження: Процес розробки складних багатокомпонентних програмних систем.

Предмет дослідження: Інтеграція технологій C++, OpenGL та Laravel для створення ігрового Симулятора.

Мета кваліфікаційної роботи: *Розробка ігрового симулятора управління космічними місіями.*

5. Дата видачі завдання «04» квітня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів МДР	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	04.04.2025-13.04.2025	
2	Підготовка розділу 2	14.04.2025-26.04.2025	
3.	Підготовка розділу 3	27.04.2025-15.05.2025	
4	Підготовка розділу 4	16.05.2025-08.06.2025	
5	Реєстрація завершеної кваліфікаційної роботи	09.06.2025	Реєстраційний №___ «09»червня 2025 р.
6	Отримання відгуку від наукового керівника	10.06.2025	
7	Подання кваліфікаційної роботи на перегляд завідувачу кафедри	11.06.2025	
8	Отримання зовнішньої рецензії	12.06.2025	
9	Попередній захист кваліфікаційної роботи на кафедрі	13.06.2025	
10	Підготовка до захисту в ЕК	16.06.2025-21.06.2025	

Завдання підготував науковий керівник _____ __Ткаліченко С.В.___
(підпис) (прізвище та ініціали)

Завдання одержав _____ __Колесников О.А.__
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

на кваліфікаційну бакалаврську роботу

«Розробка ігрового симулятора управління космічними місіями»

Колесникова Олександра Андрійовича

Кваліфікаційна бакалаврська робота на здобуття освітньо-кваліфікаційного рівня бакалавра зі спеціальності 121 «Інженерія програмного забезпечення» - Державний університет економіки і технологій - Кривий Ріг, 2025.

У бакалаврській дипломній роботі розроблено програмний комплекс, що є ігровим симулятором управління космічними місіями. Проєкт реалізує ключові ігрові механіки: управління фінансами, дослідження технологій, вибір та виконання сюжетних місій, проєктування та створення космічних кораблів, а також симуляцію їх польоту.

Програмний комплекс складається з трьох взаємодіючих компонентів. Веб-застосунок для адміністрування розроблено на фреймворку Laravel з використанням Vue.js. Консольний та графічний клієнти для створення кораблів і візуалізації польотів реалізовано на мові C++ з використанням MFC та бібліотеки OpenGL. Усі частини системи взаємодіють через єдину базу даних PostgreSQL.

Ключові слова: ІГРОВИЙ СИМУЛЯТОР, КОСМІЧНІ МІСІЇ, C++, OPENGL, LARAVEL, VUE.JS, POSTGRESQL, БАГАТОКОМПОНЕНТНА СИСТЕМА, ІГРОВА МЕХАНІКА.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД(база даних)	**База даних.** Впорядкований набір логічно взаємопов’язаних даних, що використовуються для задоволення інформаційних потреб користувачів.
СУБД	Система управління базами даних. Програмний комплекс для створення та управління базами даних. У проєкті використовується PostgreSQL.
ПЗ	Програмне забезпечення.
C++	Об'єктно-орієнтована мова програмування, що використовувалася для створення консольного та графічного компонентів проєкту.
API	Application Programming Interface. Програмний інтерфейс застосунку. Набір правил, за якими різні програмні компоненти взаємодіють між собою.
MFC	Microsoft Foundation Classes. Бібліотека класів C++ від Microsoft для розробки застосунків з графічним інтерфейсом для ОС Windows.
OpenGL	Open Graphics Library. Програмний інтерфейс (API) для створення 2D та 3D графіки. Використано для візуалізації польоту.
Laravel	Веб-фреймворк, написаний на мові PHP, що використовує архітектурний патерн MVC (Model-View-Controller) для розробки веб-застосунку.
Vue.js	Прогресивний JavaScript-фреймворк для створення користувацьких інтерфейсів у веб-застосунку.

ЗМІСТ

ВСТУП

Сучасна індустрія розробки програмного забезпечення, зокрема в сфері створення комп'ютерних ігор, характеризується стрімким розвитком і постійним ускладненням технологічних процесів. Ця галузь, яка поєднує в собі передові досягнення комп'ютерних наук, інженерії програмного забезпечення та творчого дизайну, демонструє чітку тенденцію до створення складних, багатокомпонентних програмних продуктів, які здатні задовольнити зростаючі потреби сучасного користувача. Якщо в минулому розробка комп'ютерних ігор часто обмежувалася створенням монолітних застосунків, які функціонували як єдиний цілісний модуль, то сучасний ринок диктує необхідність розробки гнучких, модульних і високоефективних екосистем. Такі екосистеми включають у себе як клієнтські частини, що забезпечують взаємодію з користувачем, так і потужні серверні рішення, які відповідають за обробку великих обсягів даних, управління ігровою логікою та забезпечення стабільної роботи системи в цілому.

Особливо яскраво ці тенденції проявляються в жанрі ігрових симуляторів, які є одним із найбільш технологічно складних і вимогливих напрямів у сфері розробки ігор. Ігрові симулятори вирізняються високим рівнем деталізації, необхідністю створення реалістичних фізичних моделей, якісної графічної візуалізації, а також складних механік управління, економіки, прогресії гравця та взаємодії з ігровим світом. Усе це створює унікальні виклики для розробників, адже сучасні симулятори повинні не лише забезпечувати високий рівень занурення гравця, але й гарантувати стабільну та ефективну роботу всіх компонентів системи. Таким чином, однією з ключових інженерних проблем сучасності є необхідність ефективного поєднання різнорідних технологій для створення єдиного, цілісного ігрового світу, який би відповідав високим стандартам якості та функціональності.

Для вирішення таких завдань сучасні розробники використовують комбінацію різноманітних технологічних підходів. Наприклад, нативні мови програмування, такі як C++, широко застосовуються для реалізації

обчислювально складних завдань, таких як обробка фізичних моделей, рендеринг графіки чи оптимізація продуктивності. У поєднанні з графічними бібліотеками, такими як OpenGL, ці інструменти дозволяють створювати високоякісні візуальні ефекти та реалістичні симуляції. Водночас сучасні веб-технології, зокрема фреймворки на кшталт Laravel, відіграють ключову роль у створенні адміністративних панелей, систем управління даними та інших серверних компонентів, які забезпечують зручність адміністрування та масштабування системи. Таке поєднання технологій є типовим архітектурним рішенням для багатьох комерційних проєктів у сфері розробки ігор, що дозволяє досягти оптимального балансу між продуктивністю, гнучкістю та зручністю використання.

У цьому контексті розробка багатокомпонентного програмного комплексу, який об’єднує веб-застосунок, консольну програму та графічний клієнт через єдину базу даних, є не лише актуальним, але й надзвичайно важливим практичним завданням. Такий підхід дозволяє не тільки продемонструвати комплексний підхід до сучасної інженерії програмного забезпечення, але й показати, як різні технології можуть бути інтегровані для створення єдиного продукту, що відповідає сучасним вимогам ринку. Крім того, подібні проєкти мають значний потенціал для практичного застосування, оскільки вони можуть слугувати основою для створення комерційних продуктів або прототипів для подальшого розвитку.

Метою даної кваліфікаційної роботи є створення програмного комплексу ігрового симулятора управління космічними місіями, який би демонстрував ефективну інтеграцію різнорідних технологій, зокрема веб-технологій, нативних консольних додатків та засобів графічної візуалізації. Такий комплекс має на меті не лише забезпечити функціональну взаємодію всіх компонентів, але й продемонструвати їхню здатність працювати як єдина система, що забезпечує стабільність, масштабованість і зручність використання.

Для досягнення поставленої мети було визначено низку ключових завдань, які охоплюють усі етапи розробки програмного комплексу:

Проведення аналізу предметної області, що включає дослідження існуючих ігрових симуляторів, їхніх архітектурних особливостей, технологічних рішень і підходів до інтеграції різнорідних компонентів.

Проєктування архітектури багатокомпонентної системи, що передбачає чітке визначення ролей і способів взаємодії між веб-застосунком, консольною програмою та графічним клієнтом, а також створення схеми їхньої інтеграції через єдину базу даних.

Розробка структури та реалізація реляційної бази даних у системі управління базами даних (СУБД) PostgreSQL, яка забезпечуватиме ефективне зберігання, обробку та управління всіма ігровими сутностями, такими як космічні кораблі, місії, ресурси тощо.

Створення веб-застосунку на основі фреймворку Laravel, який слугуватиме для адміністрування ігрових місій, управління фінансами, дослідженнями технологій та іншими аспектами ігрового процесу.

Розробка консольного додатку на мові програмування C++, яке відповідатиме за реалізацію логіки створення космічних кораблів, їхньої конфігурації та взаємодії з іншими компонентами системи.

Створення графічного додатку на основі C++/MFC з використанням бібліотеки OpenGL для реалізації візуалізації польоту космічних кораблів, симуляції місій і створення імерсивного ігрового досвіду.

Проведення інтеграційного тестування, яке дозволить переконатися в коректній взаємодії всіх компонентів системи, їхній стабільності та відповідності вимогам проєкту.

Об’єктом дослідження у даній роботі є процес розробки складних багатокомпонентних програмних систем, зокрема тих, що застосовуються в ігрових симуляторах. Цей процес включає аналіз, проєктування, реалізацію та

тестування програмних комплексів, які об’єднують різнорідні технології для досягнення єдиної мети.

Предметом дослідження виступають методи, алгоритми та інструментальні засоби, які використовуються для інтеграції веб-технологій (зокрема фреймворку Laravel), нативних додатків, розроблених на мові C++, та графічних бібліотек, таких як OpenGL, у процесі створення ігрового симулятора управління космічними місіями.

У результаті виконання поставлених завдань буде створено програмний комплекс, який не лише демонструватиме життєздатність обраного архітектурного підходу, але й слугуватиме прикладом практичної реалізації сучасних інженерних рішень у сфері розробки програмного забезпечення. Такий комплекс може бути використаний як прототип для подальшого вдосконалення або як основа для створення комерційних продуктів у сфері ігрових симуляторів. Крім того, результати роботи матимуть практичну цінність для розробників, які прагнуть створювати складні багатокомпонентні системи, що відповідають сучасним вимогам ринку.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ

1.1. Аналіз предметної області та основні визначення

Сучасна індустрія розробки комп’ютерних ігор є однією з найдинамічніших і технологічно складних галузей у сфері програмної інженерії, яка стрімко розвивається завдяки інноваціям у комп’ютерних науках, проєктуванні програмного забезпечення та створенні ігрового контенту. Ця галузь поєднує передові технологічні досягнення, такі як обробка графіки в реальному часі, штучний інтелект, фізичне моделювання, із творчими аспектами дизайну, що включають розробку наративів, візуального стилю та звукового супроводу. Високий рівень конкуренції в індустрії змушує розробників постійно вдосконалювати свої продукти, шукати нові підходи до створення унікального ігрового досвіду та адаптуватися до зростаючих очікувань користувачів щодо якості, продуктивності та інноваційності.

Індустрія комп’ютерних ігор характеризується широким різноманіттям жанрів, кожен з яких має власні особливості, вимоги до архітектури програмного забезпечення, ігрових механік і технологічного стеку. Наприклад, жанри, такі як шутери від першої особи, покладаються на швидку обробку графіки та низьку затримку, тоді як стратегічні ігри вимагають складних алгоритмів управління ресурсами та штучного інтелекту. У цьому контексті ігрові симулятори виділяються як один із найскладніших жанрів, оскільки вони прагнуть максимально точно відтворювати реальні процеси, явища чи системи, що потребує інтеграції різнорідних технологій і глибокого розуміння предметної області.

Ігрові симулятори, зокрема симулятори управління космічними місіями, є унікальними через їхній акцент на реалізм і деталізацію. Вони поєднують елементи технічного моделювання (наприклад, симуляцію фізичних законів,

таких як гравітація чи аеродинаміка), економічного менеджменту (управління ресурсами, бюджетами) та інтерактивного дизайну, що забезпечує занурення гравця в ігровий світ. Такі симулятори вимагають створення складних систем, які включають реалістичні фізичні движки, модульні конструктори для створення об’єктів (наприклад, космічних апаратів), а також продумані механіки взаємодії між різними компонентами гри. Це, у свою чергу, накладає високі вимоги до архітектури програмного забезпечення, яка має бути гнучкою, масштабованою та здатною підтримувати інтеграцію різноманітних модулів, таких як графічні рендеринги, бази даних і алгоритми обробки даних у реальному часі.

Для аналізу предметної області було розглянуто ключові аспекти розробки ігрових симуляторів, включаючи їхні технічні, функціональні та користувацькі вимоги. Зокрема, досліджено сучасні тенденції в індустрії, такі як зростання популярності симуляторів із відкритим світом, використання хмарних технологій для обробки великих обсягів даних і впровадження штучного інтелекту для створення адаптивних ігрових механік. Також було проаналізовано приклади успішних проєктів, таких як Kerbal Space Program, які демонструють баланс між реалістичною симуляцією та доступністю для широкої аудиторії. Ці приклади дозволили визначити сильні сторони (високий рівень реалізму, модульність) та слабкі сторони (обмеження в архітектурі, складність масштабування) сучасних рішень, що стало основою для формування концепції власного проєкту.

Ключові визначення предметної області включають поняття ігрового симулятора як програмного продукту, що моделює реальні чи вигадані процеси з високим ступенем деталізації, ігрової механіки як набору правил і алгоритмів, що визначають взаємодію гравця з ігровим світом, а також технологічного стеку як сукупності інструментів і бібліотек, що використовуються для реалізації проєкту. У контексті цього проєкту особлива увага приділяється розробці симулятора космічних місій, який інтегрує реалістичну фізику, економічні

механіки та модульну архітектуру для забезпечення гнучкості й масштабованості. Таким чином, аналіз предметної області закладає теоретичний фундамент для подальшого проєктування та розробки програмного забезпечення, що відповідає сучасним стандартам індустрії та очікуванням користувачів.

1.2. Аналіз предметної області та основні визначення

Ігровий симулятор — це спеціалізований програмний продукт, який призначений для імітації управління певним процесом, об’єктом або системою з максимальною точністю та реалістичністю. На відміну від аркадних ігор, які зосереджені на спрощеному геймплеї, видовищності та швидкому залученні гравця, симулятори мають на меті відтворення реальних фізичних законів, технічних характеристик об’єктів і складної логіки управління. Такі продукти не лише надають розважальний досвід, але й можуть виконувати освітні функції, дозволяючи гравцям поглиблено вивчати принципи роботи складних систем, розвивати інженерні навички чи вдосконалювати управлінські здібності. Завдяки цьому ігрові симулятори є унікальним жанром, що поєднує в собі розвагу, навчання та вирішення складних завдань [11].

У межах жанру ігрових симуляторів можна виділити кілька основних підкатегорій, кожна з яких має свої особливості та цільову аудиторію:

Технічні симулятори — фокусуються на імітації управління різноманітними транспортними засобами або технічними системами. Прикладами є авіасимулятори (наприклад, Microsoft Flight Simulator), автосимулятори (Gran Turismo) або симулятори поїздів (Train Simulator). Такі ігри вимагають від гравця розуміння технічних аспектів і точного відтворення фізичних моделей.

Економічні симулятори — зосереджені на управлінні складними системами, такими як бізнес, місто чи навіть ціла цивілізація. Приклади

включають SimCity, Cities: Skylines або Civilization. Ці ігри акцентують увагу на стратегічному плануванні, управлінні ресурсами та економічних процесах.

Симулятори життя — дозволяють гравцям керувати повсякденним життям віртуальних персонажів, моделюючи соціальні взаємодії, кар'єрний ріст чи побутові задачі (наприклад, The Sims).

Космічні симулятори — є особливою категорією, що синтезує елементи технічних і економічних симуляторів. Вони вимагають від гравця як навичок управління складними технічними об'єктами (наприклад, конструювання та пілотування космічних кораблів), так і вміння ефективно керувати ресурсами, фінансами, науковими дослідженнями та довгостроковим плануванням. Цей піджанр є одним із найскладніших через необхідність поєднання різноманітних механік і технологій.

Космічні симулятори, як піджанр, є особливо актуальними в сучасному контексті, оскільки вони дозволяють гравцям не лише зануритися в унікальний ігровий світ, але й отримати знання про космічні технології, орбітальну механіку та принципи управління складними проєктами. Такі ігри мають значний потенціал як для розважальних, так і для освітніх цілей, що робить їх привабливими для широкої аудиторії.

Аналіз ринку та існуючих аналогів

Для успішної постановки задачі на розробку власного програмного продукту необхідно ретельно проаналізувати ринок ігрових симуляторів, зокрема космічних, а також вивчити сильні та слабкі сторони існуючих рішень. Такий аналіз дозволяє не лише визначити усталені стандарти жанру, але й виявити незайняті ніші, які можуть бути використані для створення унікального продукту з конкурентними перевагами.

Одним із найвідоміших і найуспішніших представників жанру космічних симуляторів є гра Kerbal Space Program (KSP), розроблена студією Squad. Цей проєкт здобув культовий статус серед гравців завдяки своєму інноваційному підходу до симуляції космічних польотів, поєднанню розважальних і освітніх

елементів, а також унікальній ігровій механіці. KSP є прикладом того, як складні технічні концепції можуть бути представлені в доступній і захоплюючій формі.

Ключові особливості Kerbal Space Program:

1. Реалістична фізична модель: Гра базується на принципах орбітальної механіки Ньютона, що дозволяє гравцям вивчати основи космічної навігації, такі як розрахунок орбіт, виконання гравітаційних маневрів і оптимізація траєкторій. Ця особливість робить KSP не лише розважальним продуктом, але й цінним інструментом для вивчення основ аерокосмічної інженерії.

2. Модульний конструктор: KSP надає гравцям широкий набір компонентів, таких як двигуни, паливні баки, командні модулі та наукові прилади, які можна комбінувати для створення космічних апаратів будь-якої конфігурації. Такий підхід стимулює інженерну творчість, дозволяючи гравцям експериментувати з різними конструкціями та підходами до вирішення завдань.

3. Режим кар’єри: Окрім режиму "пісочниці", де всі компоненти доступні одразу, KSP пропонує режим кар’єри, у якому гравець повинен виконувати контракти, заробляти кошти та наукові очки для розблокування нових технологій і компонентів. Цей режим додає стратегічний елемент, змушуючи гравця планувати ресурси та приймати виважені рішення.

Незважаючи на численні переваги, KSP має певні обмеження, які відкривають можливості для створення альтернативних продуктів із покращеною функціональністю та сучаснішою архітектурою. До основних недоліків належать:

1. Монолітна архітектура: KSP є прикладом класичного монолітного десктопного застосунку, у якому всі ігрові системи — фізичний рушій, графічний рендерер, логіка кар’єри, інтерфейс користувача — тісно пов’язані в єдиному виконуваному файлі. Такий підхід, хоч і був ефективним на момент створення гри, сьогодні вважається застарілим, оскільки ускладнює оновлення окремих компонентів, інтеграцію з зовнішніми сервісами та масштабування проєкту.

2. Відсутність багатокомпонентності: Гра не використовує сучасний підхід до розподілу логіки між клієнтською та серверною частинами. Усі дані та ігрова логіка зберігаються локально, що унеможливлює реалізацію повноцінного онлайн-режиму, створення централізованої платформи для обміну даними між гравцями чи інтеграцію з веб-сервісами.

3. Спрощений менеджмент: Хоча KSP включає елементи управління бюджетом і науковими дослідженнями, ці механіки реалізовані відносно просто. Гра не пропонує глибоких систем управління логістикою, персоналом чи інфраструктурою космодрому, що могло б додати додатковий рівень складності та залучення.

Постановка задачі для власного проєкту

На основі аналізу ринку, зокрема сильних і слабких сторін Kerbal Space Program, а також враховуючи сучасні тенденції у сфері розробки програмного забезпечення, було сформульовано концепцію власного проєкту. Замість створення ще одного монолітного ігрового симулятора пропонується розробити багатокомпонентний програмний комплекс, який складається з трьох взаємопов’язаних застосунків[10, с. 21]: веб-застосунку, консольного додатку та графічного клієнта. Такий підхід дозволяє усунути ключові обмеження KSP і забезпечити низку переваг:

Гнучкість і масштабованість: Кожен компонент системи може розроблятися, тестуватися та оновлюватися незалежно від інших, що значно спрощує підтримку та вдосконалення продукту.

Розподіл логіки: Складні обчислювальні задачі, такі як симуляція фізики польоту чи обробка параметрів космічних кораблів, виконуються нативними клієнтськими додатками, тоді як логіка управління даними, фінансами та дослідженнями винесена на серверну частину.

Кросплатформова взаємодія: Гравець може взаємодіяти з різними аспектами гри з різних пристроїв. Наприклад, керувати фінансами та

дослідженнями через веб-браузер на будь-якому пристрої, а виконувати симуляцію польотів на високопродуктивному ПК.

Таким чином, основною задачею кваліфікаційної роботи є проєктування та створення програмного комплексу ігрового симулятора управління космічними місіями, що складається з трьох ключових компонентів:

Веб-застосунок (адміністративна панель): Призначений для управління глобальними ігровими сутностями, такими як фінанси, місії, наукові дослідження та технології.

Консольний застосунок (конструктор кораблів): Високопродуктивний додаток на C++, який відповідає за логіку створення та конфігурації космічних апаратів.

Графічний застосунок (симулятор польоту): Додаток на C++/MFC з використанням OpenGL для візуалізації польотів і симуляції космічних місій.

Взаємодія між цими компонентами забезпечується через єдину централізовану базу даних, що гарантує цілісність, консистентність і безперебійну роботу ігрового світу.

1.3. Огляд технологій для розробки багатокомпонентної системи

Вибір технологічного стеку є критично важливим етапом[10, с. 135] у процесі розробки складних багатокомпонентних систем, таких як ігровий симулятор. Оскільки проєкт передбачає створення трьох окремих застосунків (веб-застосунку, консольного додатку та графічного клієнта), які взаємодіють через спільну базу даних, необхідно ретельно підібрати інструменти, що відповідають специфічним вимогам кожного компонента. Нижче наведено детальний огляд обраних технологій та обґрунтування їхнього вибору.

1. Серверна частина (веб-застосунок) — фреймворк Laravel

Для реалізації серверної частини, зокрема веб-панелі адміністрування, було обрано PHP-фреймворк Laravel, який є одним із найпопулярніших

інструментів для створення веб-застосунків. Вибір Laravel обґрунтований його численними перевагами, які ідеально відповідають потребам проєкту:

Архітектура MVC: Laravel[5, с. 15] базується на патерні Model-View-Controller, який забезпечує чітке розділення бізнес-логіки, представлення даних і обробки запитів користувача[2, с. 4]. Це сприяє створенню структурованого, зрозумілого та легкого для підтримки коду.

Розвинена екосистема: Laravel пропонує широкий набір готових рішень і пакетів, таких як Laravel Breeze для автентифікації чи Laravel Sanctum для API-токенів, що значно прискорює розробку та дозволяє зосередитися на реалізації ігрової логіки.

Eloquent ORM: Вбудована система об’єктно-реляційного відображення (ORM) дозволяє зручно працювати з базою даних, абстрагуючись від складних SQL-запитів, що забезпечує швидку та ефективну розробку.

2. Клієнтська частина (веб-інтерфейс) — Vue.js та Inertia.js

Для створення динамічного та інтерактивного користувацького інтерфейсу веб-панелі було обрано JavaScript-фреймворк Vue.js у поєднанні з бібліотекою Inertia.js[7, с. 22]:

Компонентний підхід Vue.js: Vue.js дозволяє створювати інтерфейс із модульних, перевикористовуваних компонентів, що ідеально підходить для побудови складних сторінок, таких як дашборди, форми управління місіями чи таблиці ресурсів.

Inertia.js для спрощення розробки: Inertia.js забезпечує інтеграцію між Laravel і Vue.js, дозволяючи створювати сучасні односторінкові застосунки (SPA) без необхідності розробки окремого REST API. Це значно скорочує час розробки та спрощує підтримку проєкту.

3. Нативні застосунки (консольний та графічний) — C++/MFC

Для реалізації компонентів, що вимагають високої продуктивності та низькорівневого доступу до апаратних ресурсів, було обрано мову програмування C++[4, с. 30]:

Висока продуктивність: Як компільована мова, C++ забезпечує максимальну швидкість виконання, що є критично важливим для таких завдань, як симуляція фізики, обробка складних математичних розрахунків і рендеринг графіки в реальному часі.

Контроль ресурсів: C++ надає розробнику повний контроль над управлінням пам’яттю та апаратними ресурсами, що дозволяє оптимізувати код для забезпечення максимальної ефективності.

Бібліотека MFC: Для створення графічного інтерфейсу симулятора польоту використано бібліотеку Microsoft Foundation Classes (MFC), яка є стандартним інструментом для розробки нативних Windows-застосунків на C++.

4. Графічна візуалізація — OpenGL

Для реалізації графічної складової симулятора польоту було обрано бібліотеку OpenGL (Open Graphics Library):

Низькорівневий контроль: OpenGL дозволяє розробнику повністю контролювати процес рендерингу, що дає змогу створювати власні графічні рішення, оптимізовані під конкретні завдання проєкту.

Широка підтримка: Як стандарт індустрії, OpenGL має велику кількість документації, навчальних матеріалів і підтримку спільноти, що полегшує розробку та вирішення потенційних проблем.

5. Система управління базами даних — PostgreSQL

Для забезпечення централізованого зберігання даних було обрано об’єктно-реляційну СУБД PostgreSQL[8, с. 29]:

Надійність і транзакційність: PostgreSQL підтримує ACID-транзакції, що гарантує цілісність даних навіть у разі збоїв[6, с. 81]. Це критично важливо для збереження ігрових сутностей, таких як результати місій чи фінансові дані.

Сумісність: PostgreSQL має потужні драйвери для інтеграції з Laravel (через Eloquent ORM) і C++ (через бібліотеку libpqxx[17]), що забезпечує безперебійну взаємодію всіх компонентів системи.

1.4. Формулювання вимог до продукту

Формулювання вимог до програмного продукту є одним із найважливіших етапів розробки, оскільки саме чітко визначені вимоги забезпечують правильне розуміння цілей проєкту та його успішну реалізацію. У контексті розробки багатокомпонентного ігрового симулятора управління космічними місіями необхідно сформувати повний набір вимог, які охоплюють як функціональні, так і нефункціональні аспекти системи.

Процес формування вимог

Формування вимог є ітеративним процесом, який включає кілька ключових етапів:

Бізнес-моделювання: Визначення основної концепції гри, її механік і циклів взаємодії гравця з системою (game loops).

Виявлення вимог: Збір функціональних і нефункціональних побажань до кожного компонента системи.

Аналіз і проєктування: Створення архітектури системи, проєктування бази даних і визначення взаємозв’язків між компонентами.

Реалізація та тестування: Перевірка відповідності розробленого продукту сформульованим вимогам.

Вимоги можна класифікувати за кількома критеріями:

За ступенем пріоритетності:

Обов’язкові (Must-have): Наприклад, можливість створення космічного корабля та запуску симуляції польоту.

Бажані (Should-have): Наприклад, детальна історія виконаних місій або розширений інтерфейс управління.

Необов’язкові (Nice-to-have) [11]: Наприклад, система досягнень чи кастомізація інтерфейсу.

За рівнем абстракції:

Бізнес-вимоги: Створення ігрового симулятора, що поєднує менеджмент і симуляцію.

Вимоги користувача: Наприклад, можливість досліджувати нові технології чи керувати фінансами.

Функціональні вимоги: Детальний опис поведінки системи для реалізації вимог користувача.

Класифікація та опис вимог до ігрового симулятора

Функціональні вимоги визначають конкретну поведінку системи та включають:

Веб-застосунок: Реалізація автентифікації, управління фінансами, місіями, дослідженнями технологій.

Консольний застосунок: Розрахунок параметрів і вартості космічних кораблів, збереження даних у базу.

Графічний застосунок: Візуалізація польоту, симуляція фізичних процесів, відображення результатів місій.

Нефункціональні вимоги описують якість і обмеження системи:

Продуктивність: Графічний компонент повинен забезпечувати стабільну частоту кадрів (не менше 30 FPS). Веб-сервер має обробляти запити за менш ніж 200 мс.

Надійність: Система повинна гарантувати цілісність даних навіть у разі збоїв. Усі критичні операції мають бути транзакційними.

Безпека: Дані користувачів повинні зберігатися в зашифрованому вигляді, а система — бути захищеною від SQL-ін’єкцій[8, с. 250] та інших атак.

Інтеграція: Усі компоненти системи повинні безперебійно взаємодіяти через базу даних PostgreSQL, використовуючи сумісні драйвери.

Висновки до розділу

У цьому розділі проведено детальний аналіз предметної області для розробки ігрового симулятора космічних місій і сформульовано задачу проєкту. Досліджено особливості жанру космічних симуляторів, який поєднує технічне моделювання (симуляція орбітальної механіки, аеродинаміки) та економічні аспекти (управління ресурсами, бюджетами). Цей жанр вимагає високого реалізму й інтеграції різнорідних технологій для створення захопливого ігрового досвіду.

Аналіз гри Kerbal Space Program виявив її сильні сторони: реалістичну фізичну модель і модульний конструктор для створення космічних апаратів. Водночас виявлено обмеження, зокрема монолітну архітектуру, що ускладнює масштабування, та слабкі механіки менеджменту, які обмежують стратегічну глибину. На основі цього запропоновано концепцію проєкту з багатокомпонентною архітектурою, яка забезпечує гнучкість і масштабованість [1, с. 198]. Цей підхід передбачає розподіл логіки між незалежними модулями, що спрощує розробку, підтримку та інтеграцію нових функцій.

Для реалізації проєкту обґрунтовано вибір технологічного стеку: Laravel і Vue.js для веб-частини, що забезпечує інтерактивний інтерфейс; C++/MFC для нативних застосунків, що гарантує продуктивність; OpenGL для рендерингу 3D-графіки; PostgreSQL як СУБД, що підтримує ACID-транзакції та складні запити. Сформульовано функціональні вимоги (конструювання апаратів, планування місій, симуляція) та нефункціональні (продуктивність, масштабованість), які створюють основу для подальшого проєктування та розробки, забезпечуючи відповідність сучасним стандартам індустрії.

РОЗДІЛ 2
РОЗРОБКА АЛГОРИТМУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

Розробка алгоритму є центральним етапом у процесі створення будь-якого програмного продукту, оскільки саме алгоритм визначає послідовність дій, які повинна виконувати система для досягнення поставленої мети. Для складних, багатокомпонентних систем, якою є розроблюваний ігровий симулятор, цей етап набуває особливої ваги. Він вимагає не лише визначення логіки окремих операцій, але й проєктування високорівневих сценаріїв взаємодії між незалежними програмними модулями.

2.1. Методологія розробки алгоритму

Розробка алгоритму для ігрового симулятора управління космічними місіями ґрунтується на структурному підході, який передбачає декомпозицію складної задачі на менші, керовані підзадачі. Цей метод, широко застосовуваний у програмній інженерії, забезпечує систематизацію розробки, спрощує тестування та інтеграцію. Головна задача — створення ігрового процесу, що поєднує реалістичну симуляцію, стратегічне планування та інженерне конструювання, — розбита на три логічні блоки: стратегічне планування, інженерне конструювання та симуляція польоту.

Стратегічне планування відповідає за визначення цілей місії, розподіл ресурсів (бюджет, паливо) і прогнозування результатів. Цей блок включає алгоритми для розрахунку траєкторій і оцінки доцільності місії. Інженерне конструювання охоплює створення космічного апарата через модульний конструктор, де гравець обирає компоненти (двигуни, баки, системи навігації). Алгоритми цього блоку перевіряють сумісність компонентів і розраховують технічні характеристики апарата. Симуляція польоту відтворює поведінку

апарата в космосі, враховуючи закони небесної механіки, гравітацію та інші фізичні параметри, що забезпечує реалізм гри.

Декомпозиція спростила розробку та лягла в основу архітектурного рішення. Кожен блок реалізовано як окремий програмний компонент, що підвищує модульність і гнучкість. Наприклад, планування може бути частиною веб-застосунку на Laravel, а симуляція — нативним модулем на C++ з OpenGL для рендерингу 3D-графіки. Це оптимізує ресурси, полегшує тестування та інтеграцію.

Для документування алгоритму використано блок-схеми, які наочно ілюструють послідовність операцій і потоки даних між компонентами. Наприклад, блок-схема показує, як дані про апарат із модуля конструювання передаються до симуляції для обробки фізичним движком. Це спрощує розуміння логіки та верифікацію системи.

Структурний підхід забезпечує масштабованість і повторне використання коду. Компоненти розроблено з можливістю розширення, наприклад, додавання нових апаратів чи механік. Оптимізація, як-от кешування даних і паралельні обчислення, забезпечує продуктивність симуляції в реальному часі. Подальші розділи розкриють деталі реалізації, інтеграцію з PostgreSQL і взаємодію з інтерфейсом.

2.2. Алгоритм основного ігрового циклу

Основний ігровий цикл є ядром ігрового процесу та визначає послідовність дій, які гравець виконує протягом ігрової сесії. Цей алгоритм був спроєктований таким чином, щоб забезпечити логічний та інтуїтивно зрозумілий перебіг гри. Блок-схема основного алгоритму, що ілюструє взаємодію компонентів, представлена на рис. 2.1.

Рис. 2.1. Архітектурна схема програмного комплексу

Алгоритм складається з трьох ключових етапів, кожен з яких реалізує певну частину ігрової механіки.

Крок 1: Етап стратегічного планування.

Цей етап реалізовано у веб-застосунку. Алгоритм цього кроку наступний:

Початок: Користувач автентифікується в системі.

Завантаження даних: Система завантажує з бази даних поточний стан гри користувача (game_state).

Аналіз та прийняття рішень: Користувачеві надається інтерфейс для виконання стратегічних дій:

Аналіз списку доступних місій.

Управління фінансами (переказ коштів між бюджетами).

Інвестування в наукові дослідження в "Лабораторії".

Вибір однієї з місій як активної для подальшого виконання.

Збереження стану: Будь-яка дія користувача, що змінює стан гри (наприклад, переказ коштів або вибір місії), ініціює запит до сервера, який оновлює відповідні записи в базі даних.

Кінець етапу: Етап вважається завершеним, коли гравець обрав активну місію та готовий перейти до конструювання корабля.

Крок 2: Етап інженерного конструювання.

Цей етап реалізовано у консольному застосунку. Алгоритм цього кроку включає:

Початок: Користувач запускає застосунок та ідентифікує себе.

Отримання контексту: Програма підключається до бази даних та зчитує актуальний стан гри, включаючи ID обраної місії, бонуси від технологій та стан інженерного бюджету.

Розрахунок та проєктування: На основі отриманих даних система розраховує та відображає розвідані вимоги місії. Гравець вводить бажані параметри корабля, а система проводить детальний розрахунок його вартості з урахуванням усіх бонусів.

Перевірка та збереження: Програма перевіряє, чи достатньо коштів на інженерному бюджеті. Якщо так, після підтвердження користувача, виконується транзакція: новий корабель зберігається в базу даних, а його вартість списується з бюджету.

Кінець етапу: Після успішного створення корабля етап завершується.

Крок 3: Етап симуляції польоту.

Цей етап реалізовано у графічному застосунку.

Початок: Гравець запускає графічний клієнт.

Завантаження даних: Застосунок завантажує з БД дані про обрану місію та щойно створений корабель.

Запуск симуляції: Після підтвердження гравцем, починається візуальна симуляція польоту, реалізована за допомогою OpenGL.

Визначення результату: По завершенні анімації, алгоритм порівнює параметри корабля з істинними вимогами місії та визначає успіх або провал.

Оновлення стану: Результат місії записується в базу даних: оновлюється статус у журналі запусків, нараховується нагорода, видаляється використаний корабель.

Кінець циклу: Ігровий цикл завершено. Гравець може повернутися до етапу планування для вибору нової місії.

Висновки до розділу

У даному розділі було ґрунтовно розроблено та детально описано алгоритмічну основу функціонування ігрового симулятора управління космічними місіями. Запропоновано високорівневий алгоритм основного ігрового циклу, який логічно структуровано поділено на три ключові етапи: планування місії, конструювання космічного апаратного забезпечення та симуляція польоту. Для наочної візуалізації послідовності операцій і потоків даних між компонентами системи використано детальну блок-схему, яка чітко ілюструє взаємодію модулів.

Розроблений алгоритм відіграє центральну роль у проєктуванні, оскільки він не лише визначає логіку ігрового процесу, але й формує основу для архітектурного рішення. Кожен етап алгоритму реалізується окремим спеціалізованим програмним модулем, що забезпечує високу модульність, гнучкість і полегшує подальше масштабування системи. Такий підхід сприяє чіткій структуризації коду, спрощує тестування та інтеграцію компонентів, а також створює надійний фундамент для розробки інформаційного та програмного забезпечення. Детальний розгляд реалізації цих модулів, включаючи їх програмну та інформаційну взаємодію, буде представлено в наступних розділах.

РОЗДІЛ 3
ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Проєктування бази даних (БД) є одним із найважливіших етапів створення будь-якої інформаційної системи, оскільки саме від якості структури даних залежить цілісність, консистентність та швидкість доступу до інформації. Для багатокомпонентного програмного комплексу, де декілька незалежних застосунків одночасно працюють з єдиним сховищем, цей етап набуває особливої ваги.

3.1. Вибір системи управління базами даних (СУБД)

На етапі вибору системи управління базами даних (СУБД) для розробки ігрового симулятора управління космічними місіями було проведено детальний порівняльний аналіз кількох популярних СУБД, включаючи MySQL, SQLite, PostgreSQL, MongoDB та Oracle Database. Основна мета аналізу полягала в ідентифікації оптимального рішення, яке б відповідало функціональним і нефункціональним вимогам проєкту, забезпечувало високу продуктивність, надійність і сумісність із вибраним технологічним стеком.

Критерії вибору включали такі аспекти: підтримка складних запитів, масштабованість, забезпечення цілісності даних, продуктивність при великих обсягах даних, гнучкість у роботі з різними типами даних, а також легкість інтеграції з іншими компонентами системи, такими як веб-застосунок на базі Laravel і нативні модулі на C++/MFC. Додатково враховувалися ліцензійні умови, активність спільноти розробників, наявність документації та підтримка сучасних стандартів.

MySQL була розглянута як популярна реляційна СУБД із простим налаштуванням і широкою підтримкою в екосистемі PHP/Laravel. Проте її обмеження в підтримці складних транзакцій і менша гнучкість у роботі з

нестандартними типами даних порівняно з PostgreSQL зробили її менш привабливою для цього проєкту.

SQLite оцінювалася як легковагова СУБД, ідеальна для невеликих проєктів або вбудованих систем. Однак її обмеження в паралельній обробці запитів і масштабованості роблять її непридатною для проєкту, який передбачає інтенсивну взаємодію з базою даних і обробку складних ігрових даних у реальному часі.

MongoDB, як представник NoSQL-систем, розглядалася для зберігання неструктурованих або слабко структурованих даних. Проте, враховуючи реляційну природу даних у симуляторі (наприклад, зв’язки між місіями, апаратними компонентами та параметрами симуляції), MongoDB виявилася менш ефективною для забезпечення цілісності та складних запитів.

Oracle Database була виключена через високу вартість ліцензії та надмірну складність для потреб проєкту, що не виправдовує її використання в порівнянні з відкритими альтернативами.

За результатами аналізу перевага була віддана об'єктно-реляційній СУБД PostgreSQL, яка вирізняється низкою ключових переваг. По-перше, PostgreSQL забезпечує повну підтримку ACID-транзакцій, що гарантує цілісність і надійність даних навіть при складних операціях, таких як одночасне оновлення параметрів місії кількома користувачами. По-друге, вона пропонує розширюваність типів даних, дозволяючи створювати власні типи та функції, що є важливим для моделювання унікальних ігрових об’єктів, наприклад, параметрів космічних апаратів чи траєкторій. По-третє, PostgreSQL демонструє високу продуктивність при обробці великих обсягів даних і складних запитів, що критично для симуляції в реальному часі. Крім того, ця СУБД має відмінну сумісність із технологічним стеком проєкту: вона легко інтегрується з Laravel через ORM Eloquent, а також підтримує ефективну роботу з нативними модулями через відповідні драйвери.

Додатковими перевагами PostgreSQL є активна спільнота розробників, яка забезпечує регулярні оновлення та підтримку, а також наявність розвинених інструментів для адміністрування, резервного копіювання та моніторингу. Відкрита ліцензія PostgreSQL (PostgreSQL License) дозволяє використовувати її без додаткових витрат, що є важливим фактором для економічної ефективності проєкту.

Таким чином, вибір PostgreSQL як централізованої СУБД для проєкту обґрунтований її надійністю, гнучкістю, продуктивністю та сумісністю з іншими компонентами системи. Ця СУБД створює міцний фундамент для реалізації складної логіки зберігання й обробки даних, що буде детально розглянуто в наступних розділах, присвячених проєктуванню бази даних і розробці програмного забезпечення.

3.2. Розробка логічної та фізичної моделі даних

Процес проєктування бази даних (БД) для ігрового симулятора управління космічними місіями був ретельно структурований і поділений на два ключові етапи: створення логічної моделі даних та її подальша реалізація у вигляді фізичної моделі. Ці етапи забезпечують створення надійної, ефективної та масштабованої структури для зберігання й обробки даних, що відповідає функціональним і нефункціональним вимогам проєкту.

Логічна модель даних була розроблена з метою формалізації основних сутностей предметної області, їхніх атрибутів і взаємозв’язків між ними. На цьому етапі проведено детальний аналіз предметної області, що дозволив ідентифікувати ключові об’єкти, такі як космічні апарати, місії, компоненти обладнання, траєкторії польоту та параметри симуляції. Для кожної сутності визначено набір атрибутів, які відображають їхні характеристики, наприклад, технічні параметри апаратного забезпечення, часові характеристики місій чи фізичні властивості траєкторій. Особлива увага приділялася принципам

нормалізації, зокрема забезпеченню структури бази даних у третій нормальній формі (3НФ) [6, с. 112]. Це дозволило усунути надмірність даних, мінімізувати аномалії при вставці, оновленні чи видаленні записів, а також забезпечити логічну цілісність і ефективність запитів. Взаємозв'язки між сутностями (наприклад, «один до багатьох» між місією та її компонентами або «багато до багатьох» між апаратними модулями та їхніми характеристиками) були формалізовані за допомогою ER-діаграм (Entity-Relationship Diagram), які наочно ілюструють структуру даних і логіку їхньої взаємодії.

Фізична модель даних стала наступним етапом, на якому логічна модель була адаптована до технічних особливостей обраної СУБД PostgreSQL. На цьому етапі визначено конкретні типи даних для кожного атрибута (наприклад, INTEGER для ідентифікаторів, VARCHAR для текстових полів, NUMERIC для фізичних параметрів, TIMESTAMP для часових міток), а також встановлено первинні та зовнішні ключі для забезпечення реляційної цілісності. Наприклад, первинні ключі використовувалися для унікальної ідентифікації записів у таблицях, таких як «Місії» чи «Космічні апарати», тоді як зовнішні ключі забезпечували зв'язок між таблицями, наприклад, між таблицею компонентів і таблицею апаратного забезпечення. Додатково було враховано оптимізацію продуктивності: створено індекси для часто використовуваних полів, таких як ідентифікатори місій чи параметри траєкторій, що прискорює виконання запитів у реальному часі. Також розглянуто використання специфічних можливостей PostgreSQL, таких як підтримка JSONB для зберігання гнучких структур даних (наприклад, конфігурацій апаратного забезпечення) та створення тригерів для автоматизації певних операцій, таких як оновлення статусу місії.

Для забезпечення зручності подальшої розробки та підтримки фізична модель була задокументована у вигляді схем бази даних, включаючи специфікацію таблиць, їхніх атрибутів, типів даних, обмежень і зв'язків. Такий підхід не лише полегшує інтеграцію бази даних із програмними компонентами (наприклад, веб-застосунком на Laravel чи нативними модулями на C++/MFC),

але й створює міцний фундамент для масштабування системи в майбутньому. У наступних розділах буде детально розглянуто реалізацію запитів до бази даних, їхню оптимізацію та інтеграцію з іншими компонентами програмного забезпечення.

3.3. Структура бази даних та опис ключових таблиць

Фінальна структура бази даних (ER-діаграма) представлена на рис. 3.1.

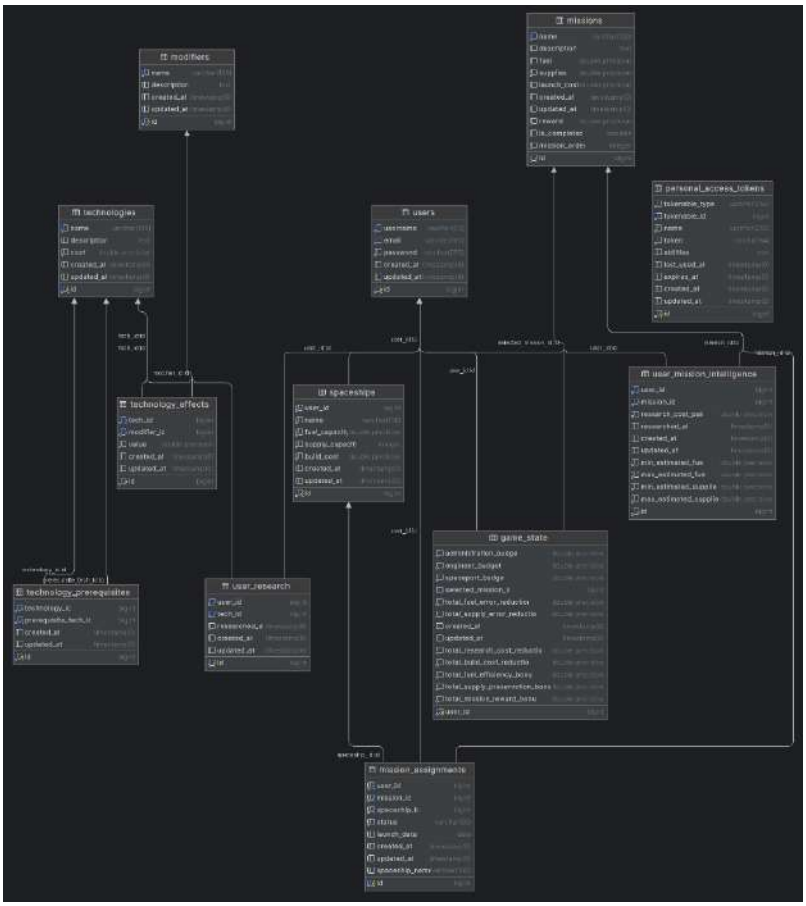

Рис. 3.1. ER-діаграма бази даних ігрового симулятора

Нижче наведено опис ключових таблиць та їх призначення.

Таблиця users (табл. 3.1) є центральною сутністю в системі, що відповідає за ідентифікацію та автентифікацію гравців. Поле id є первинним ключем (Primary Key) та забезпечує унікальність кожного запису. Поля name та email зберігають реєстраційні дані та мають обмеження UNIQUE, щоб уникнути дублювання акаунтів. Поле password зберігає не сам пароль, а його хеш,

отриманий за допомогою стійкого алгоритму (наприклад, bcrypt), що є фундаментальною вимогою безпеки для захисту облікових даних користувачів.

Таблиця 3.1

Users

Основна таблиця для зберігання даних про гравців.

Назва поля	Тип даних	Опис	Обмеження
id	BIGINT	Унікальний ідентифікатор гравця	PRIMARY KEY
Name	VARCHAR(255)	Ім'я користувача (логін)	NOT NULL, UNIQUE
Email	VARCHAR(255)	Електронна пошта	NOT NULL, UNIQUE
password	VARCHAR(255)	Хешований пароль користувача	NOT NULL

id: Унікальний ідентифікатор користувача (первинний ключ).

name, email, password: Є основною таблицею для ідентифікації гравця в системі.

Таблиця game_state (табл. 3.2) призначена для зберігання поточного стану ігрового прогресу кожного користувача. Вона має зв'язок "один-до-одного" з таблицею users через унікальний зовнішній ключ user_id. Таке рішення дозволяє відокремити дані автентифікації від ігрових даних. Таблиця містить поля для трьох типів ігрових бюджетів, що використовуються в різних ігрових механіках. Поле selected_mission_id є зовнішнім ключем до таблиці missions і може бути NULL, якщо гравець на даний момент не обрав активну місію. Група полів з префіксом total_ є прикладом керованої денормалізації: вони зберігають сумарні, вже розраховані бонуси від усіх досліджених технологій, що дозволяє уникнути складних обчислень при кожному запиті та значно підвищує продуктивність системи.

Таблиця 3.2

Game_state

Зберігає поточний стан гри для кожного користувача.

Назва поля	Тип даних	Опис	Обмеження
id	BIGINT	Унікальний ідентифікатор запису	PRIMARY KEY
user_id	BIGINT	Зовнішній ключ до таблиці users	FOREIGN KEY, UNIQUE
administration_budget	DOUBLE PRECISION	Бюджет для досліджень та вивчення місій	NOT NULL, DEFAULT 0
engineer_budget	DOUBLE PRECISION	Бюджет для будівництва кораблів	NOT NULL, DEFAULT 0
spaceport_budget	DOUBLE PRECISION	Бюджет для запуску місій	NOT NULL, DEFAULT 0
selected_mission_id	BIGINT	ID активної місії	FOREIGN KEY (nullable)
total_mission_bonus	DOUBLE PRECISION	Агреговані бонуси від технологій	NOT NULL, DEFAULT 0

user_id: Зовнішній ключ, що реалізує зв'язок "один-до-одного" з таблицею users.

administration_budget, engineer_budget, spaceport_budget: Поля для зберігання трьох типів ігрових бюджетів.

selected_mission_id: ID місії, яку гравець обрав як активну.

Поля з префіксом total_: Агреговані бонуси від досліджених технологій.

Таблиця missions (табл. 3.3) функціонує як довідник, що містить статичну інформацію про всі доступні в грі місії. Вона включає текстові описи (name, description), а також ключові числові параметри, що визначають складність та економіку місії: fuel та supplies (базові вимоги для успішного виконання),

launch_cost (вартість запуску, що списується з бюджету космодрому) та reward (нагорода). Поле is_completed є глобальним прапорцем, що може використовуватися для глобальних ігрових подій, тоді як mission_order забезпечує послідовність проходження сюжетної кампанії.

Таблиця 3.3

Missions

Довідник усіх доступних в грі місій.

Назва поля	Тип даних	Опис	Обмеження
id	BIGINT	Ідентифікатор місії	PRIMARY KEY
name	VARCHAR(255)	Назва місії	NOT NULL
description	TEXT	Опис місії	
fuel	DOUBLE PRECISION	Істинні вимоги до палива	NOT NULL
supplies	DOUBLE PRECISION	Істинні вимоги до припасів	NOT NULL
launch_cost	DOUBLE PRECISION	Вартість запуску місії	NOT NULL
reward	DOUBLE PRECISION	Нагорода за успішне виконання	NOT NULL
is_completed	BOOLEAN	Прапорець, чи була місія пройдена	DEFAULT false
mission_order	INTEGER	Порядковий номер у сюжетній лінії	NOT NULL

name, description: Назва та опис місії.

fuel, supplies: Істинні вимоги місії до палива та припасів.

launch_cost, reward: Вартість запуску та нагорода за успішне виконання.

Таблиця spaceships (табл. 3.4) призначена для зберігання інформації про всі космічні кораблі, створені гравцями. Кожен запис унікально ідентифікується

полем id і обов'язково пов'язаний з власником через зовнішній ключ user_id. Поля fuel_capacity та supply_capacity фіксують ключові технічні характеристики корабля на момент його створення, а поле build_cost зберігає його фінальну вартість з урахуванням усіх знижок.

Таблиця 3.4

Spaceships

Зберігає інформацію про космічні кораблі, створені користувачем.

Назва поля	Тип даних	Опис	Обмеження
id	BIGINT	Унікальний ідентифікатор корабля	PRIMARY KEY
user_id	BIGINT	Власник корабля (до users)	FOREIGN KEY
name	VARCHAR(255)	Назва корабля, введена гравцем	NOT NULL
fuel_capacity	DOUBLE PRECISION	Місткість паливних баків	NOT NULL
supply_capacity	DOUBLE PRECISION	Місткість відсіків для припасів	NOT NULL
build_cost	DOUBLE PRECISION	Фактична вартість будівництва	NOT NULL

user_id: Зовнішній ключ, що вказує на власника корабля.

name, fuel_capacity, supply_capacity: Основні параметри корабля.

Таблиця mission_assignments (табл. 3.5) є асоціативною сутністю, що виконує роль журналу запусків та реалізує зв'язок "багато-до-багатьох" між користувачами та місіями. Вона фіксує кожну спробу проходження місії. Збереження spaceship_name є важливим для історії, оскільки сам корабель після місії видаляється. Поле status дозволяє відстежувати результат кожної конкретної спроби.

Таблиця 3.5

Mission_assignments:

Журнал запусків місій. Реалізує зв'язок "багато-до-багатьох" між користувачами та місіями.

Назва поля	Тип даних	Опис	Обмеження
id	BIGINT	Унікальний ідентифікатор запису	PRIMARY KEY
user_id	BIGINT	Гравець, що запустив місію	FOREIGN KEY
mission_id	BIGINT	Запущена місія	FOREIGN KEY
spaceship_id	BIGINT	Корабель, що використовувався	FOREIGN KEY
spaceship_name	VARCHAR(255)	Збережене ім'я корабля на момент запуску	NOT NULL
status	VARCHAR(255)	Статус запуску (Launched, Completed, Failed)	NOT NULL
launch_date	TIMESTAMP	Дата та час запуску	NOT NULL

user_id, mission_id: Зовнішні ключі.

spaceship_name, status, launch_date: Деталі конкретного запуску.

technologies, modifiers, technology_effects: Група таблиць, що реалізує гнучку систему технологій та їхніх ефектів. technologies — довідник технологій, modifiers — довідник можливих бонусів, technology_effects — зв'язуюча таблиця, що визначає, який ефект і якої сили надає конкретна технологія[2, с. 315].

Така структура забезпечує високий рівень нормалізації, гнучкість для подальшого розширення (наприклад, додавання нових типів бонусів чи місій) та ефективну взаємодію між усіма компонентами програмного комплексу.

Висновки до розділу

У даному розділі описано організацію інформаційного забезпечення проєкту ігрового симулятора космічних місій. Обґрунтовано вибір СУБД PostgreSQL через її надійність, підтримку ACID-транзакцій, гнучкість типів даних і сумісність із технологічним стеком (Laravel, C++/MFC). Розроблено логічну модель даних, яка відображає сутності (місії, апарати, компоненти), їхні атрибути та зв’язки, з дотриманням принципів нормалізації (3НФ). ER-діаграма наочно ілюструє структуру. Фізична модель, адаптована для PostgreSQL, включає типи даних, первинні та зовнішні ключі, індекси для оптимізації запитів. Детально описано таблиці, їхні поля та взаємозв’язки, що забезпечує міцну основу для розробки та інтеграції програмних компонентів системи.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Цей розділ присвячено детальному опису процесу практичної реалізації трьох ключових компонентів програмного комплексу. Кожен компонент розроблявся з використанням технологій, найбільш відповідних для його завдань, що дозволило створити збалансовану та ефективну систему.

4.1. Розробка веб-застосунку для адміністрування

Веб-застосунок є центральним елементом програмного комплексу, що виконує роль адміністративної панелі для стратегічного планування ігрового процесу. В основі його архітектури лежить патерн MVC (Model-View-Controller) [2, с. 4]. Для реалізації системи автентифікації було використано стартовий пакет Laravel Breeze.

Ключові ігрові інтерфейси, такі як Панель управління (Dashboard) (рис. 4.1), Модуль управління фінансами (рис. 4.2), Модуль вибору місій (рис. 4.3), Лабораторія (рис. 4.4) та Історія місій (рис. 4.5), були реалізовані як інтерактивні Vue.js компоненти. Вся серверна логіка інкапсульована у відповідних контролерах. Наприклад, логіка переказу коштів, що включає валідацію та транзакції з блокуванням, знаходиться у FinanceController. Детальний лістинг коду цього контролера представлено у Додатку А. Для забезпечення актуальності даних в реальному часі на ключових сторінках було імплементовано механізм polling.

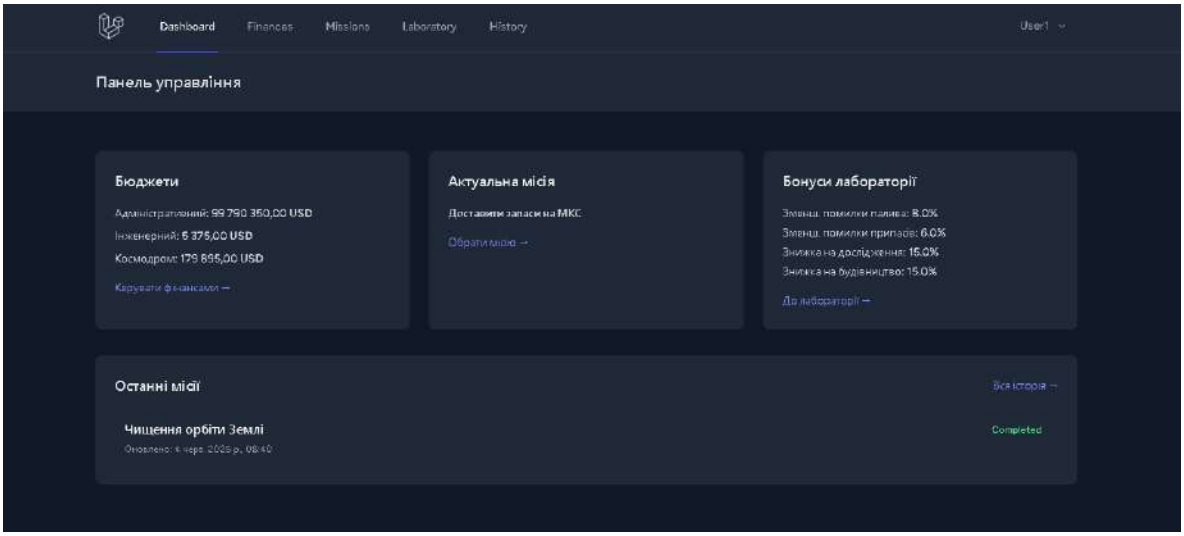

Рис. 4.1. Загальний вигляд панелі управління

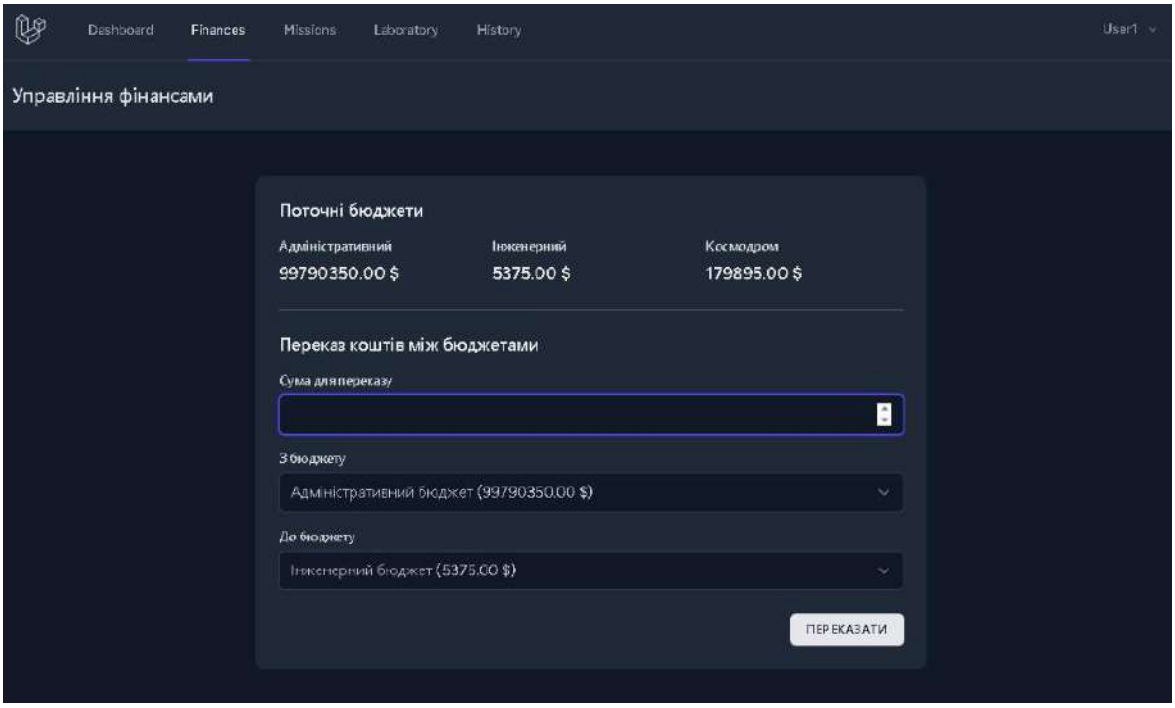

Рис. 4.2. Модуль управління фінансами

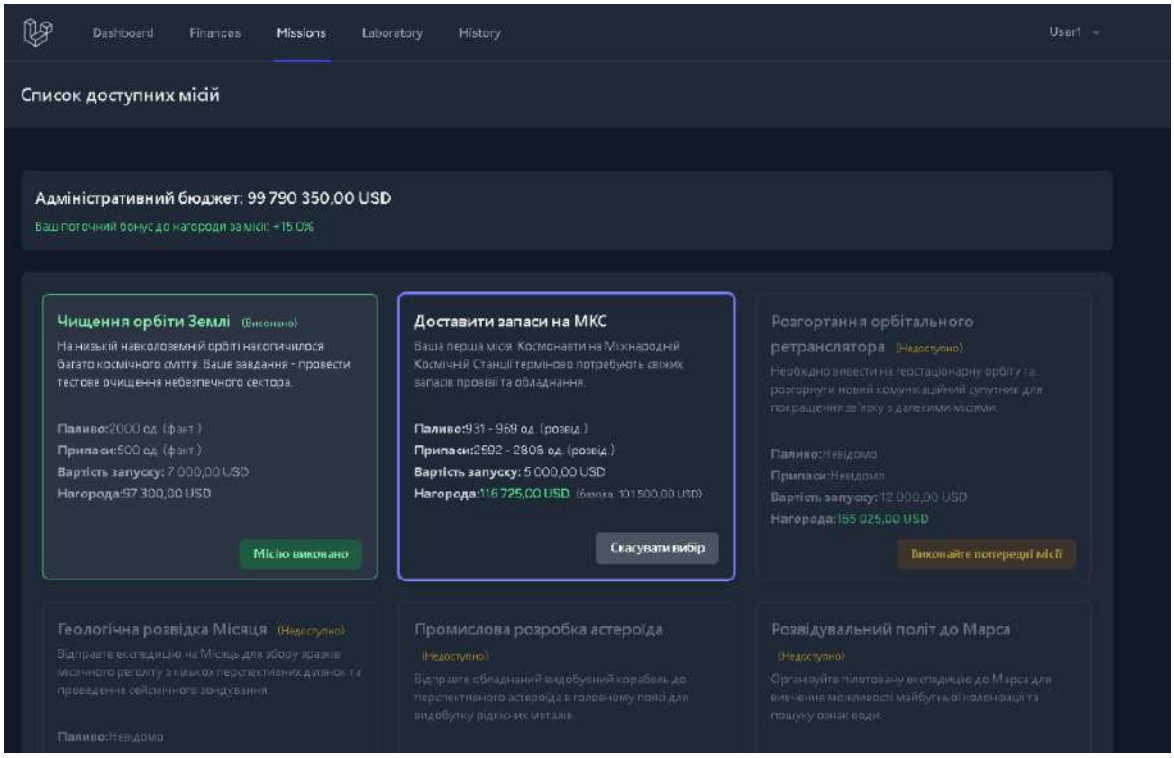

Рис. 4.3. Модуль вибору місій

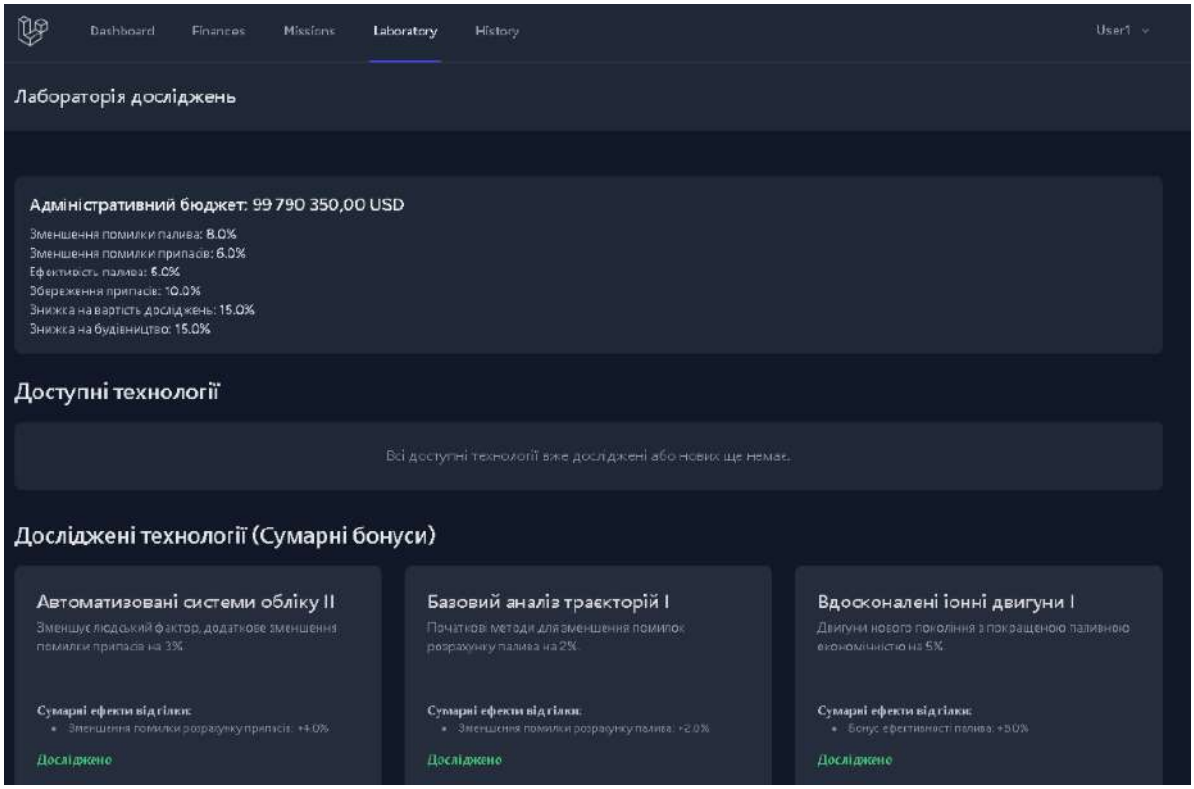

Рис. 4.4. Модуль лабораторії

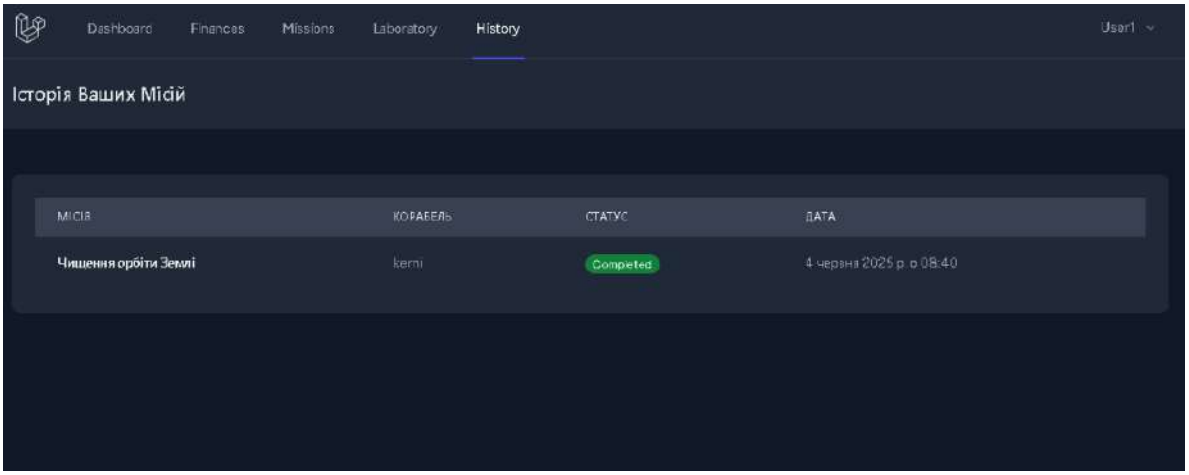

Рис. 4.5. Історія місій

4.2. Розробка консольного застосунку для створення кораблів

Консольний застосунок "Інженерний термінал" реалізований на мові C++ і виконує обчислювально складні задачі з проєктування космічних кораблів. Взаємодія з базою даних PostgreSQL реалізована за допомогою бібліотеки libpqxx [17].

Інтерфейс застосунку реалізовано у вигляді класичного текстового меню (рис. 4.6), що імітує роботу з інженерним терміналом. Основна функціональність — процес створення нового корабля (рис. 4.7), що включає розрахунок вартості на основі базових констант та динамічних бонусів, отриманих з БД. Для забезпечення цілісності даних операція створення корабля виконується в рамках транзакції. Для захисту від SQL-ін'єкцій використовуються параметризовані запити. Повний лістинг коду застосунку наведено у Додатку Б.

```
=====================================
Головне меню консолі створення кораблів
-------------------------------------

Поточний стан гри (Користувач ID: 1):
  Інженерний бюджет: 5375.00 USD
  Обрана місія ID: 2
  Знижка на будівництво: 15.00%
-------------------------------------

Опції:
  1. Створити новий корабель для обраної місії (ID: 2)
  o. Оновити дані стану гри
  q. Вийти з програми
Ваш вибір:
```

Рис. 4.6. Головне меню консольного застосунку

```
--- Створення нового корабля для місії ID: 2 ---
Поточний інженерний бюджет для цієї операції: 5375.00 USD

Дані розвідки для обраної місії (ID 2):
  Приблизне паливо: 931.00 - 969.00 од.
  Приблизні припаси: 2592.00 - 2808.00 од.
-----------------------------------
  Введіть назву корабля: spaceship
  Введіть бажану місткість палива:931.00 - 969.00): 950
  Введіть бажану місткість припасів:2592.00 - 2808.00): 2700

--- Параметри нового корабля ---
Назва: spaceship
Місткість палива: 950.00 од.
Місткість припасів: 2700.00 од.
-----------------------------------

--- Деталізація вартості будівництва ---
  Вартість корпусу:                          10000.00 USD
  Вартість паливних баків (на 950 од.):      19000.00 USD
  Вартість відсіків для припасів (на 2700 од.):    27000.00 USD
  ------------------------------------------------
  Базова вартість (сума):                    56000.00 USD
  Знижка (15%):                              -8400.00 USD
  ------------------------------------------------
  Підсумкова вартість будівництва:           47600.00 USD
-------------------------------------------------
Недостатньо коштів в інженерному бюджеті для будівництва цього корабля.
  Потрібно: 47600.00 USD
  Доступно: 5375.00 USD

Натисніть Enter для повернення до головного меню...
```

Рис. 4.7. Процес створення корабля та розрахунок вартості

4.3. Розробка графічного застосунку для симуляції польоту

Графічний застосунок "Центр управління польотами" розроблено на C++ з використанням бібліотек MFC та OpenGL [5, с. 55]. Проєкт створено як діалогове вікно (рис. 4.8), яке умовно поділено на інформаційну панель та область візуалізації.

Процес налаштування графічного конвеєра інкапсульовано в методі SetupOpenGL. За малювання сцени відповідає метод RenderScene, що викликається по таймеру. В ньому реалізована базова 2D/псевдо-3D анімація польоту корабля (рис. 4.9). Логіка запуску місії та обробки її результатів інкапсульована в методах OnBnClickedButtonLaunch та UpdateMissionStatus відповідно. Всі критичні зміни в базі даних також виконуються в рамках транзакцій. Лістинги ключових методів представлено у Додатку В та Г.

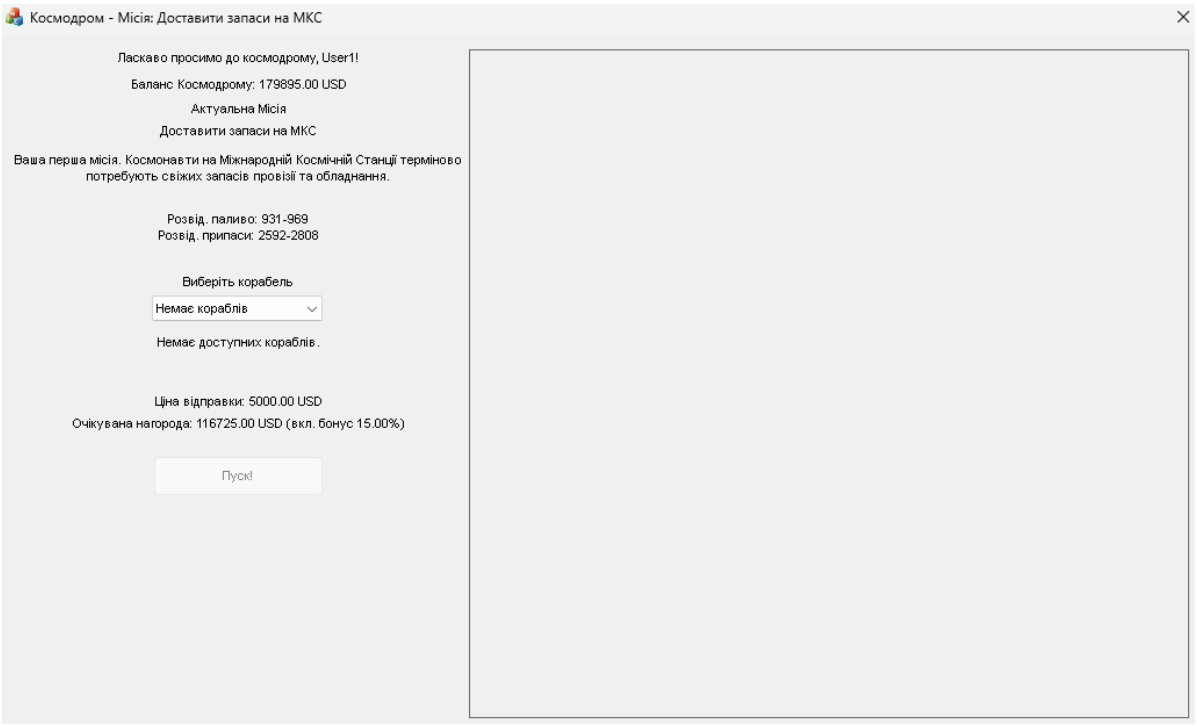

Рис. 4.8. Головне вікно графічного застосунку до запуску симуляції

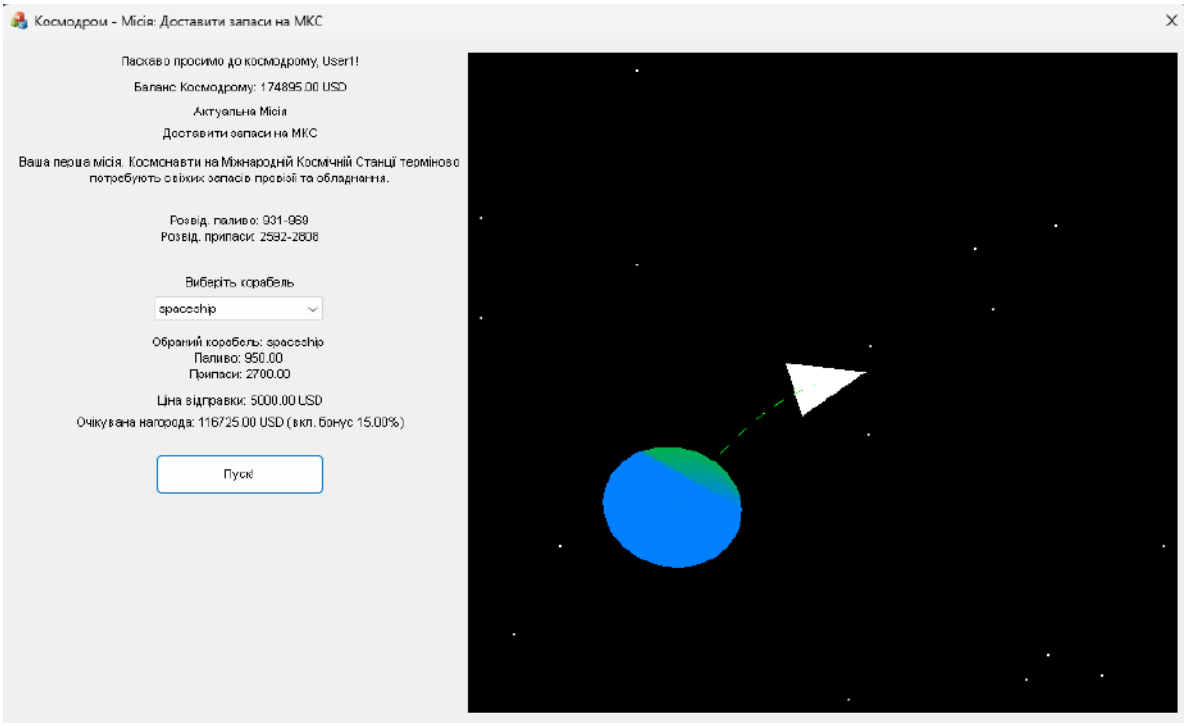

Рис. 4.9. Процес симуляції польоту корабля

Висновки до розділу

У даному розділі було детально розглянуто процес практичної реалізації трьох програмних компонентів, що складають ігровий симулятор. Було описано архітектурні та технологічні рішення, застосовані при розробці кожного з компонентів.

По-перше, було розроблено веб-застосунок на базі Laravel та Vue.js, що реалізує ключові ігрові механіки управління. По-друге, було створено консольний застосунок на C++, що виконує обчислювально складні задачі. По-третє, розроблено графічний застосунок на C++/MFC та OpenGL для візуалізації польоту.

Продемонстровано, як кожен компонент взаємодіє з єдиною базою даних, використовуючи транзакції для забезпечення цілісності даних. Таким чином, у третьому розділі було показано повний цикл розробки всіх складових програмного комплексу, кожен з яких є функціонально завершеним та готовим до тестування.

РОЗДІЛ 5.
ТЕСТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ

Тестування програмного забезпечення є невід'ємним етапом життєвого циклу розробки, метою якого є перевірка відповідності продукту заявленим вимогам, виявлення та виправлення помилок. Враховуючи багатокомпонентну архітектуру розробленого ігрового симулятора, процес тестування був комплексним і включав у себе кілька рівнів.

5.1. Методологія та види тестування

Для забезпечення якості програмного комплексу застосовувалися наступні види тестування:

1. Модульне тестування (Unit Testing):Цей вид тестування проводився на рівні окремих функцій та класів у кожному з трьох компонентів.

У веб-застосунку (Laravel): Перевірялася коректність роботи окремих методів контролерів (наприклад, валідація даних форми переказу коштів), а також логіка, інкапсульована в сервісних класах та моделях Eloquent.

У консольному та графічному застосунках (C++): Тестувалися окремі функції, відповідальні за розрахунки (наприклад, функція розрахунку вартості корабля), функції взаємодії з БД та функції конвертації даних.

2. Інтеграційне тестування: Цей етап був ключовим, оскільки перевіряв коректність взаємодії між трьома незалежними компонентами через спільну базу даних.

Сценарій: Створювався тестовий користувач. Через веб-інтерфейс досліджувалася технологія, що дає знижку на будівництво. Потім запускався консольний застосунок, який мав коректно зчитати новий бонус і застосувати його при розрахунку вартості корабля. Нарешті, у графічному застосунку

перевірялося, чи з'явився новостворений корабель у списку доступних для запуску.

Перевірка транзакцій: Окремо тестувалася надійність транзакційних операцій. Наприклад, імітувалася ситуація, коли під час створення корабля коштів на бюджеті достатньо, але з'єднання з БД розривається. Тест вважався успішним, якщо після відновлення з'єднання кошти на бюджеті гравця залишалися незмінними, а новий корабель у базі не з'являвся.

3. Тестування користувацького інтерфейсу (UI Testing): Проводилася ручна перевірка коректності роботи всіх елементів інтерфейсу у веб-застосунку та графічному клієнті. Перевірялася робота кнопок, форм, випадаючих списків, а також коректність відображення даних.

4. Системне тестування (End-to-End Testing)[11, с. 201]: На фінальному етапі проводився повний наскрізний тест, що імітував повний ігровий цикл з точки зору користувача:

Реєстрація нового гравця через веб-інтерфейс.

Вивчення місії та дослідження технології у веб-додатку.

Запуск консольного застосунку та створення корабля.

Запуск графічного застосунку, вибір корабля та запуск симуляції.

Перевірка успішного завершення місії та нарахування нагороди.

Перегляд оновленої інформації в історії місій у веб-додатку.

5.2. Результати тестування

У ході тестування було виявлено та виправлено низку помилок, переважно пов'язаних з некоректним оновленням даних в інтерфейсах та логічними неточностями при розрахунках бонусів. Проведене комплексне тестування показало, що всі компоненти системи коректно взаємодіють між собою, дані в базі залишаються цілісними, а основний функціонал відповідає поставленим вимогам. Програмний комплекс визнано готовим до демонстрації.

Висновки до розділу

У даному розділі було описано процес комплексного тестування розробленого ігрового симулятора. Була представлена методологія тестування, що включала в себе чотири основні рівні: модульне, інтеграційне, тестування користувацького інтерфейсу та системне тестування.

Було детально описано цілі та сценарії кожного виду тестування, що дозволило всебічно перевірити як окремі компоненти, так і систему в цілому. Особливу увагу було приділено інтеграційному тестуванню для перевірки коректності взаємодії між веб-додатком, консольним та графічним клієнтами через спільну базу даних.

За результатами тестування було виявлено та усунено низку помилок, що дозволило підвищити стабільність та надійність програмного комплексу. Фінальне системне тестування підтвердило, що розроблений продукт повністю відповідає поставленим функціональним та нефункціональним вимогам. Таким чином, етап тестування успішно завершено, і система готова до впровадження та подальшого розвитку.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи досягнуто мети — розроблено програмний комплекс, ігровий симулятор управління космічними місіями, що демонструє ефективну інтеграцію різнорідних технологій. Усі завдання виконано:

1. Проведено аналіз предметної області та аналогів, сформульовано унікальну концепцію проєкту.
2. Спроєктовано багатокомпонентну архітектуру з трьох застосунків (веб, консольний, графічний), що взаємодіють через базу даних.
3. Розроблено нормалізовану базу даних у PostgreSQL, що забезпечує цілісність даних.
4. Створено веб-застосунок на Laravel і Vue.js для управління фінансами, місіями та дослідженнями.
5. Реалізовано консольний додаток на C++ для обчислень, пов’язаних із проєктуванням космічних кораблів.
6. Розроблено графічний додаток на C++/MFC та OpenGL для візуалізації польотів і симуляції місій.
7. Проведено тестування, яке підтвердило коректність роботи всіх компонентів.

Практичне значення полягає у створенні прототипу, який демонструє життєздатність архітектурного рішення. Комплекс може слугувати навчальним посібником для вивчення принципів побудови складних систем і адаптуватися для комерційних чи освітніх проєктів. У результаті отримано оптимізований, протестований програмний продукт, що відповідає всім вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення. - Фабула, 2018. - 368 с.

2. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Прийоми об'єктно-орієнтованого проєктування. Патерни проєктування. - СПб.: Пітер, 2020. - 368 с.

3. Фаулер М. Архітектура корпоративних програмних додатків. - М.: Вільямс, 2016. - 544 с.

4. Страуструп Б. Мова програмування C++. 4-е вид. - М.: Вільямс, 2016. - 1328 с.

5. Шрайнер Д., Ейнджел Е. Інтерактивна комп'ютерна графіка. Вступний курс на базі OpenGL. 6-е вид. - М.: Вільямс, 2013. - 896 с.

6. Отуелл Т. Laravel: Up & Running. A Framework for Building Modern PHP Apps. 3rd ed. - O'Reilly Media, 2021. - 450 p.

7. Фейн Я., Мойз Р. Vue.js у дії. - Manning Publications, 2018. - 336 p.

8. Харрисон Г. PostgreSQL. Введення та оптимізація. - СПб.: Пітер, 2018. - 368 с.

9. Рэдмонд А., Уилсон Дж. Сім баз даних за сім тижнів. Вступ до сучасних баз даних та ідеології NoSQL. - ДМК Прес, 2018. - 408 с.

10. Макконнелл С. Досконалий код. Майстер-клас. - М.: Російська Редакція, 2010. - 896 с.

11. Nystrom R. Game Programming Patterns. [Електронний ресурс]. - Режим доступу: https://gameprogrammingpatterns.com/

12. Офіційна документація C++ (Standard C++ Foundation). [Електронний ресурс]. - Режим доступу: https://isocpp.org/

13. Офіційна документація Laravel. [Електронний ресурс]. - Режим доступу: https://laravel.com/docs

14. Офіційна документація OpenGL. [Електронний ресурс]. - Режим доступу: https://www.opengl.org/

15. Офіційна документація PostgreSQL. [Електронний ресурс]. - Режим доступу: https://www.postgresql.org/docs/

ДОДАТКИ

Додаток А

Лістинг коду контролера FinanceController.php

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Log;
class FinanceController extends Controller
{
    /**
     * Обробляє запит на переказ коштів між бюджетами користувача.
     */
    public function processTransfer(Request $request)
    {
        $request->validate([
            'amount' => 'required|numeric|min:0.01',
            'from_budget' => 'required|string|in:administration,engineer,spaceport',
            'to_budget' =>
'required|string|in:administration,engineer,spaceport|different:from_budget',
        ]);
        $user = Auth::user();
        $amount = (float) $request->input('amount');
        $fromField = $request->input('from_budget') . '_budget';
        $toField = $request->input('to_budget') . '_budget';
        // Використання транзакції для забезпечення цілісності даних
        DB::beginTransaction();
```

```
        try {
            // Блокування рядка для уникнення стану гонитви (race conditions)
            $gameState = $user->gameState()->lockForUpdate()->firstOrFail();
            if ($gameState->{$fromField} < $amount) {
                DB::rollBack();
                return redirect()->route('finances.transfer.form')
                    ->with('error', 'Insufficient funds in the source budget.')
                    ->withInput();
            }
            // Виконання операції
            $gameState->{$fromField} -= $amount;
            $gameState->{$toField} += $amount;
            $gameState->save();
            DB::commit();
            return redirect()->route('finances.transfer.form')->with('success', 'Funds
transferred successfully!');
        } catch (\Exception $e) {
            DB::rollBack();
            Log::error('Finance transfer error for user ' . $user->id . ': ' . $e-
>getMessage());
            return redirect()->route('finances.transfer.form')
                ->with('error', 'An error occurred during the transfer. Please try again.');
        }
    }
}
```

Додаток Б

Лістинг коду консольного застосунку для створення кораблів

```
#include <iostream>
#include <string>
#include <pqxx/pqxx>
// ... інші необхідні заголовки

using namespace std;
using namespace pqxx;

// Структури для зберігання даних
struct GameState { /* ... поля ... */ };
struct MissionIntelligenceData { /* ... поля ... */ };
struct SpaceshipParams { /* ... поля ... */ };

int main() {
    // ... (код підключення до БД та ідентифікації користувача)

    // Головний цикл програми
    do {
        // ... (вивід головного меню)



       // Обробка вибору користувача для створення корабля
        if (tolower(user_choice_main_menu) == '1') {
```

Продовження Додатку Б

```
        // ... (отримання даних місії та введення параметрів корабля)

        // --- Ключова логіка розрахунку вартості ---
        const double BASE_SHIP_HULL_COST = 10000.0;
        const double COST_PER_FUEL_CAPACITY_UNIT = 20.0;
        const double COST_PER_SUPPLY_CAPACITY_UNIT = 10.0;

        double base_build_cost = BASE_SHIP_HULL_COST +
            (newShip.fuel_capacity * COST_PER_FUEL_CAPACITY_UNIT) +
            (newShip.supply_capacity * COST_PER_SUPPLY_CAPACITY_UNIT);

        double actual_build_cost = base_build_cost;
        double discount_amount = 0.0;



        // Застосування знижки від досліджених технологій
        if (currentUserGameState.total_build_cost_reduction > 0) {
            double discountPercentage =
currentUserGameState.total_build_cost_reduction / 100.0;
            discount_amount = base_build_cost * discountPercentage;
            actual_build_cost = base_build_cost - discount_amount;
        }

        // ... (вивід деталізації вартості)
        // --- Перевірка бюджету та виконання транзакції ---
```

```
        if (currentUserGameState.engineer_budget >= actual_build_cost) {
            cout << "Розпочати будівництво корабля? (y/n): ";
            char confirm_build_action;
            cin >> confirm_build_action;

            if (tolower(confirm_build_action) == 'y') {
                work W_build(C); // Початок транзакції
                try {
                    // 1. Створення нового корабля
                    pqxx::params insert_params;
                    // ... (додавання параметрів)
                    W_build.exec(pqxx::zview(
                        "INSERT INTO spaceships (...) VALUES (...)"),
                        insert_params
                    );

                    // 2. Списання коштів з бюджету
                    double new_engineer_budget =
currentUserGameState.engineer_budget - actual_build_cost;
                    pqxx::params update_params;
                    // ... (додавання параметрів)
                    W_build.exec(pqxx::zview(
                        "UPDATE game_state SET engineer_budget = $1 WHERE user_id
= $2"),
                        update_params
                    );
```

Продовження Додатку Б

```
                W_build.commit(); // Фіксація транзакції
                cout << "Будівництво завершено успішно!" << endl;
            }
            catch (const exception& e_build) {
                // Транзакція автоматично відкотиться при виході з блоку try-
catch
                cerr << "Помилка будівництва. Зміни відкочено." << endl;
            }
        }
    } else {
        cout << "Недостатньо коштів в інженерному бюджеті." << endl;
    }
  }
} while (tolower(user_choice_main_menu) != 'q');

return 0;
}
```

Додаток В

Лістинг коду графічного застосунку

Фрагмент SpaceMissionSimDlg.h (ключові оголошення)

```
#ifndef SPACEMISSIONSIMDLG_H
#define SPACEMISSIONSIMDLG_H
#include "pch.h"
class CSpaceMissionSimDlg : public CDialogEx
{
public:
   // ... (конструктор та деструктор)
protected:
   // ... (оголошення обробників повідомлень)
   afx_msg void OnBnClickedButtonLaunch();
   afx_msg void OnTimer(UINT_PTR nIDEvent);
   DECLARE_MESSAGE_MAP()
private:
   // Ключові змінні-члени
   bool m_isMissionActive; // Прапорець активності симуляції
   HGLRC m_hRC;           // Контекст рендерингу OpenGL
   HDC m_hDC;             // Контекст пристрою вікна
   FlightPath m_flightPath; // Структура для анімації польоту
   // Ключові методи
   bool SetupOpenGL();
   void RenderScene();
   void CleanupOpenGL();
   void UpdateMissionStatus();
};
```

Додаток Г

Лістинг коду графічного застосунку

Фрагмент SpaceMissionSimDlg.cpp (ключові методи)

```
bool CSpaceMissionSimDlg::SetupOpenGL()
{
    m_pOpenGLWnd = GetDlgItem(IDC_OPENGL_AREA);
    m_hDC = ::GetDC(m_pOpenGLWnd->GetSafeHwnd());
    // ... (код вибору та встановлення формату пікселів)
    m_hRC = wglCreateContext(m_hDC);
    wglMakeCurrent(m_hDC, m_hRC);
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Чорний фон
    glEnable(GL_DEPTH_TEST);
    return true;
}
// Метод рендерингу сцени
void CSpaceMissionSimDlg::RenderScene()
{
    if (!m_isMissionActive) return; // Малюємо тільки якщо місія активна
    wglMakeCurrent(m_hDC, m_hRC);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    // Оновлення позиції корабля для анімації
    m_flightPath.time += 0.016f;
    float t = m_flightPath.time / (2 * 3.14159f);
    m_flightPath.x = 5.0f * t;
    m_flightPath.y = sin(m_flightPath.time) * 1.0f;
```

```
    // Камера слідує за кораблем
    gluLookAt(m_flightPath.x, m_flightPath.y + 2.0f, 5.0f,
              m_flightPath.x, m_flightPath.y, 0.0f,
              0.0f, 1.0f, 0.0f);

    // ... (код для малювання зірок, планет)

    // Малювання корабля (трикутник)
    glPushMatrix();
    glTranslatef(m_flightPath.x, m_flightPath.y, 0.0f);
    // ... (розрахунок кута повороту)
    glRotatef(angle, 0.0f, 0.0f, 1.0f);
    glBegin(GL_TRIANGLES);
    // ... (вершини трикутника)
    glEnd();
    glPopMatrix();

    SwapBuffers(m_hDC);

    // Перевірка завершення місії
    if (t >= 1.0f) {
        m_isMissionActive = false;
        UpdateMissionStatus(); // Виклик методу для обробки результатів
    }
}
```

Продовження Додатку Г

```
// Обробник натискання кнопки "Пуск!"
void CSpaceMissionSimDlg::OnBnClickedButtonLaunch()
{
    // ... (перевірки наявності обраного корабля та бюджету)

    // Підтвердження від користувача
    if (AfxMessageBox(confirmMsg, MB_YESNO) == IDYES) {
        // Виконання транзакції для запуску місії
        pqxx::work W(*m_pDbConnection);
        try {
            // INSERT в mission_assignments
            W.exec(pqxx::zview("INSERT INTO mission_assignments ..."));
            // UPDATE для списання коштів з game_state
            W.exec(pqxx::zview("UPDATE game_state SET spaceport_budget = ..."));
            W.commit();

            // Активація симуляції
            m_isMissionActive = true;
        } catch (const std::exception& e) {
            // ... (обробка помилки)
        }
    }
}
```

ЗГОДА здобувача вищої освіти

Державного університету економіки і технологій про перевірку кваліфікаційної роботи на прояви академічного плагіату
та розміщення в Репозитарії Університету

Я, Колесников Олександр Андрійович, підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота «Розробка ігрового симулятора управління космічними місіями» виконана самостійно та не містить академічного плагіату. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований, що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлений(на).

Дата підпис ініціали, прізвище (власноруч)

ВІДГУК

на кваліфікаційну роботу здобувача освітнього ступеня «бакалавр»
Державного університету економіки і технологій

Колесников Олександр Андрійович

(Прізвище, ім'я, по батькові)

Тема роботи*:* «Розробка ігрового симулятора управління космічними місіями»

Обсяг роботи: __67__ стор., _5_ табл., _11_ рис., __4__ додатків, __23__ бібліогр.

Ступінь розкриття теорії дослідження*:* У роботі продемонстровано глибоке розуміння основ розробки складних програмних систем. Студент провів аналіз предметної області, дослідив архітектурні підходи та обґрунтував вибір технологічного стеку. Теоретичні положення викладено логічно та послідовно.

Оцінка самостійності та творчості при розкритті теми*:* Робота виконана студентом повністю самостійно. Колєсніков О.А. продемонстрував творчий підхід при проєктуванні багатокомпонентної архітектури та реалізації взаємодії між різнорідними технологіями. Студент самостійно розробив структуру бази даних, реалізував ігрову логіку та провів комплексне тестування системи.

Наукова та практична цінність*:* Наукова цінність роботи полягає в системному підході до інтеграції веб-технологій (Laravel), нативної розробки (C++) та комп'ютерної графіки (OpenGL) для вирішення комплексної задачі. Практична цінність полягає у створенні готового програмного прототипу, що демонструє життєздатність обраної архітектури. Розроблений комплекс може бути використаний як навчальний посібник або основа для подальшої розробки комерційних чи освітніх ігрових проєктів.

Недоліки роботи*:* Мають місце незначні технічні недоліки в оформленні роботи.

Загальні висновки*:* Кваліфікаційна робота Колесникова О.А. є завершеним науково-практичним дослідженням, що повністю відповідає вимогам, які висуваються до бакалаврських робіт. Студент продемонстрував високий рівень теоретичної підготовки та практичних навичок. Робота заслуговує на високу позитивну оцінку, а її автор - на присвоєння освітнього ступеня «бакалавр» за спеціальністю «Інженерія програмного забезпечення».

Науковий керівник _____ д.т.н., професор О.С. Зеленський

(підпис) (посада, ініціали, прізвище)

__10 червня 2025 р.__

(дата)

РЕЦЕНЗІЯ

на кваліфікаційну роботу здобувача освітнього ступеня «бакалавр»
Державного університету економіки і технологій

Колесникова Олександра Андрійовича
(прізвище, ім'я, по батькові здобувача)

На тему: «Розробка ігрового симулятора управління космічними місіями»
(повна назва теми)

1. Актуальність і практичне значення роботи*:* Тема роботи є надзвичайно актуальною, оскільки відповідає сучасним тенденціям в індустрії розробки ПЗ, де все більшої популярності набувають складні багатокомпонентні системи. Практичне значення полягає у демонстрації успішної інтеграції різнорідних платформ (веб, консоль, графіка) для створення єдиного програмного продукту, що є цінним досвідом для інженерії ПЗ.

2. Якість проведеного аналізу проблеми*:* Проведено глибокий аналіз предметної області. Автор детально дослідив архітектурні особливості існуючих ігрових симуляторів, виявив їхні переваги та недоліки, на основі чого сформулював власну унікальну концепцію та обґрунтовані вимоги до програмного комплексу.

3. Недоліки або дискусійні питання у роботі*:* Робота виконана на високому рівні. Як дискусійне питання можна відзначити, що в майбутньому архітектуру можна було б розвинути в бік використання REST API для взаємодії компонентів замість прямого доступу до БД, що підвищило б її гнучкість. Однак, обраний підхід є цілком виправданим для цілей даного проєкту.

4. Наукова і практична цінність*:* Наукова цінність роботи полягає у систематизації підходів до проєктування та розробки складних систем, що поєднують різні технології. Практична цінність підтверджується створенням повноцінного програмного прототипу, який може бути використаний як у навчальних, так і в комерційних цілях.

5. Висновки*:* Кваліфікаційна робота Колесникова О.А. є завершеною та самостійною науково-дослідною працею. Робота розкриває актуальну тему, виконана на високому технічному рівні та відповідає всім вимогам. Рекомендую роботу до захисту з високою позитивною оцінкою.

Рецензент:

_____
(посада і місце роботи рецензента)

_____
(П.І.Б.)

_____ 12 червня 2025 р
(підпис) (дата)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ОЦІНЮВАЛЬНИЙ ЛИСТ НА КВАЛІФІКАЦІЙНУ РОБОТУ[1]
здобувача освітнього ступеня «бакалавр» спец. Інженерія програмного забезпечення

Колесникова Олександра Андрійовича
(Прізвище, ім'я, по батькові)

Критерії	Нормативне значення, балів	Оцінка кваліфікаційної роботи за критеріями
1. Оцінювання змістовних аспектів кваліфікаційної роботи		
1.1. *Загальний рівень розкриття теми* (відповідність виконаної роботи затвердженому завданню, повнота та рівень розв'язання завдань дослідження, досягнення поставленої мети)	**0-10**	
1.2. *Теоретична цінність отриманих результатів* (глибина, всебічність і повнота викладення теоретичного матеріалу, відображення дискусійних питань, загальний рівень опрацювання джерел, рівень узагальнення існуючих теоретичних підходів, методів і методик, обґрунтованість висновків щодо узагальнення теоретичних положень теми)	**0- 15**	
1.3. *Практична цінність отриманих результатів* (використання сучасних та оригінальних методів дослідження, наявність елементів наукової новизни, рівень вірогідності і надійності аналітичного обґрунтування, глибина аналітично-діагностичного вивчення стану прояву проблеми, відсутність помилок у розрахунках, актуальність зібраних і проаналізованих даних, обґрунтованість висновків щодо фактичного стану та перспектив розвитку проблеми, що досліджується)	**0-20**	
1.4. *Наявність логічної послідовності й наукового стилю викладу матеріалу дослідження* (володіння студентом літературною мовою і професійною термінологією, вміння логічно, аргументовано викладати результати досліджень і розробок, вдало використовувати графічний матеріал)	**0-10**	
2. Оцінювання організаційних аспектів кваліфікаційної роботи		
2.1. Дотримання графіка виконання кваліфікаційної роботи	**0-5**	
2.2. Відповідність роботи вимогам щодо її оформлення	**0-5**	
2.3. Наявність підтвердження апробації результатів дослідження	**0-5**	
Усього за критеріями змістовності і оформлення	Max **70 балів**	
3. Захист кваліфікаційної роботи		
3.1. *Презентація роботи* (якість викладення змісту кваліфікаційної роботи в доповіді, стиль викладення, мовна /грамотність, якість графічного матеріалу)	**0-10**	
3.2. *Відповіді на поставлені запитання, вміння вести наукову дискусію* (вміння стисло, змістовно, переконливо, аргументовано відповідати на поставлені запитання, а також на зауваження керівника та рецензента)	**0-20**	
Усього за критеріями захисту кваліфікаційної роботи	**0-30** балів	
Загальна кількість балів	Max **100 балів**	

Голова комісії _____ _____
(підпис) (ініціали, прізвище, посада, н.с., вч. зв.)
Заст. голови комісії _____ _____
(підпис) (ініціали, прізвище, посада, н.с., вч. зв.)
Члени комісії _____ _____
(підпис) (ініціали, прізвище, посада, н.с., вч. зв.)

[1] Назва роботи відповідно до ОПП