**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**

**КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ**

# Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Комп'ютерна система керування та стабілізації руху колісних роботів**

Виконав: студент 2 курсу, групи 8.1219-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
Д.А. Старшеков
(ініціали та прізвище )
Керівник професор, д. ф.-м. н. В. Г. Вербицький
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «АйтіДіменшн»
В.С. Тряпічко
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2020

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**

Кафедра програмного забезпечення автоматизованих систем
Рівень вищої освіти другий (магістерський)
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Інженерія програмного забезпечення
(код та назва)

**ЗАТВЕРДЖУЮ**
Завідувач кафедри _____ В.Г. Вербицький
"01" вересня 2020 року

**З А В Д А Н Н Я**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ**

Старшекову Данилу Анатолійовичу
(прізвище, ім'я, по батькові)

1.Тема роботи Комп'ютерна система керування та стабілізації руху колісних роботів

керівник роботи професор, доктор фіз-мат. наук В. Г. Вербицький
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від "25" травня 2020 року № 600-с

2. Строк подання студентом кваліфікаційної роботи 30.11.2020
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження засобів впливу на рух неголономної колісної системи;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

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

6. Консультанти розділів магістерської роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата Завдання прийняв |
|---|---|---|
| | | |

7. Дата видачі завдання 10.04.2020

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів магістерської роботи | Строк виконання етапів магістерської роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної області | 11.12.2019 – 1.03.2020 | виконано |
| 2 | Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником | 10.04.2020 | виконано |
| 3 | Аналіз існуючих методів рішення | 11.04.2020 –25.05.2020 | виконано |
| 4 | Проєктування серверу застосунків | 26.05.2020 – 1.06.2020 | виконано |
| 5 | Проєктування системи керування колісним роботом | 2.06.2020 – 5.06.2020 | виконано |
| 6 | Проєктування інтерфейсу системи керування колісним роботом | 6.06.2020 – 7.06.2020 | виконано |
| 7 | Реалізація серверу застосунків | 8.06.2020 – 7.07.2020 | виконано |
| 8 | Реалізація системи керування колісним роботом | 8.07.2020 – 10.08.2020 | виконано |
| 9 | Реалізація інтерфейсу системи керування колісним роботом | 11.08.2020 –15.08.2020 | виконано |
| 10 | Представлення отриманих результатів науковому керівнику і узгодження плану подальшого дослідження | 1.09.2020 | виконано |
| 11 | Проєктування системи моделювання руху колісного робота | 15.10.2020 | виконано |
| 12 | Реалізація системи моделювання руху колісного робота | 16.10.2020 –20.10.2020 | виконано |
| 13 | Проведення аналізу можливостей розроблених програмних застосунків | 21.10.2020 – 1.11.2020 | виконано |
| 14 | Оформлення звіту | 15.11.2020 –10.12.2020 | виконано |

Студент _____ Старшеков Д.А.
( підпис ) (прізвище та ініціали)

Керівник роботи _____ Вербицький В.Г.
( підпис ) (прізвище та ініціали)

**Нормоконтроль пройдено**
Нормоконтролер _____ Скрипник І.А.
( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

Старшеков Д.А. Комп’ютерна система керування та стабілізації руху колісних роботів.

**Кваліфікаційна робота магістра складається із введення, 4 розділів і висновків, списку використаних джерел з 25 найменувань. Робота містить 120 сторінок тексту, 53 рисунки, 12 таблиць, 13 лістингів коду.**

Кваліфікаційна робота для здобуття ступеня вищої освіти магістра за спеціальністю 121 – Інженерія програмного забезпечення, науковий керівник В.Г. Вербицький. Інженерний інститут ЗНУ.

Мета дослідження полягає у вивченні проблем з області керування та стабілізації руху колісних роботів, порівняння існуючих рішень, дослідження засобів впливу на рух неголономної колісної системи, дослідженні інструментів моделювання руху колісних роботів та створення програмного забезпечення для моделювання руху колісних роботів.

Досліджено існуючі проблеми з області керування та стабілізації руху колісних роботів. Проведено порівняння існуючих рішень і задач, які їх вирішують. На основі досліджених даних створено програмне забезпечення для керування колісним роботом та моделювання руху. Розроблена система була протестована на фізичній моделі робота.

**Ключові слова:** НЕГОЛОНОМНА КОЛІСНА СИСТЕМА, КОЛІСНИЙ РОБОТ, МОДЕЛЮВАННЯ РУХУ, СТАБІЛІЗАЦІЯ РУХУ, UNITY, C#, WEBRTC,

# ANNOTATION

D. A. Starshekov, Computer-Based Motion Control and Stabilization System for Wheeled Robots.

**The Master's qualifying paper consists of an introduction, 4 sections and conclusions, and references with 25 items. The paper contains: 120 pages of text, 53 figures, 12 tables, and 13 code listings.**

Qualifying paper to obtain a Master's Degree with a major in 121 — Software Engineering, supervisor V. H. Verbytskyi. Engineering Institute of ZNU.

The goal of research is to study the motion control and stabilization problems in case of wheeled robots, compare the existing solutions, research what impacts the motion of nonholonomic wheeled systems, study the tools that simulate the motion of wheeled robots, and create software that allows simulating the motion of wheeled robots.

The existing motion control and stabilization problems in case of wheeled robots have been researched. The existing solutions and problems have been compared. Based on the data being studied, the software used to control wheeled robots and simulate their motion has been created. The developed system has been tested on a physical robot.

**Keywords:** NONHOLONOMIC WHEELED SYSTEM, WHEELED ROBOT, MOTION SIMULATION, MOTION STABILIZATION, UNITY, C#, WEBRTC.

# ЗМІСТ

# ВСТУП

## Актуальність теми

Сфер використання колісних роботів безліч. Роботи можуть доставляти ліки та інші предмети хворим на інфекційні захворювання, контакт з якими є небезпечним. Такі роботи можуть дезінфікувати приміщення ультрафіолетовим променем без допомоги людини. Вони можуть виконувати роль саперів, які мають право на помилку, можуть брати участь у спеціальних операціях, де існує загроза людському життю. Впровадження роботів у суспільстві зазвичай покращує життя людей. На сьогодні є багато компаній, які розробляють роботів для різних цілей: від доставки їжі та прибирання квартири до роботизованих платформ для потреб армії. Впровадження роботів також зумовлено боротьбою з так званим «людським фактором», ціна якого зазвичай значно перевищує ціну робота. Компанії-гіганти, такі як Google, активно стежать за появою та розвитком різних стартапів, пов’язаних з робототехнікою. Існують спеціальні відділи, які знаходять розробників та пропонують придбати або профінансувати їх творіння. Однією з провідних інженерних компаній світу, що спеціалізується на робототехніці, є Boston Dynamics. Багато країн світу надають запити на купівлю роботів для використання у своїх цілях. Так у Сінгапурі роботи-собаки від Boston Dynamics патрулюють парки та сквери під час пандемії COVID-19, а у Сполучених Штатах Америки товари доставляють вуличні роботи-кур’єри від Starship Technologies та багатьох інших компаній.

У перспективі такі роботи зможуть повністю витіснити людей, роботу яких вони виконують. Сьогодні все більше країн світу фінансують державні компанії, створюють фонди та програми для розвитку робототехніки або звертаються за послугами закордонних компаній з метою впровадження роботів. Тому розвиток цього напрямку є максимально перспективним і важливим, а дослідження фізичних та віртуальних моделей таких роботів дають змогу

аналізувати поведінку руху робота у різних середовищах з різними силами, що діють на робота, лише змінивши параметри середовища у програмі, що виконує аналіз.

Саме моделювання середовища та фізичного прототипу, аналіз поведінки при різних ситуаціях є інструментами вдосконалення таких динамічних моделей. Таке моделювання дає змогу визначити проблеми фізичної моделі на етапі конструювання, що значно зменшує фінансові витрати на розробку. Також моделювання дає змогу перевірити, як буде поводити себе прототип, якщо змінити його фізичні складові, такі як вагу, розмір тощо, або змінити середовище, яке також безпосередньо впливає на роботу робота. Отже, перед створенням реального прототипу або його модифікацією інженери мають протестований віртуальний прототип, який допомагає виправити помилки, що були виявленні на етапі тестування у віртуальному середовищі.

## Мета і завдання дослідження

Мета дослідження полягає у вивченні методів керування та аналізу стійкості автономних колісних роботів та розробці необхідного математичного та програмного забезпечення для реалізації програмного руху уздовж замкнутих траєкторій неголономної моделі колісного робота заданої триколісної схеми.

## Об’єкт дослідження

Процеси керування автономним колісним роботом (Robot Tima від компанії Infocom Ltd) при реалізації класичних неголономних зв’язків коліс з опорною поверхнею.

**Предмет дослідження**

Синтез законів керування автономним колісним роботом при маневруванні уздовж програмних кривих, що можуть бути задані в різній математичній формі, та розробка відповідного програмного забезпечення.

**Методи дослідження**

Методи теорії керування рухомими механічними системами, теорії неголономних механічних систем, теорії планування коридорів руху в умовах обмеженого простору (наявності перешкод) та теорії досяжності. Методи математичного моделювання та чисельного інтегрування диференціально-алгебраїчних систем. Методи теорії інформаційних систем.

**Наукова новизна одержаних результатів**

Новизна очікуваних результатів дослідження полягає у виборі нових типів керуючих впливів на неголономну модель колісного робота (моментів інерційного походження) та синтезу відповідних впливів керування, що забезпечують рух по програмній траєкторії системи при збереженні умов реалізації неголономних зв’язків.

**Практичне значення одержаних результатів**

На базі отриманих (прогнозованих) теоретичних результатів можна сподіватись на успішну розробку відповідного програмного забезпечення адаптованого до практичного застосування в умовах керованого руху прототипу колісного робота із заздалегідь визначеною точністю проходження маршруту.

**Глосарій**

*Робот* — автоматичний пристрій, призначений для здійснення різного роду механічних операцій, який діє відповідно заздалегідь закладеною програмою.

*Програмне забезпечення* — сукупність програм системи обробки інформації та програмних документів, необхідних для експлуатації цих програм.

*Неголономна система* — механічна система, на яку, крім геометричних зв'язків, накладаються неголономні.

*Неголономні зв'язки* — кінематичні зв'язки, які не можна звести до геометричних.

*Електропривід* — це електромеханічна система для надання руху із заданими параметрами виконавчим механізмам робочих машин в цілях здійснення їх функцій.

*Nvidia Jetson* — це серія вбудованих обчислювальних плат від Nvidia Corporation.

*Nvidia Corporation* — американський виробник графічних процесорів, відеоадаптерів під торговими марками Riva TNT та GeForce, мультимедійних та комунікаційних пристроїв для ПК та ігрових консолей.

*Персональний комп'ютер* (скорочено ПК) — електронна обчислювальна машина, що призначена для зберігання і переробки інформації, ціна, розміри та можливості якої задовольняють потреби багатьох людей.

*GPS, Система глобального позиціонування* — сукупність радіоелектронних засобів, що дозволяє визначити положення та швидкість руху об'єкта на поверхні Землі або в атмосфері.

*Операційна система*, скорочено ОС (operating system, OS) — це базовий комплекс програм, що виконує керування апаратною складовою комп'ютера або віртуальної машини

*Ядро Linux* — ядро UNIX-подібної операційної системи. Розповсюджується під ліцензією GNU General Public License (GPL), і розробляється людь-

ми з усього світу, що дозволило йому стати одним із найвидатніших прикладів відкритого програмного забезпечення та увійти до числа наймасштабніших проектів з розробки програмного забезпечення.

*Відкрите програмне забезпечення* (open-source software) — програмне забезпечення з відкритим програмним кодом.

*Android* — операційна система і платформа для мобільних телефонів та планшетних комп'ютерів, створена компанією Google на базі ядра Linux.

*Фреймворк* (Framework) — інфраструктура програмних рішень, що полегшує розробку складних систем. Спрощено дану інфраструктуру можна вважати своєрідною комплексною бібліотекою.

*.NET* (раніше названий .NET Core) - це безкоштовна модульна платформа для розробки програмного забезпечення для операційних систем Windows, Linux та macOS. Це крос-платформний наступник .NET Framework. Проект в основному розроблений працівниками Microsoft за допомогою .NET Foundation і випущений під ліцензією MIT.

*WebRTC* (real-time communications — комунікація в реальному часі) — інтернет-протокол із відкритим кодом, призначений для організації голосового та відеозв'язку через інтернет у режимі реального часу.

*SDP* (Session Description Protocol, протокол опису сеансу зв'язку) — мережевий протокол, призначений для опису сеансу передачі потокових даних, включаючи телефонію ТМЗК і VoIP, інтернет-радіо та програми мультимедіа.

*STUN* (Session Traversal Utilities for NAT) — мережний протокол, який використовується в обхід NAT. Дозволяє користувачу, що знаходиться за межами серверу трансляції адрес, визначити свою зовнішню IP-адресу, спосіб трансляції адреси та порт у зовнішній мережі, пов'язаною із визначенням внутрішнього номера порту. Ця інформація використовується для встановлення з'єднання UDP між двома хостами тоді, коли вони знаходяться за NAT-маршрутизатором.

*User Datagram Protocol, UDP* (Протокол датаграм користувача) — один із протоколів в стеку TCP/IP. Від протоколу TCP він відрізняється тим, що працює без встановлення з’єднання. UDP — це один з найпростіших протоколів транспортного рівня моделі OSI, котрий виконує обмін повідомленнями (датаграмами) без підтвердження та гарантії доставки.

*Модель OSI* (базова еталонна модель взаємодії відкритих систем, англ. Open Systems Interconnection Basic Reference Model) — абстрактна мережева модель для комунікацій і розробки мережевих протоколів.

*TURN* (Traversal Using Relay NAT) — це протокол, який дозволяє вузлу за NAT або брандмауером отримувати вхідні дані через TCP або UDP з'єднання.

*JSON* (JavaScript Object Notation) — це текстовий формат обміну даними між комп’ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дає змогу описувати об’єкти та інші структури даних.

*SignalR* — це безкоштовна бібліотека програмного забезпечення з відкритим кодом для Microsoft ASP.NET, яка дозволяє коду сервера надсилати асинхронні сповіщення веб-додаткам на стороні клієнта.

*Воксель* — елемент простору, позначає значення певної величини в клітинках рівномірної просторової ґратки. Аналогічний пікселю, у двовимірних зображеннях.

*Тривимірна графіка, або 3D-графіка* — розділ комп’ютерної графіки, сукупність прийомів та інструментів (як програмних, так і апаратних), призначених для зображення об’ємних об’єктів. Найбільше застосовується для створення зображень, які в подальшому використовуватимуться на екрані або роздруківках в архітектурній візуалізації, кінематографі, телебаченні, відеоіграх, друкованій продукції, а також у науці та промисловості.

*Граф сцени* — загальна структура даних, що зазвичай використовується в застосуваннях для роботи з векторною графікою і в сучасних комп’ютерних іграх, яка впорядковує логічне і часто (але не обов’язково) просторове представлення графічної сцени.

*Сплайн* — функція, область визначення якої розбита на частини, на кожній з частин функція є деяким поліномом.

*Полярна система координат* — двовимірна система координат, в якій кожна точка на площині визначається двома числами — кутом та відстанню.

*Цифрове моделювання* — дослідження об'єктів (явищ, процесів, пристроїв, систем тощо) за допомогою математичних моделей на ЕОМ.

*Комп'ютер, електронно-обчислювальна машина (ЕОМ)* — програмно-керований пристрій для обробки інформації.

*Суперкомп'ютер* — сучасний термін, який використовують для позначення класу наявних найпотужніших комп'ютерних систем.

*Відеокарта* (також графічна карта, графічний адаптер, графічний прискорювач) — електронний пристрій, частина комп'ютера, призначена для генерації та обробки зображень з подальшим їхнім виведенням на екран периферійного пристрою.

*Інтегроване середовище розробки* (ICP, англ. IDE) — комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм.

*ORM* (англ. Object-relational mapping, Об'єктно-реляційна проекція) — технологія програмування, яка пов'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

*Транскомпіляція* — це процес перетворення програмного коду з одної мови програмування на програмний код іншої.

*СКБД* — система керування базами даних, набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних.

# РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ МОДЕЛЮВАННЯ КЕРОВАНОГО РУХУ НЕГОЛОНОМНОЇ МОДЕЛІ КОЛІСНОГО РОБОТА

## 1.1 Неголономна система

Неголономна система — це механічна система, на яку, крім геометричних зв’язків, накладаються й неголономні, тобто такі, на які додатково накладаються кінематичні зв’язки, що неможливо привести до геометричних. Прикладом таких кінематичних зв’язків є умови відсутності поздовжнього та поперечного прослизання при русі колеса по опорній поверхні.

Існує велика різноманітність кінематичних схем, що забезпечують різні режими руху. Такі схеми можна класифікувати так:

1. Класичні, які використовують звичайні колеса.
2. Комбіновані, які використовують для руху звичайні та роликові колеса.

Насамперед неголономні системи є механічними системами. Неголономні системи виконують роль моделей для багатьох технічних об’єктів, але не тільки механічних, а й електромеханічних. Необхідно відзначити, що зі стрімким розвитком робототехніки у світі виникла потреба вирішення задач та дослідження стабілізації, динаміки та стійкості неголономних механічних систем, які складають значну частину базових моделей різних за видами колісних роботів. Також через популярність нових самокатів типу Segway та подібних до такого типу самокатів збільшилася зацікавленість задачами неголономної системи, через те що саме вона описує такі транспортні засоби.

Зазвичай при побудові рівнянь руху таких систем використовують або рівняння Лагранжа першого роду з невизначеними множниками, або рівняння у квазікоординатах Гамеля.

Рівняння руху неголономних систем можуть мати певну специфіку — у вигляді неізольованої множини положень рівноваги. Так, таку структуру не-

обхідно враховувати як при вирішенні задач стійкості, так і при вирішенні задач стабілізації.

### 1.2 Проблеми математичного моделювання

Метод математичного моделювання дає змогу поєднати переваги теорії та застосування емпіричних методів. З одного боку, математика дозволяє описати об’єкт та дослідити його достатньо детально від його властивостей до поведінки при різних ситуаціях. Це саме ті переваги, які надає теорія. З іншого боку, обчислювальні експерименти з математичною моделлю за допомоги сучасних комп’ютерів та суперкомп’ютерів дають змогу у достатньому обсязі дослідити об’єкт, що не було б можливим при використанні виключно теоретичних відомостей та теорії взагалі. Це переваги застосування емпіричних методів. Зазначивши цю інформацію, можна дати моделюванню таке визначення: моделювання — це один з загальнонаукових методів пізнання навколишнього світу, який використовується на емпіричному та теоретичному рівні. Водночас при створенні та дослідженні моделі можуть використовуватися всі інші методи пізнання та аналізу.

Дослідження руху динамічних моделей є складною задачею, яку неможливо ефективно вирішити без використання математичного моделювання, моделювання у тривимірній графіці та чисельних методів. Аналіз математичної моделі дає змогу виявити проблеми та недоліки на етапі проектування. Крім того, математична модель допомагає виявити проблеми при модифікації фізичної моделі. Все це значно зменшує витрати на розробку та модифікацію робота. Отже, зрозуміло чому математичне моделювання стало невід’ємною частиною розробки складних технічних систем та подальшого їх дослідження.

Задачу математичного моделювання можна розділити на:

1. Дослідження динамічних неголономних систем колісних роботів.
2. Дослідження взаємодії таких систем з навколишнім середовищем.

У загальному випадку дослідження руху динамічних моделей полягає у розробці математичної моделі робота та дослідженні цієї моделі. Після чого математична модель вдосконалюється шляхом дослідження та порівняння результатів, які отримали під час тестування фізичної та математичної моделей. Необхідно відзначити, що цей процес є досить складним через високу складність динамічних систем колісних роботів. У такому разі головною вимогою математичної моделі є логічний зв'язок між результатами чисельного аналізу та результатами тестування фізичної моделі, для чого математична модель повинна виконувати дві умови:

1. У математичній моделі повинні виконуватися фундаментальні закони збереження енергії, імпульсу та маси.
2. Математична модель повинна достовірно відображати сутність об'єкта, що аналізується.

Також необхідно відзначати, що дослідження динамічної системи колісного робота без дослідження взаємодії такої системи з навколишнім середовищем не є ефективним, якщо мати на увазі повний аналіз. Серед різновидів руху, які розглядає сучасна теорія колісних транспортних засобів, є багато винятків, таких як рух колісної моделі при екстреному гальмуванні, рух при моментальному блокуванні коліс, рух, при якому колеса відірвані від землі, або на головні осі діє сила, яка значно перевищує максимально допустиму, та багато інших ситуацій. Всі ці ситуації є складними та потребують окремого детального аналізу.

### 1.3 Проблеми комп'ютерного моделювання

Комп'ютерне 3D-моделювання — це моделювання, яке для створення та подальшого відображення об'єкта використовує 3D-графіку. Однією з очевидних переваг використання комп'ютерного 3D-моделювання, як порівняти з двомірним кресленням, є побудова точної за розмірами 3D-моделі. Отже, за допомоги сучасних 3D-редакторів можна оглядати 3D-модель з усіх

ракурсів та маніпулювати нею, як справжнім об’єктом. Також необхідно відзначити, що створення моделей саме у трьох вимірах є більш прийнятним для людини, через те що ми живемо у тривимірному середовищі та мислимо трьома вимірами. Тому продуктивність інженерів, які працюють с тривимірними моделями, є значно вищою.

Методи та інструменти, що необхідні для створення 3D-моделі, значно змінилися в останній час. Це зумовлено швидким ростом цієї галузі та вдосконаленням як апаратного, так і програмного забезпечення для роботи з об’єктами у тривимірному просторі. З вдосконаленням галузі значно змінились і вимоги до якості 3D-моделей. У 2020 році на ринку комплектуючих модулів персонального комп’ютера можна придбати графічні адаптори, потужність яких дає можливість ефективно виконувати паралельні обчислювання, завдяки великій кількості ядер адаптора. Тому 3D-моделі стали більш детальними, а різні інтегровані середовища розробки, такі як Unity 3D або Unreal Engine, відповідають за різного роду сили, що можуть діяти на таку модель.

Розвитку 3D-моделювання значно сприяла ігрова індустрія. З кожним роком відеоігри отримують детальніші 3D-моделі, кращу анімацію, досконалішу «фізику» тощо. Це дає змогу без великих зусиль завдяки сучасним інтегрованим середовищам розробки створити об’єкт у віртуальному середовищі, який буде здатний реагувати на навколишнє, для нього віртуальне, середовище, яке так само може застосовувати на об’єкт сили тяжіння, пружності, гравітаційні сили, тощо. Проте звичайно віртуальна «фізика» не є досконалою, тим паче, що одним з основних рушіїв розвитку виступають відеоігри, тому головною метою для таких середовищ розробки є не точне математичне та фізичне представлення об’єкта, а ті бізнес-задачі, які ставлять перед собою розробники відеоігор. Кожне з інтегрованих середовищ розроби має свій або спільний модуль, що відповідає за «фізику». Такі модулі існують як у відкритому вигляді, так і у закритому: корпоративні, платні тощо. Кожна з компаній намагається вдосконалити цей модуль, щоб створити найкращу фізичну

взаємодію об’єктів, що надає реалістичності. Безумовно, такі інструменти надають величезні можливості у моделюванні об’єктів, але водночас вони можуть значно обмежити розробника. Як саме ці обмеження можуть впливати на моделювання? Коли ви використовуєте фізичні можливості «з коробки», ви, безумовно, отримуєте багато переваг, але якщо вам потрібно щось змінити або навіть подивитися як той чи інший фізичний закон реалізовано, ви можете потрапити в таке становище, в якому ви не зможете змінити поведінку об’єкта та навіть не отримаєте інформацію про формули, які були застосовані в тій чи іншій функції.

### 1.4 Вплив розвитку програмного забезпечення на моделювання

Якщо досліджувати вплив апаратного забезпечення на моделювання в цілому, то часто можна зустріти визначення «апаратне прискорення». Апаратне прискорення — це застосування комп’ютерного апаратного забезпечення для виконання деяких ефективніше, ніж є можливим у програмному забезпеченні, яке виконується на центральному процесорі загального призначення.

Прикладами апаратного прискорення може бути функціонування передачі блоків бітів у графічних процесорах та апаратне прискорення регулярних виразів для контролю спаму у серверній промисловості. Традиційно архітектура центральних процесорів була розрахована на послідовне виконання команд. Сучасні центральні процесори мають багатоядерну архітектуру, але програмне прискорення навіть для таких процесорів є актуальним. Якщо апаратне забезпечення, що реалізує апаратне прискорення, знаходиться в окремому від центрального процесора вузлі, то воно називається апаратним прискорювачем. Але часто такі прискорювачі отримують більш конкретні назви, такі як криптографічний прискорювач або 3D-прискорювач. Але ці терміни застарілі і сьогодні вони трансформувалися у менш описові терміни, такі як відеокарта та мережевий адаптер.

Розглянемо особливості архітектури центрального процесора (CPU) та графічного модулю, відеокарти (GPU). Коли треба щось порівняти ми повинні визначитися з метрикою. Для центральних процесорів часто основними характеристиками є частота, кількість ядер та розмір кешу. Але основною характеристикою чіпу, мета роботи якого елементарні математичні обчислення — це яку кількість таких операцій в одиницю часу він може виконати. Однією з таких метрик є кількість операцій з плаваючою комою в секунде, так звані flops. Така метрика добре підходить для порівняння графічного та центрального процесорів. На рисунку 1 показано зростання flops з плином часу модифікації CPU та GPU.

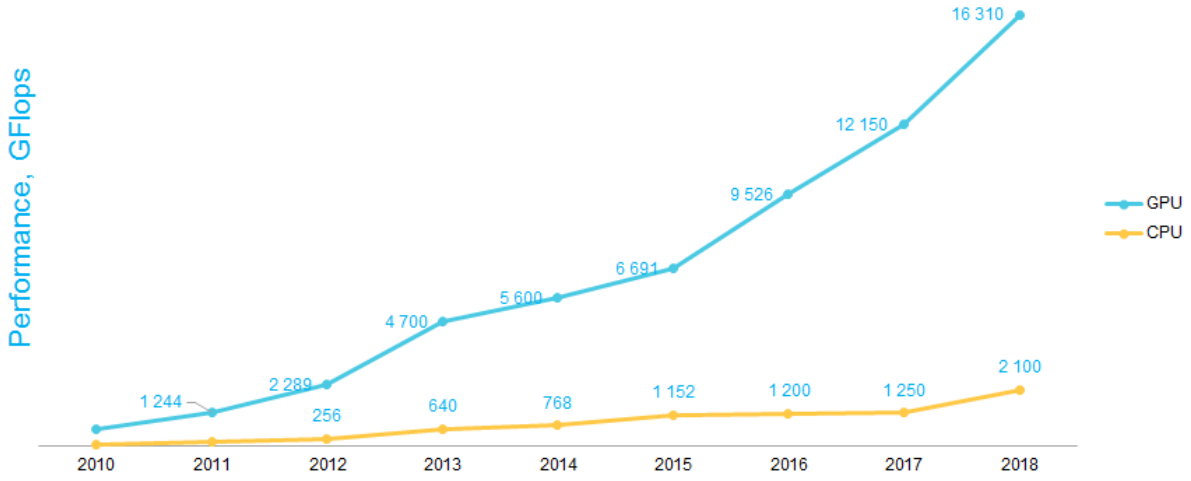



*Рис. 1 Зростання flops для CPU та GPU*

Виглядає так ніби розрахунки краще проводити саме на відеокартах, але це не зовсім так. Розглянемо архітектури процесорів. На рисунку 2 показана архітектура CPU, На рисунку 3 показана архітектура GPU.

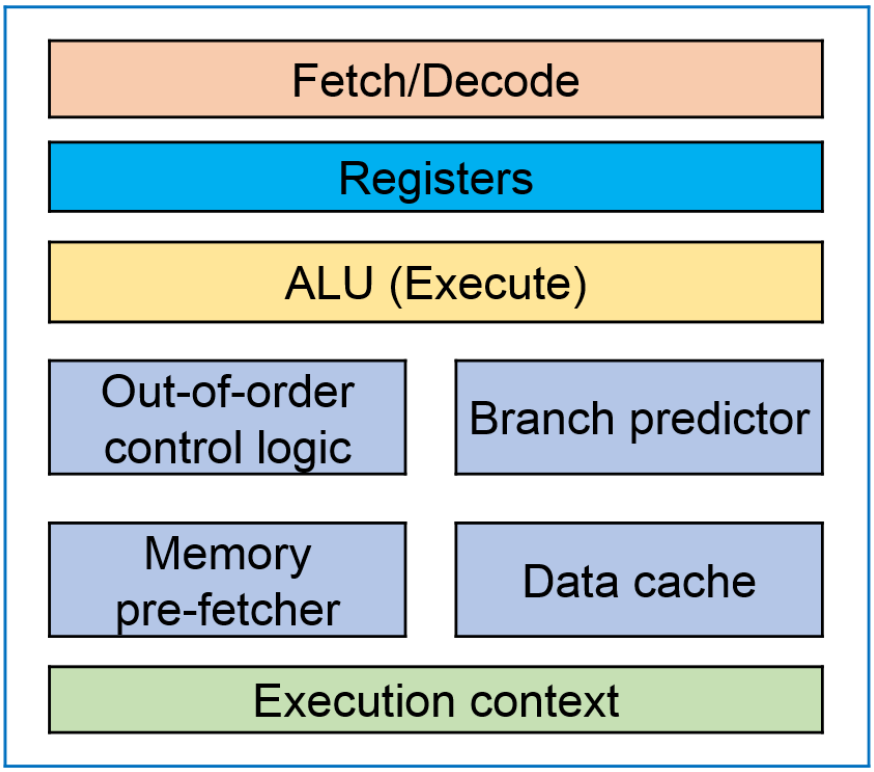



*Рис. 2 Архітектура CPU*

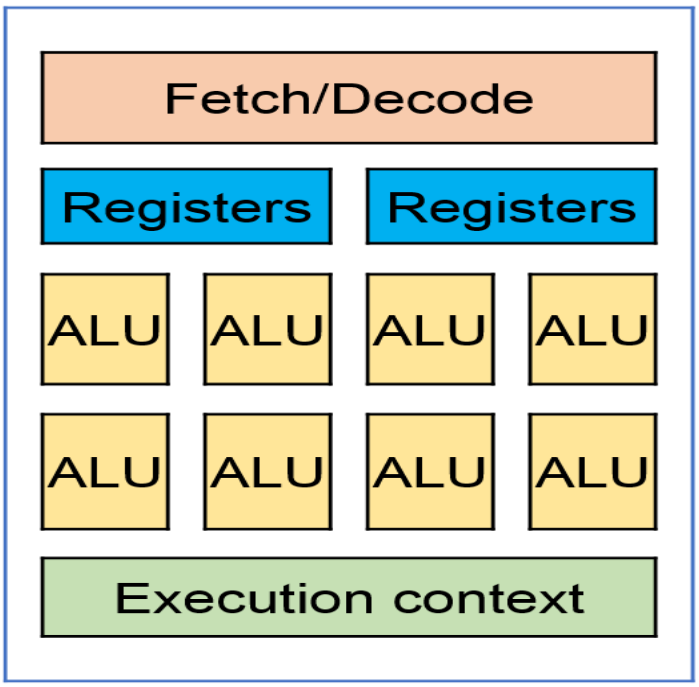



*Рис. 3 Архітектура GPU*

Як видно на схемах CPU має ядро та багато допоміжних компонентів. В той час як GPU має багато множину обчислювальних ядер. Для сучасних

відеокарт кількість ядер досягає десяти тисяч. У свою чергу ядра об’єднані у блоки, зазвичай по 32, та мають спільні елементи. Архітектура GPU значно простіша ніж CPU, що з одного боку надає переваги а з іншого накладає обмеження.

Обмеження, що накладає така архітектура:

1. Усі ядра виконують однакові інструкції але з різними даними. Такі обчислення називаються Single-Instruction-Multiple-Data або SIMD.
2. При обчисленні за рахунок GPU ми не можемо використати тільки одне ядро, буде задіяно цілий блок ядер.
3. Через відносно спрощений набір умовних блоків GPU не так ефективно виконує алгоритми, які мають багато розгалужень та у цілому складні або складно реалізовані.

Розглянемо головні переваги, які надає спрощена архітектура GPU:

1. Прискорення SIMD обчислень, прикладом якого є поелементне додавання матриць.

Розглянемо приклади приведення класичних алгоритмів до SIMD-виду. Перший алгоритм — трансформація, маємо два масиви A та B та кожному елементу масиву A додаємо елемент масиву B. Класичний приклад з лінійною швидкістю буде виглядати так:

*Лістинг 1 Алгоритм трансформації у лінійній формі*

```
void func(float *A, float *B, size)
{
   for (int i = 0; i < size; i++)
   {
       A[i] += B[i]
   }
}
```

*Лістинг 2 SIMD-еквівалент алгоритму трансформації*

```
void func(float *A, float *B, size)
{
   int i = threadIdx.x;
   if (i < size)
      A[i] += B[i]
}
```

Діаграма порівняння показана на рисунку 4.

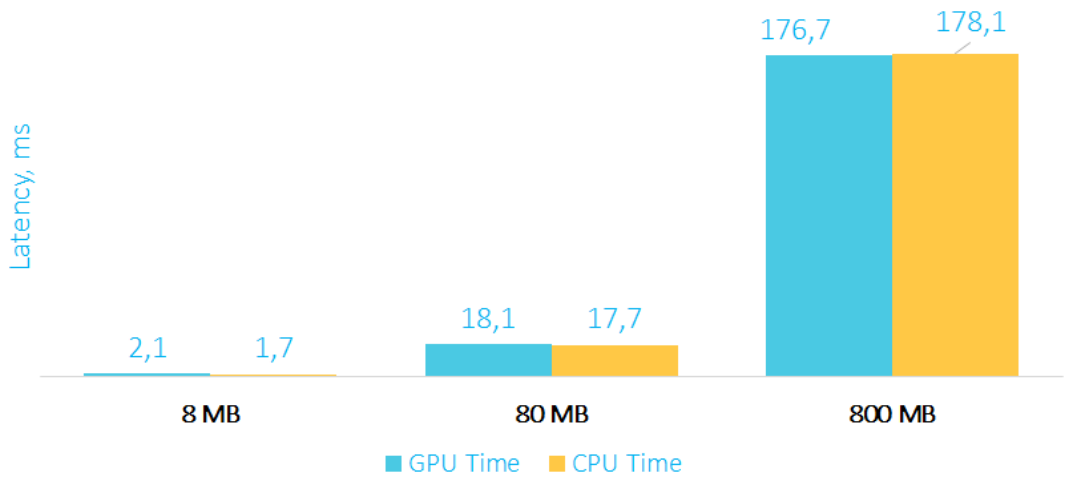



*Рис. 4 Час виконання трансформації на GPU та CPU*

Наступний приклад алгоритму — агрегація. На рисунку 5 показано схему алгоритму агрегації у SIMD-виді.

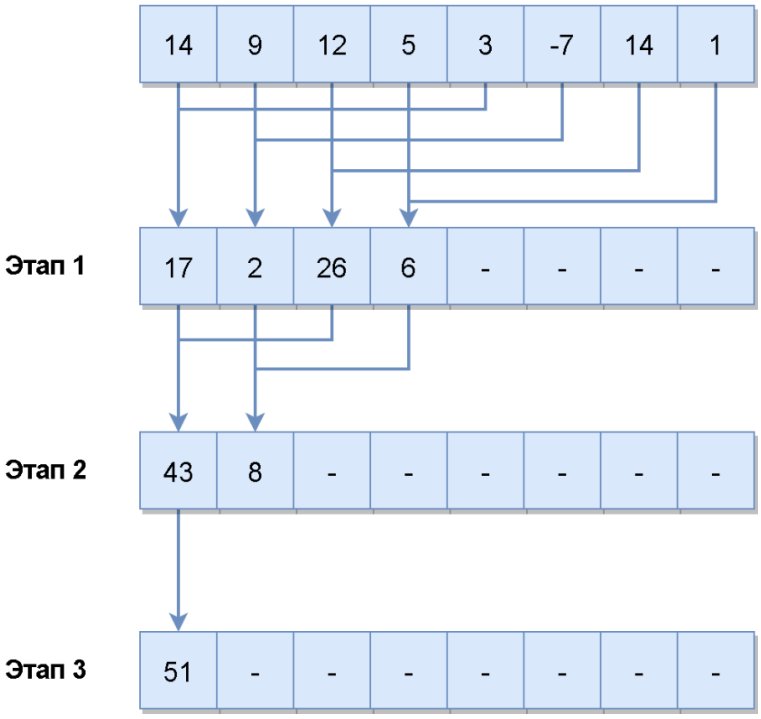



*Рис. 5 Алгоритм агрегації у SIMD-виді*

Маємо масив з n елементів. На першому етапі запускаємо n/2 потоків та кожен потік додає по два елементи за одну ітерацію. Тобто після закінчення першої ітерації буде додано половину масиву. Далі алгоритм рекурсивно повторюється для масиву, що утворився на попередньому кроці, доки не отримаємо результат агрегації останніх двох елементів. Отже можна констатувати, що чим менший розмір масиву, тим меншу кількість паралельних потоків ми можемо використати. Тому слід використовувати такий підхід тільки для масивів великого розміру. Такий алгоритм можна використовувати для пошуку екстремуму, простого пошуку або обчислення суми елементів. Діаграма порівняння показана на рисунку 6.

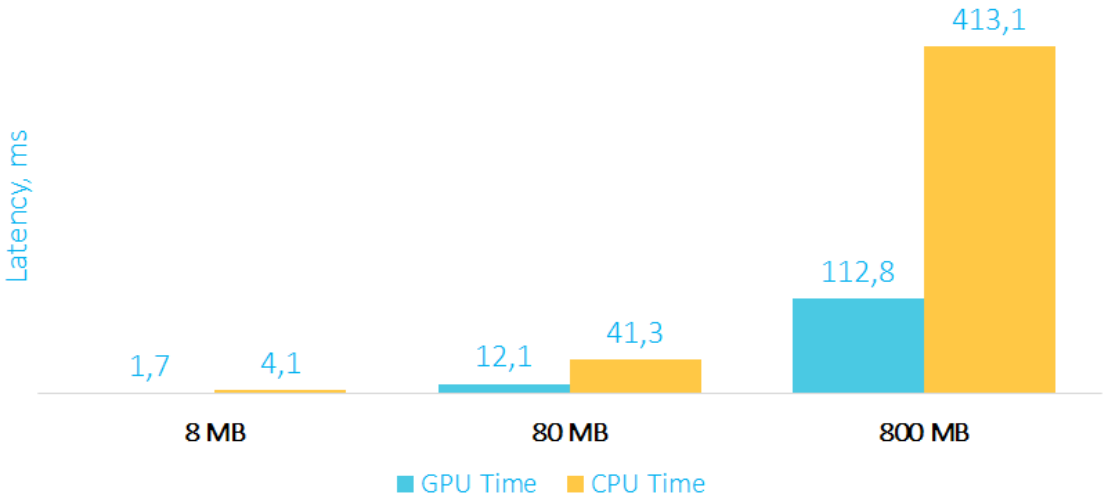



*Рис. 6 Час виконання агрегації на GPU та CPU*

Алгоритми сортування дещо складніші. Виділяють два основні алгоритми сортування SIMD-виду:

1. Bitonic-sort.
2. Radix-sort.

Відзначимо, що алгоритм Radix-sort зустрічається частіше. Діаграма порівняння показана на рисунку 7.

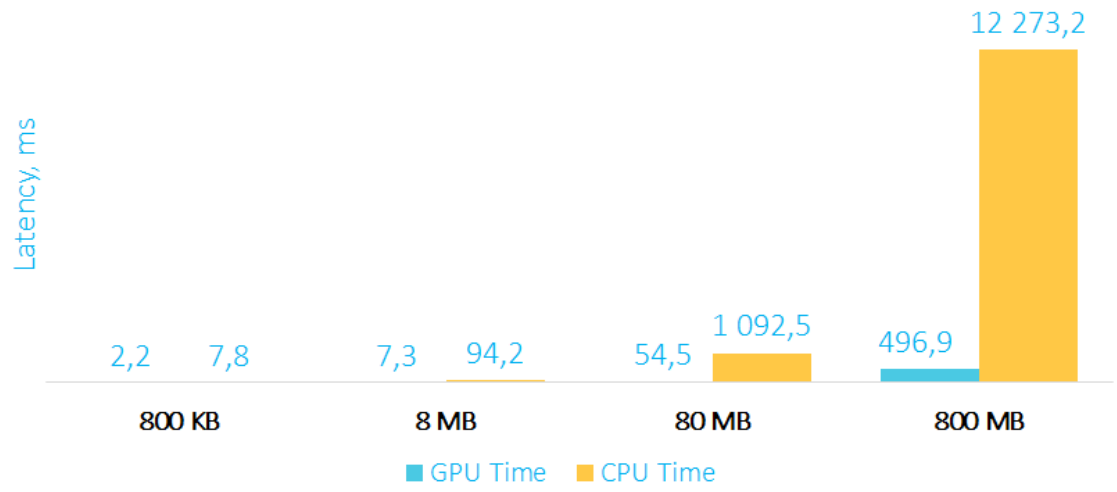



*Рис. 7 Час виконання сортування на GPU та CPU*

Висновки, які можна зробити на основі даного матеріалу це те, що навіть такі нелінійні алгоритми, як сортування можливо привести до SIMD-

виду та що дійсно правильне використання обчислень на GPU може значно покращити ефективність вашого програмного забезпечення.

## 1.5 Засоби та інструменти математичного моделювання

### 1.5.1 Пакет прикладних програм MATLAB

MATLAB — найпопулярніший пакет прикладних програм, який відрізняються своєю ефективністю, орієнтацією на продуктивність виконання програм та величезною кількістю пакетів, які здатні вирішити майже будь-яку математичну задачу. Ефективність MATLAB обумовлена її архітектурою, яка націлена на матричні обчислення, програмною емуляцією паралельних обчислень та спрощенням методів задання циклів. Останні версії продукту підтримують 64-розрядні мікропроцесори та багатоядерні мікропроцесори, що забезпечує найвищі показники швидкості обчислень та швидкості математичного моделювання.

У MATLAB визначають дуже зручну інструменти роботи з багатовимірними масивами, потужні інструменти діалогу, графіки та комплексної візуалізації обчислень. MATLAB дійсно має велику кількість різних пакетів, але навіть без встановлення додаткових пакетів, MATLAB — це потужне операційне середовище для виконання величезної кількості математичних та науково-технічних обчислень та створення користувачами особистих пакетів та бібліотек.

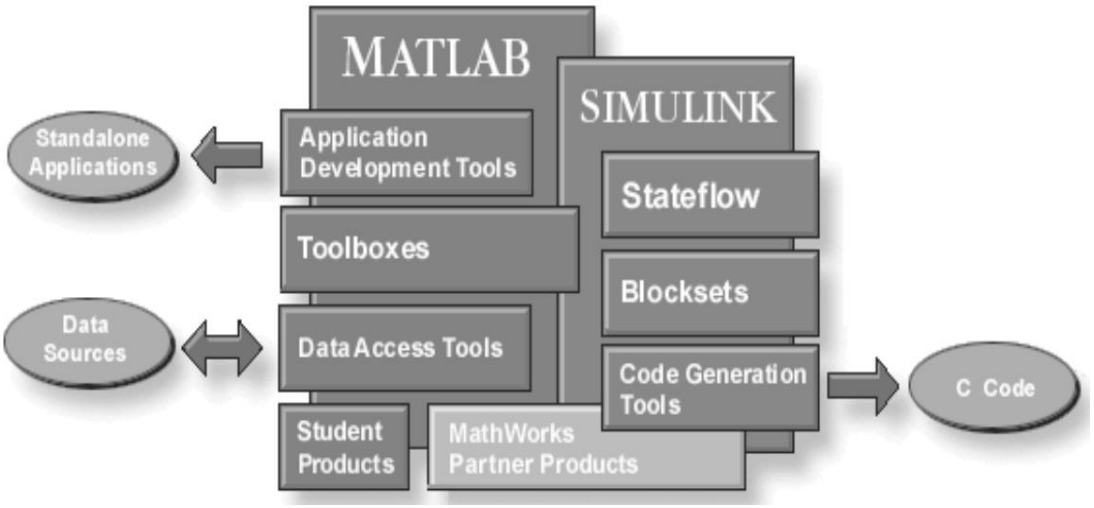



*Рис. 8 Структура системи MATLAB*

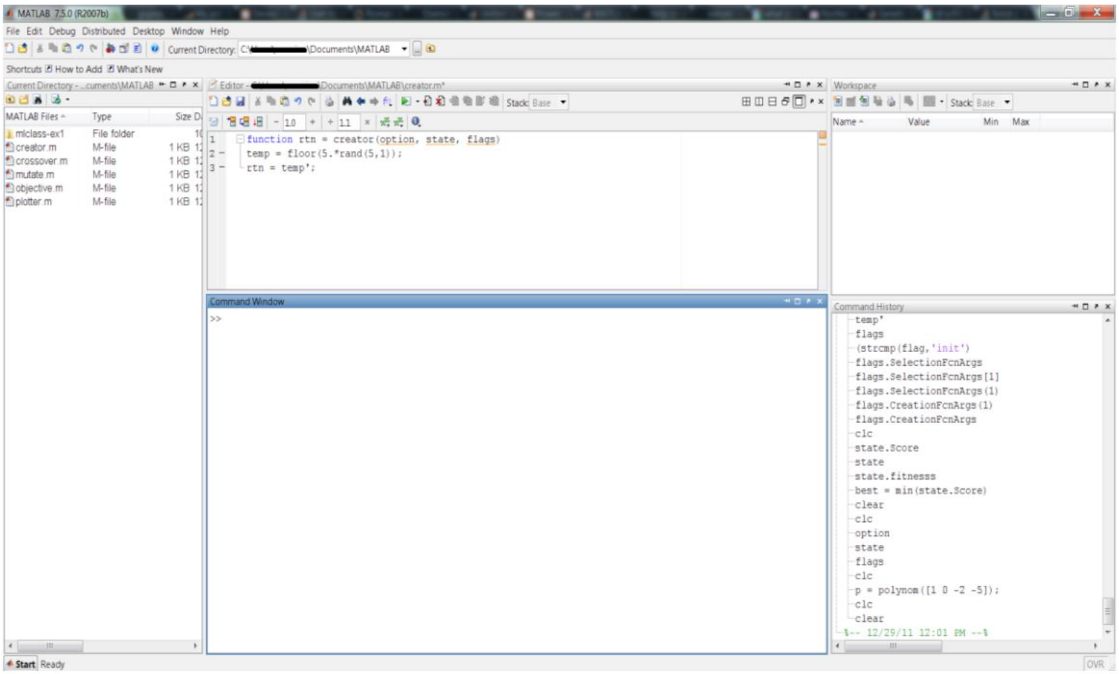


*Рис. 9 Інтерфейс MATLAB*

### 1.5.2 Пакет прикладних програм MAPLE

Maple — це пакет прикладних програм, одним з головних застосувань якого є аналітичні розрахунки. Maple містить більше двох тисяч команд, які дозволяють вирішувати задачі прикладних наук, таких як алгебра, геометрія, математичний аналіз, статистика, математична фізика, а також обчислювати диференційні рівняння. Основним об’єктом у Maple є формула та дію з нею.

Робота у Maple має сесійний режим, тобто користувач вводить команди, формули або вирази, які сприймаються умовно та виконуються Maple.

Робочий інтерфейс поділяються на три частини:

1. Область для введення (команди вводяться після символу >)
2. Область виведення, яка містить результати обробки а виконання команд у вигляді аналітичних виразів та графічних об’єктів або повідомлень про помилку.
3. Область текстових коментарів, яка містить будь-яку текстову інформацію, яка може пояснювати команди, що виконуються.

Для того щоб відмінити усі створені команди та перейти до нового сеансу слід використати команду restart.

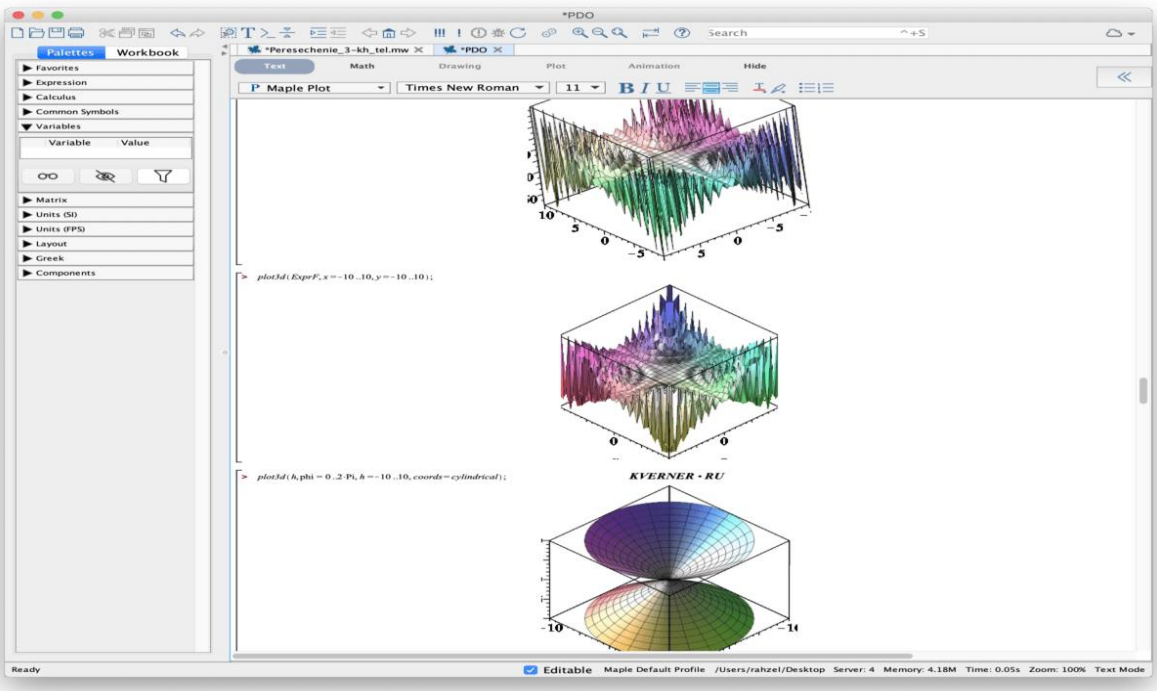



*Рис. 10 Приклад інтерфейсу Maple*

## 1.6 Засоби та інструменти комп’ютерного моделювання

### 1.6.1 Операційна система роботів ROS

Robot Operating System (ROS) — це гнучкий фреймворк для розробки програмного забезпечення роботів. В загальному розумінні ROS — це набір різного типу модулів, інструментів та програмних бібліотек, які розроблені для покращення розробки програмного забезпечення для роботизованих систем різного рівня. Головна ідея створення ROS полягає у наданні стандарту та стимуляції спільної розробки програмного забезпечення для роботизованих систем.

Платформи у роботизованих системах поділяють на програмні та апаратні. Відповідно, програмна платформа поєднує у собі бібліотеки, модулі та інструменти для розробки програмного забезпечення, які можуть виконувати типові задачі, а саме низькорівнева робота з різними приладами, навігація, розпізнавання об’єктів та образів, підключення бібліотек, модулі відлагодження тощо. Також необхідно відзначити, що апаратні платформи є сумісними з програмними, що надає змогу розробникам, які ніколи не працювали з обладнання на низькому рівні, створювати програмне забезпечення, не витрачаючи часу на роботу з приладами та обладнанням. Загальні інтерфейси та методи взаємодії з обладнанням дозволяють швидко та ефективно розробити модуль, що значно прискорює розвиток репозиторія модулів ROS. Найбільш активні платформи показано у Таблиці 1:

Таблиця *1*

*Найактивніші платформи для робототехніки*

| | |
|---|---|
| MSRDS10 | Microsoft Robotics Developer Studio, Microsoft - U.S. |
| ERSP11 | Evolution Robotics Software Platform, Evolution Robotics - Europe |
| ROS | Robot Operating System, Open Robotics12 - U.S. |
| OpenRTM | National Institute of Adv. Industrial Science and Technology (AIST) - Japan |
| OROCOS | Europe |
| OPRoS | ETRI, KIST, KITECH, Kangwon National University - South Korea |

Які переваги серед усіх інших платформ має платформа ROS та чому була обрана саме ця платформа? Однією з найважливіших характеристик будь-якої платформи є кількість та активність розробників, які її використовують, кількість модулів та бібліотек, а також простота у використанні. ROS, безумовно, є платформою, що стрімко розвивається та має досить велику кількість користувачів, а також великий репозиторій модулів, що й вплинуло на вибір платформи.

Що ви отримуєте, використовуючи платформу? По-перше, це можливість повторного використання програмних модулів, бо кожен з модулів легко інтегрується та виконується в іншому застосунку. Водночас проблема залежностей є доволі автоматизованою. По-друге, платформа ROS надає розвинуті інструменти відлагодження, 2D- та 3D-візуалізації. По-третє, ви отримуєте готовий протокол комунікації. Однією з основних проблем комплексних застосунків для робототехніки є проблема комунікації всередині одного застосунку. Для цього ROS має все необхідне. Кожен з розроблених або отриманих з репозиторія модулів може бути представлений як окремий процес, який взаємодіє з іншим процесом завдяки мережевим протоколам комунікації. Такий підхід дає змогу створювати прості у використанні, незалежні модулі, які є простими у повторному використанні. Завдяки цьому можна за-

пускати, відлагоджувати та модифікувати такі модулі на будь-якому пристрої.

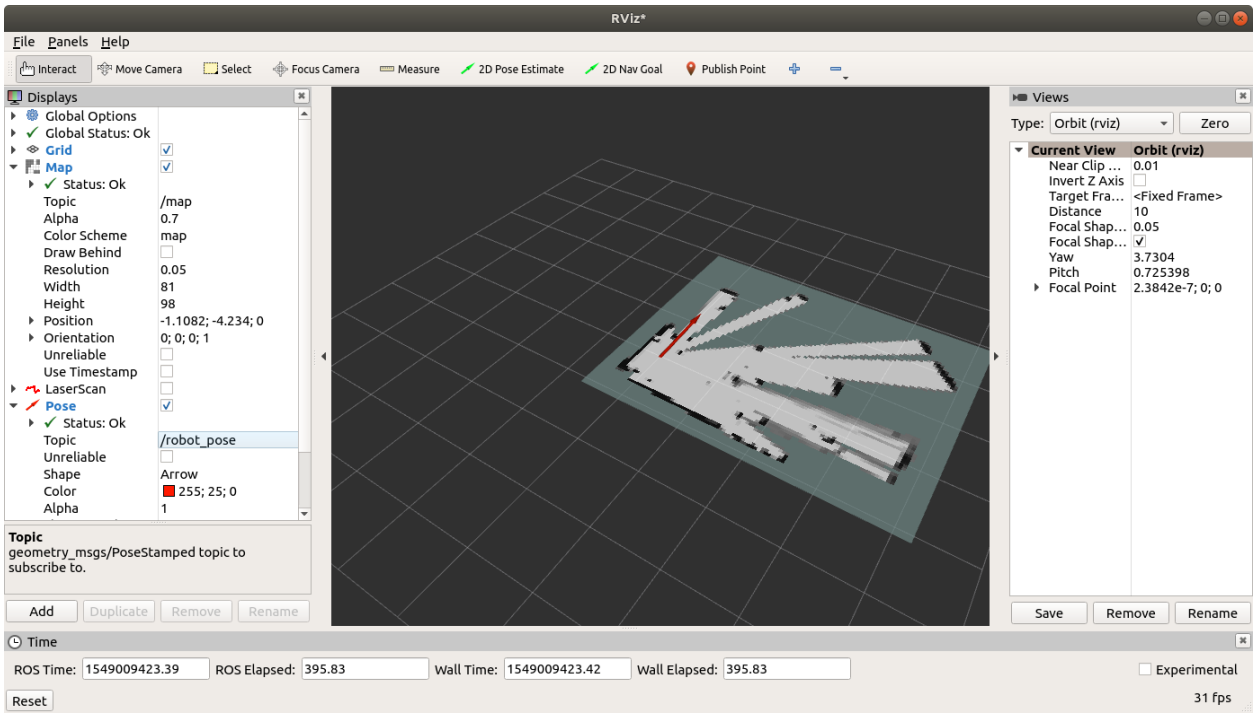



*Рис. 11 Приклад інтерфейсу ROS*

### Базові поняття ROS

### Майстер (Master)

Майстер виконує роль сервера імен для забезпечення підключення між собою різних вузлів. Команда roscore запускає сервер майстра, і після цього до нього можуть підключитися і зареєструватися вузли ROS. Зв'язок між вузлами (обмін повідомленнями) є неможливим без працюючого сервера майстра. При запуску ROS roscore майстер буде запущено за адресою URI та встановлено у змінній оточення ROS _ MASTER _ URI. За умовчанням адреса використовує IP-адресу локального ПК і номер порту 11311.

### Вузол (Node)

Поняття «вузол» належить до найменшої робочої одиниці, що використовується в ROS. Можна провести аналогію з однією програмою. ROS рекомендує створити один вузол для кожного завдання, що дозволить легше її використовувати у інших проєктах. При виконанні вузол реєструє інформацію про себе на майстрі (майстер-сервер). Зареєстрований вузол може взаємодіяти з іншими вузлами (отримувати і відправляти запити). Важливо відмітити, що обмін повідомленнями між вузлами працює без участі майстра (з'єднання між вузлами відбувається без посередників). Майстер забезпечує тільки єдиний простір імен для вирішення питання щодо того, як підключитися до конкретного вузла. Адреса запуску вузла отримується зі змінної оточення ROS _ HOSTNAME, яка має бути визначена до запуску. Порт встановлюється на довільне унікальне значення.

**Пакет (Package)**

Пакет є основною одиницею ROS. Будь-який застосунок ROS формується в пакет, в якому визначаються:

1. Конфігурація пакета.
2. Вузли, необхідні для роботи пакета.
3. Залежності від інших пакетів ROS.

Робота з пакетами ROS дуже схожа на роботу з пакетами Linux. Пакет ROS можна отримати готовим для виконання як з репозиторія пакетів, так і завантажити та скомпілювати з початкових кодів. Пошук доступних пакетів ROS можливий на сторінці за таким посиланням: http://wiki.ros.org/.

**Повідомлення (Message)**

Вузли відправляють і приймають дані між собою відповідно до заданого формату. Ці дані називають повідомленнями, а опис — типом повідомлення. Повідомлення можуть бути як простих типів (`integer, float, boolean`), так і можуть мати складну структуру, що містить вкладені повідомлення і масиви повідомлень. Наприклад, для повідомлення з координатами об’єкта (XYZ) є існуючий тип повідомлення geometry_msgs/Point.msg, який описується так:

1. float64 x.
2. float64 y.
3. float64 z.

**Топік (Topic)**

Топік — це один з видів обміну повідомленнями, який буквально схожий на тему в розмові. Вузол видавця (publisher) спочатку реєструє свою тему на майстрі, а потім починає публікувати повідомлення в цю тему (топік). Вузли підписників, які бажають отримувати інформацію з цієї теми за допомогою майстра, отримують адресу цієї теми і далі отримують повідомлення з цього топіку.

**Видавець (Publisher)**

Видавцем називається процес, який розсилає повідомлення у межах створеного топіку для інших вузлів. Один вузол може містити декілька видавців, що публікують дані в різні топіки.

**Підписник (Subscriber)**

Підписником називається процес, який отримує повідомлення з певного топіку. Підписник (Subscriber) реєструється на майстрі (Master), вказуючи, які топіки підписник (Subscriber) бажає отримувати. Після цього видавець (Publisher) починає відправляти повідомлення підписника. Зв’язок з топіком для підписника є асинхронним (видавець публікує повідомлення незалежно від статусу підписників). Цей тип взаємодії зручно застосовувати для роботи з датчиками, які безперервно передають отримані значення.

**Сервіс (Service): Сервіс Клієнт та Сервіс Сервер**

Сервіс — це модель комунікації, яка працює за принципом синхронного двонаправленого зв’язку між клієнтом (Service Client), який запитує дані, і сервером (Service Server), який відповідає на запити.

**Сервіс Сервер (Service Server)**

Сервіс Сервер — це вузол комунікації (процес), який отримує запит, обробляє дані і передає назад відповідь. Запит і відповідь є звичайним повідомленням (Message).

**Сервіс Клієнт (Service Client)**

Сервіс Клієнт — це вузол комунікації (процес), який створює запит на Сервісі Сервері (Service Server) і отримує відповідь після виконання запиту. Ця модель взаємодії застосовується для віддаленого виконання різних операцій у межах різних вузлів.

**Дія (Action), Action Goal, Action Result, Action Feedback**

Дія є моделлю зв’язку, яку використовують для асинхронного двонаправленого зв’язку. Дія використовується там, де вимагається більше часу для відповіді після отримання запиту, а також проміжні відповіді до тих пір, поки результат не буде досягнутий. Структура протоколу дії (Action) схожа на структуру сервісу (Service), проте до протоколу внесені додаткові параметри: завдання (Action Goal), зворотній зв’язок (Action Feedback) та результат (Action Result).

### 1.6.2 Інтегроване середовище розробки Unity 3D

Unity посідає далеко не останнє місце серед ігрових IDE та доволі активно застосовується як відомими розробниками, так і незалежними студіями. Тому актуальним є питання визначення сильних та слабких сторін IDE та доцільності його застосування у різних проєктах.

Unity являє собою дещо більше ніж звичайний 3D-редактор, адже це своєрідне середовище для розробки комп’ютерних ігор, яке є сукупністю програмних засобів, що використовуються при створенні ПЗ, маються на увазі текстовий редактор, компілятор, вбудоване середовище відладки тощо. Unity є максимально зручним у використанні, що дає змогу створювати ігри дійсно просто та комфортно. Мультиплатформеність двигуна IDE також дає змогу розробникам охопити якомога більшу кількість ігрових платформ та операційних систем.

Говорячи більш детально, Unity 3D дозволяє розробляти ігри навіть за відсутності якихось особливих у цій сфері знань. Це стає можливим через використання компонентно-орієнтованого підходу, завдяки якому розробник створює об’єкти (наприклад, головного героя) і до них додає різні компоненти (наприклад, візуальне відображення персонажа і засоби управління ним). Інтерфейс Drag & Drop разом із функціональним графічним редактором дають змогу малювати карти, розставляти об’єкти у реальному часі та одразу ж тестувати отриманий результат.

Не можна не зазначити ще одну важливу перевагу використання Unity, а саме наявність бібліотеки асетів та плагінів, які значно прискорюють процес створення гри. Вони знаходяться у відкритому доступі та їх можна з легкістю імпортувати чи експортувати. Можна навіть додати цілі заготовки: рівні, ворогів, патерни поведінки штучного інтелекту та багато іншого. Власне більшість асетів є безплатними, хоча існують і ті, що запропоновані за невелику суму, тож, маючи бажання, їх можна придбати та створити повністю власний контент. Надалі свою роботу можна оприлюднити у Unity Asset Store та отримувати певний прибуток.

Підтримка великого різновиду платформ та технологій — це ще одна особливість Unity 3D, бо є можливість перенесення створених ігор між OS Windows, Linux, OS X, Android, iOS, на консолі сімейств PlayStation, Xbox, Nintendo, на VR- і AR-пристрої. Unity підтримує DirectX та OpenGL, працює з усіма сучасними ефектами рендерингу та навіть з такою новинкою, як трасування променів у реальному часі.

Фізика твердих тіл, ragdoll і тканин, система Level of Detail, колізії між об’єктами, складні анімації — усе це відкрите до реалізації при застосуванні двигуна. Певний час мала місце думка, що зазначене IDE підходить виключно для невеликих інді-ігор та не може видавати гарне зображення. На сьогодні це скоріше стереотип, аніж реальний стан речей. Аби змінити думку достатньо буде подивитись на популярні проекти, реалізовані за допомогою цього IDE.

У підсумку Unity представлений у двох версіях: за передплатою та лімітована безкоштовна. У безкоштовній версії логотип Unity додається на старті гри, але це не єдине обмеження. Прибуток, що приносить гра, створена у безкоштовній версії Unity, не повинен перевищувати 100 тисяч доларів США на рік. Власне кажучи, професійна версія за передплатою не є чимось неосяжним навіть для придбання командою початківців та коштує 125 доларів США на місяць, що, порівнюючи з аналоговими IDE, не так багато. До того ж, базова безкоштовна версія пропонує ті самі функції, що й професійна.

Логічним постає питання: чи все так ідеально, чи існують певні недоліки, що роблять Unity не такою привабливою для розробників? Звичайно, якщо команда, наприклад, прагне створити щось більше, аніж звичайний «клікер» чи «платформер», доведеться пошукати гарного програмного інженера, що володіє C#, аби той написав компоненти та скрипти для подальшого застосування у грі.

Рухаючись далі, з’являється новий недолік, а саме повільність. Продуктивність гри може зазнати певних проблем, якщо мається на увазі масштабний проект з великою кількістю складних сцен та компонентів. Як наслідок, доведеться видалити деякі елементи чи витратити багато цінного часу на оптимізацію продукту. Окрім цього, продукти на базі Unity досить важкі, говорячи навіть про найпростіші ігри. Треба бути досить уважними, адже кілька сотень мегабайт для ПК це ніщо, але для мобільного додатка може виявитися вкрай нераціональним рішенням.

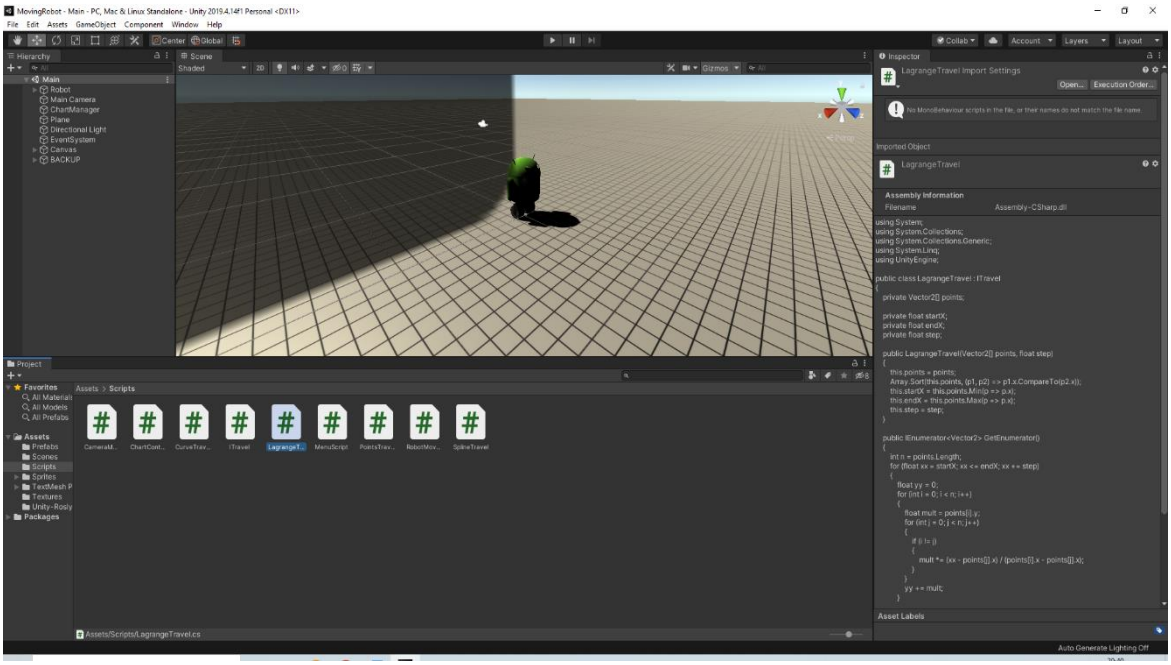


*Рис. 12 Приклад інтерфейсу Unity 3D*

### 1.6.3 Інтегроване середовище розробки Unreal Engine

Unreal Engine — один з відомих та популярних інтегрованих середовищ розробки для ігор. Головними характеристиками, що виокремлюють його серед інших аналогів є безмежні сфери застосування, свобода в творчості і визнання на ринку.

Можливості застосування Unreal Engine вже давно вийшли за рамки розробки комп'ютерних ігор . UE4 активно використовують для кіно, реклами, архітектурної візуалізації та тренувальних симуляцій. Секрет такого розмаїття в зручності, постійному розвитку та безкоштовності IDE.

Epic Games, творець Unreal Engine, активно відстежує розвиток інших компаній, підтримує їх та інтегрує їх рішення у свій глобальний продукт. Філософія Epic Games спрямована на подолання бар'єрів у творчості та цей підхід спостерігається одночасно в загальній політиці компанії, а, також, в інтуїтивності її продуктів.

Значною перевагою Unreal Engine є те, що навіть художник без попереднього досвіду програмування зможе розібратися з ним для реалізації своїх цілей. Більш того, навігація та налаштування багато в чому схожі з іншим програмним забезпеченням. Отже, якщо користувач має досвід роботи, наприклад, з Maya, то UE4 не справить враження чогось принципово нового, хоча і призначення у цих інструментів різне.

**Система Blueprint**

Blueprint — це система візуального скриптингу, що дозволяє реалізувати свій задум в спрощеному режимі. Наприклад, без знань програмування можна створити цикл дня і ночі або змусити оточення реагувати на переміщення гравця, що значно оптимізує роботу. Велика частина відомої гри Epic Games під назвою Fortnite реалізована саме через Blueprint.

Звичайно, це не означає, що під час роботи з UE4 більше не існує потреби залучення VFX-художника чи програмного інженера. Це лише свідчить про

те, що можливості звичайного художника стали ширшими та є можливість наглядно представити свою ідею. Попередня візуалізація за допомогою двигуна значно прискорює процес створення гри та роботу команди в цілому.

**Опрацювання та система часток**

Цінна перевага Unreal Engine серед інших двигунів полягає у швидкому створенні прототипу гри. У прототипі особливо важливо відразу передати атмосферность. Для цієї мети UE пропонує безліч інструментів - від симуляції природних явищ через систему часток до постпроцесингу.

Отже, таке явище як сніг можна швидко додати через Particle System. Для цього потрібно створити еммітер (компонент системи, що генерує частки) і налаштувати його під ваші потреби: пустити білі частки та задати параметри їх прольоту. А в той же час інструменти постобробки допоможуть надати зображенню цікавий вид. Для цього потрібно налаштувати значення відповідних параметрів: контрасту, колірного тону або аберацій.

У більшості випадків користувачу не доведеться шукати відеоуроків чи пояснень, оскільки всі параметри очевидно названі в інтерфейсі, і потрібний ефект можна підібрати інтуїтивно.

Вже зараз очевидно, що Unreal Engine лідирує в багатьох сферах - від креативних до наукових. Набуті навички роботи з даним двигуном допоможуть виділитися на ринку праці та ще довгий час залишатися актуальним фахівцем.

# РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ ВПЛИВУ НА РУХ НЕГОЛОНОМНОЇ КОЛІСНОЇ СИСТЕМИ

## 2.1 Задання руху неголономної моделі колісного робота

У підрозділі буде описано задання руху неголономної моделі колісного робота за програмною траєкторією для випадків, коли курсовий кут або кривизна програмної кривої задані у вигляді відомої функції від часу.

Система диференційних рівнянь, які визначають програмну траєкторію мають вид

$$\begin{cases} \dot{\psi} = \dot{\psi}(t) \\ \dot{x} = v \cdot \cos\psi \\ \dot{y} = v \cdot \sin\psi \end{cases} \qquad (1)$$

У цих рівняннях:

v — швидкість руху вздовж програмної кривої (передбачається постійної),

ψ — курсовий кут, швидкість зміни якого пов'язана з кривизною траєкторії співвідношенням

$$\dot{\psi}(t) = v \cdot Kr,$$

де $Kr$ — кривизна.

Закон зміни кута повороту колісного модуля визначається з умови рівності курсового кута куту нахилу дотичній до програмної кривої і курсового кута траєкторії заднього некерованого колеса, тоді

$$tg(\theta) = l / \rho = l \cdot Kr.$$

*Рис. 13 Прототип колісного робота*

*(3-х колісний Robot Tima від компанії Infocom Ltd)*

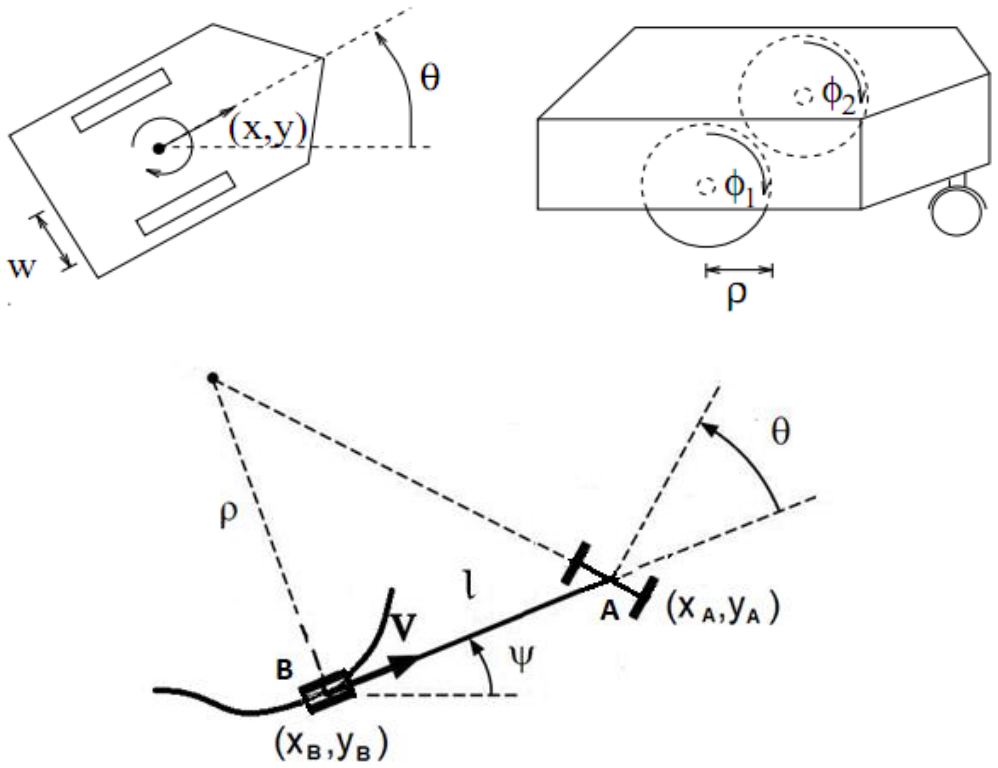



*Рис.14 Кінематична схема трьох-колісного робота*

У такому разі траєкторії характерних точок колісного робота будуть задаватися системою диференційних рівнянь:

$$\begin{aligned} \dot{x}_B &= v \cdot cos\big(\psi(t)\big) \\ \dot{y}_B &= v \cdot sin\big(\psi(t)\big) \\ \dot{\psi} &= v/\rho(t) \\ \dot{x}_A &= v \cdot cos\big(\psi_1(t)\big)/cos(\theta(t)) \\ \dot{y}_A &= v \cdot sin\big(\psi_1(t)\big)/cos(\theta(t)) \\ \dot{\psi}_1 &= v/\rho(t) + \dot{\theta}(t), \end{aligned} \qquad (2)$$

де $\psi_1 = \psi + \theta$ — курсовий кут переднього керованого колеса.

$\dot{\theta}(t)$ — швидкість зміни кута повороту керованого колісного модуля (відома функція часу).

Як приклад розглянемо випадок зміни курсового кута — «змійка».

$$\dot{\psi}(t) = v \cdot (0.5 + 1.5 \cdot sin(5t)).$$

Чисельне інтегрування системи диференційних рівнянь (1) — (2) у системі Maple дає змогу провести візуалізацію руху неголономної моделі колісного робота (Рис. 15)

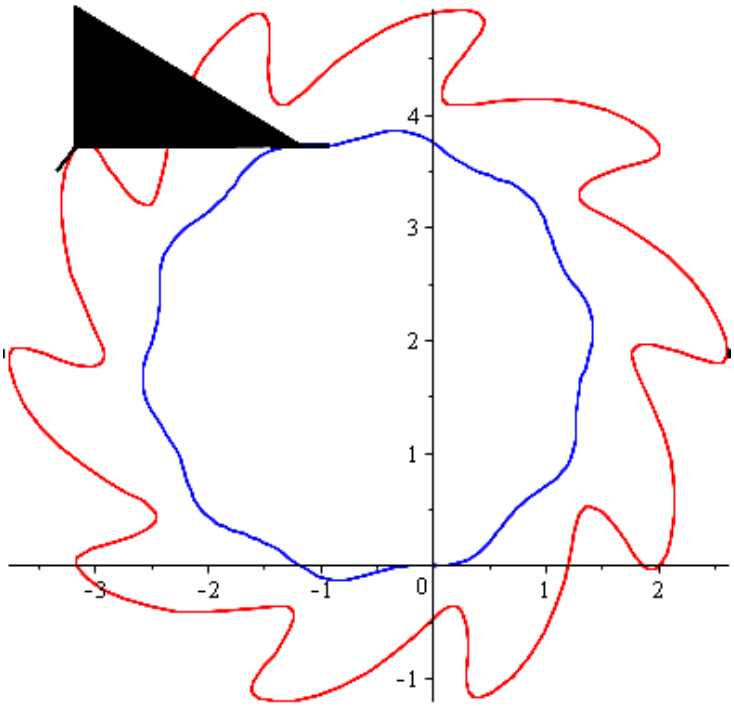



*Рис. 15 Візуалізація програмного руху робота*

Тут синім кольором визначена траєкторія заднього колеса, а червоним — траєкторія середньої точки між переднім та заднім колесами.

## 2.2 Рух неголономної моделі колісного робота

У підрозділі буде описано рух неголономної моделі колісного робота, коли програмна крива задана у полярній системі координат. У такому випадку можна перейти до параметричного задання програмної траєкторії точки B (точки контакту некерованого заднього колеса):

$$\boldsymbol{r} = \boldsymbol{r}(\varphi)$$

$$x_B = r(\varphi) \cdot \cos(\varphi);$$

$$y_B = r(\varphi) \cdot \sin(\varphi);$$

$$\dot{x}_B = \frac{\partial x_B}{\partial \varphi}\dot{\varphi}(t); \quad \dot{y}_B = \frac{\partial y_B}{\partial \varphi}\dot{\varphi}(t); \tag{3}$$

$$\dot{\psi} = v \cdot \frac{\frac{\partial x}{\partial \varphi}\frac{\partial^2 y}{\partial \varphi^2} - \frac{\partial y}{\partial \varphi}\frac{\partial^2 x}{\partial \varphi^2}}{((\frac{\partial x}{\partial \varphi})^2 + \left(\frac{\partial y}{\partial \varphi}\right)^2)^{\frac{3}{2}}}.$$

Реалізація у разі програмної кривої — «трипелюсткова троянда»:

$$x = \sin(3\varphi) \cdot cos(\varphi)$$

$$y = \sin(3\varphi) \cdot sin(\varphi)$$

$$\frac{\partial x}{\partial \varphi} = 3\cos(3\varphi) \cdot cos(\varphi) - \sin(3\varphi) \cdot sin(\varphi)$$

$$\frac{\partial y}{\partial \varphi} = 3\cos(3\varphi) \cdot sin(\varphi) + \sin(3\varphi) \cdot cos(\varphi)$$

$$\frac{\partial^2 x}{\partial \varphi^2} = -10 \cdot \sin(3\varphi) \cdot cos(\varphi) - 6 \cdot \cos(3\varphi) \cdot sin(\varphi)$$

$$\frac{\partial^2 y}{\partial \varphi^2} = -10 \cdot \sin(3\varphi) \cdot sin(\varphi) + 6 \cdot \cos(3\varphi) \cdot cos(\varphi)$$

$$\rho = \frac{((\frac{\partial x}{\partial \varphi})^2 + (\frac{\partial y}{\partial \varphi})^2)^{\frac{3}{2}}}{\frac{\partial x}{\partial \varphi}\frac{\partial^2 y}{\partial \varphi^2} - \frac{\partial y}{\partial \varphi}\frac{\partial^2 x}{\partial \varphi^2}}$$

$$\theta = \arctan(\frac{1}{\rho})$$

$\dot{\theta} = \frac{d\theta}{dt}$ — швидкість зміни кута управління θ.

Результати чисельного інтегрування відповідної системи диференційних рівнянь:

$$\dot{x}_B = v \cdot cos(\psi(t))$$

$$\dot{y}_B = v \cdot sin(\psi(t))$$

$$\dot{\psi} = v/\rho(t)$$

$$\dot{x}_A = v \cdot cos(\psi(t) + \theta(t))/cos(\theta(t)) \quad (4)$$

$$\dot{y}_A = v \cdot sin(\psi(t) + \theta(t))/cos(\theta(t))$$

$$\dot{\varphi}(t) = \frac{v}{((\frac{\partial x}{\partial \varphi})^2 + (\frac{\partial y}{\partial \varphi})^2)^{\frac{1}{2}}}$$

З початковими умовами:

$$x_A(0) = 1, y_A(0) = 0, x_B(0) = 0, y_B(0) = 0, \theta(0) = \theta_0, \varphi(0) = 0.$$

При *l=2м, v=1м/с* : $\theta_0 = \theta|_{t=0}$= 0.197395598 *rad.*

Нижче лістинг програми у системі Maple:

*Лістинг 3 чисельне інтегрування системи диференційних рівнянь*

```
> x:=sin(3*phi)*cos(phi);
> y:=sin(3*phi)*sin(phi);
> Xphi:=diff(x,phi);
> Yphi:=diff(y,phi);
> XXphi:=diff(Xphi,phi);
> YYphi:=diff(Yphi,phi);
> Xphi:=subs(phi=phi(t),Xphi);
> Yphi:=subs(phi=phi(t),Yphi);
> XXphi:=subs(phi=phi(t),XXphi);
> YYphi:=subs(phi=phi(t),YYphi);
> ro:=(Xphi^2+Yphi^2)^(3/2)/(YYphi*Xphi-XXphi*Yphi):
> l:=2;v:=1;
> thet:=arctan(l/ro):
> Dtheta - швидкість зміни кута управління theta, але у
программы зустрічається і thet (ідентичні за сенсом, була
необхідність ввести дві функціонально різні залежності):
> Dtheta:=diff(thet,t):
> Dtheta:=subs(diff(phi(t),
t)=v/(Xphi^2+Yphi^2)^(1/2),Dtheta):
> theta0:=evalf(subs(phi(t)=0,thet));
> with(plots):
> F:=dsolve({diff(xB(t),t)=v*cos(psi(t)),
```

```
diff(yB(t),t)=v*sin(psi(t)),
diff(psi(t),t)=v/ro,
diff(xA(t),t)=v*cos(psi(t)+theta(t))/cos(theta(t)),
diff(yA(t),t)=v*sin(psi(t)+theta(t))/cos(theta(t)),
diff(thta(t),t)=Dtheta,
diff(phi(t),t)=v/(Xphi^2+Yphi^2)^(1/2,xB(0)=0,yB(0)=0,psi(0
)=0,
xA(0)=l,yA(0)=0,theta(0)=theta0,phi(0)=0},
[xB(t),yB(t),psi(t),xA(t),yA(t),theta(t),phi(t)],
numeric,output=listprocedure;
```

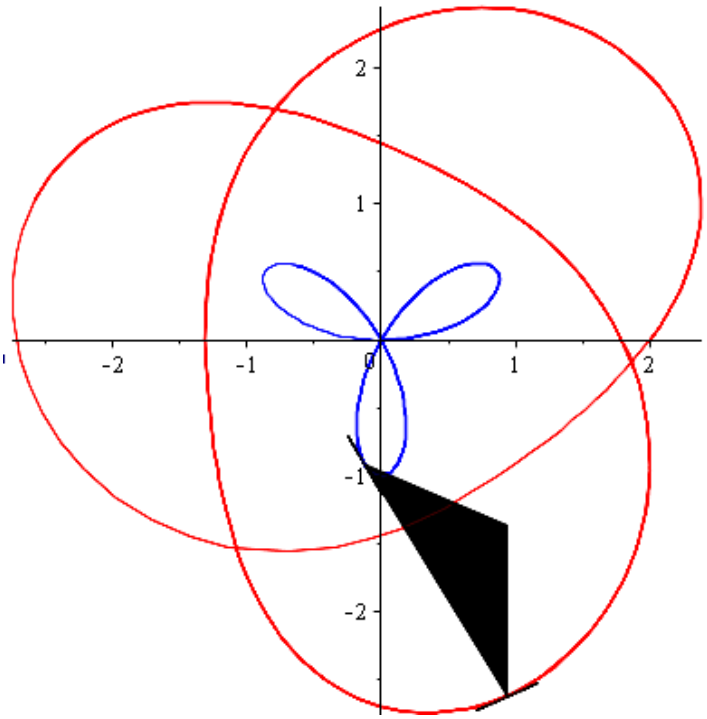



*Рис. 16 Візуалізація програмного руху у полярній системі координат*

Синім кольором визначена траєкторія заднього колеса, а червоним — траєкторія середньої точки між переднім та заднім колесами.

Функція управління ($\theta(t)$ — кут повороту керованих коліс) представлена у вигляді фазової траєкторії на рисунку 17.

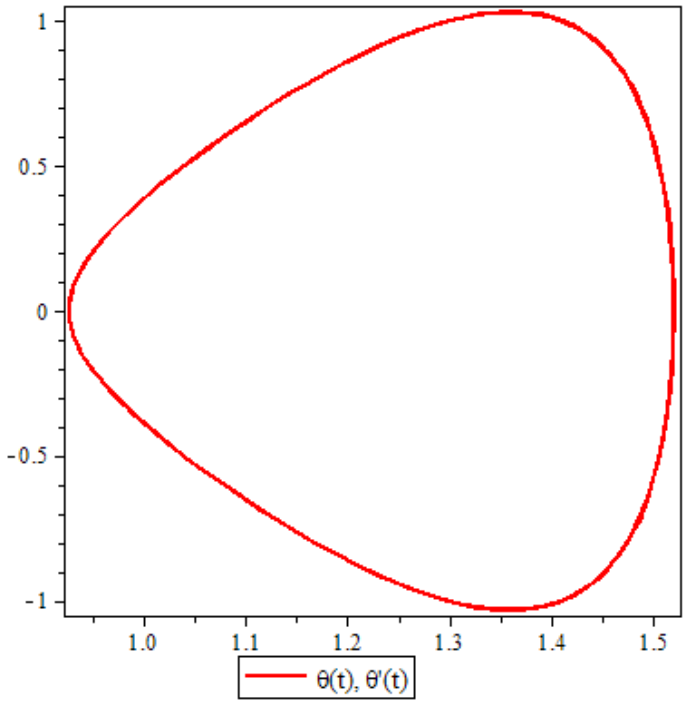



*Рис. 17 Фазова траєкторія — кут повороту та його кутова швидкість*

У розглянутих вище прикладах передбачалося, що кут повороту передніх керованих коліс (або колеса) — $\theta(t)$ у кожний момент часу задається відомою функцією від часу (arctan($l$/ρ), а швидкість заднього колеса (колес) (точки B) — константа, що відповідає велосипедній схемі робота з активним керованим (направляючим) колесом.

## 2.3 Трьох-колісний робот з пасивною центральною опорою

Для моделі триколісного робота з пасивної центральною опорою (роя-льним підвісом) рух уздовж програмної траєкторії може забезпечуватися за рахунок керування кутовою швидкістю власного обертання ведучих коліс.

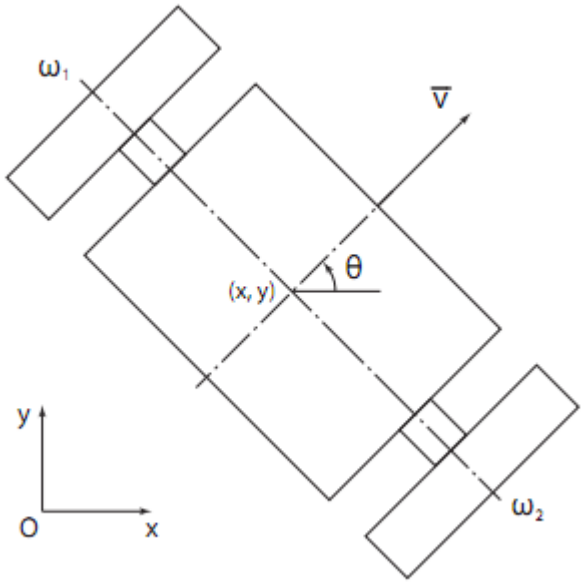



*Рис. 18 Колісний робот з пасивною опорою*

$$\omega = \frac{v_B}{\rho} = \frac{v_2 - v_1}{Kol}; \qquad v_B = \frac{v_2 + v_1}{2};$$

Швидкості центрів коліс пов'язані з кутовими швидкостями власного обертання навколо осі $B_1B_2$ та радіусом ведучих коліс $R$:

$$\omega_1 = \frac{v_1}{R}; \quad \omega_2 = \frac{v_2}{R}.$$

Кутова швидкість при обертальному русі навколо вертикальної осі (швидкість зміни курсового кута) визначається рівністю:

$$\omega = \dot{\psi},$$

де, $\psi$ - курсовий кут, утворений поздовжньою віссю симетрії робота.

Враховуючи визначення кривизни траєкторії, для точки В маємо:

$$\dot{\psi} = v_B/\rho(t).$$

З іншої сторони, з умови відсутності поздовжнього прослизання ведучих коліс маємо співвідношення для швидкостей центрів ведучих коліс

$$v_B/\rho(t) = \frac{(\omega_2 - \omega_1)R}{Kol}, \qquad v_B/R = \frac{\omega_2 + \omega_1}{2}.$$

Звідки

$$\omega_1 = \frac{v_B}{R} - \frac{v_B \cdot Kol}{2R \cdot \rho(t)}; \qquad \omega_2 = \frac{v_B}{R} + \frac{v_B \cdot Kol}{2R \cdot \rho(t)},$$

або для лінійних швидкостей центрів ведучих коліс

$$v_1 = v_B - \frac{\omega \cdot Kol}{2}; \qquad v_2 = v_B + \frac{\omega \cdot Kol}{2}.$$

Розглянемо реалізацію руху робота за складною замкнутою траєкторією:

$$\rho = \frac{1}{0.5 + 1.5 \cdot \sin(5t)}$$

З такими даними:

*m= 13.5 кг; J=0.35 кг·м$^2$; l=0.36 м; a=0.133333 м; b=0.226666 м; Kol=0.25 м;*

Тоді швидкості ведучих коліс дорівнюють:

$$v_1 = v - \frac{v \cdot Kol}{2\rho}; \qquad v_2 = v + \frac{v \cdot Kol}{2\rho}.$$

Система диференційних рівнянь:

$$\dot{x}_{B_1} = v_1 \cdot cos(\psi(t));$$
$$\dot{y}_{B_1} = v_1 \cdot sin(\psi(t));$$
$$\dot{x}_{B_2} = v_2 \cdot cos(\psi(t));$$
$$\dot{y}_{B_2} = v_2 \cdot sin(\psi(t));$$
$$\dot{\psi} = v/\rho(t);$$

Початкові умови:

$x_{B_1}(0) = 0,\ y_{B_1}(0) = \mathbf{0.125},\ x_{B_2}(0) = 0,\ y_{B_2}(0) = \mathbf{0.125},\ \psi(0) = 0.$

Нижче лістинг програми у системі Maple:

*Лістинг 4*

```
> v:=1;
> l:=0.36;Kol:=0.25;
> ro:=2;
> ro:=1/(0.5+1.5*sin(5*t));
> v1:=v-v/ro*Kol/2; v2:=v+v/ro*Kol/2;
>F:=dsolve({diff(xB1(t),t)=v1*cos(psi(t)),
diff(yB1(t),t)=v1*sin(psi(t)),
diff(psi(t),t)=v/ro,
diff(xB2(t),t)=v2*cos(psi(t)),
diff(yB2(t),t)=v2*sin(psi(t)),
xB1(0)=0,yB1(0)=0.25/2,psi(0)=0,xB2(0)=0,yB2(0)=-0.25/2},
[xB1(t),yB1(t),psi(t),xB2(t),yB2(t)],
numeric,abserr=0.1e-8,output=listprocedure);
> XB1:=subs(F,xB1(t)):
> YB1:=subs(F,yB1(t));
> XB2:=subs(F,xB2(t)):
> YB2:=subs(F,yB2(t));
> List:=[ ];
> for i from 0 by 1 to N-1 do
List:=[op(List),[[rhs(F[2](1.25*i)),rhs(F[3](1.25*i))],[rhs
(F[5](1.25*i)),rhs(F[6](1.25*i))]]] end do:
> TB1:=plot([XB1,YB1,t0..tN],style=LINE,linestyle=SOLID,
color=RED,scaling=constrained):
> B2:=plot([XB2,YB2,t0..tN],style=LINE,linestyle=SOLID,
color=RED,scaling=constrained):
> L:=plot(List,style=LINE,color=BLUE,scaling=constrained):
> display(TB1,TB2,L);
```

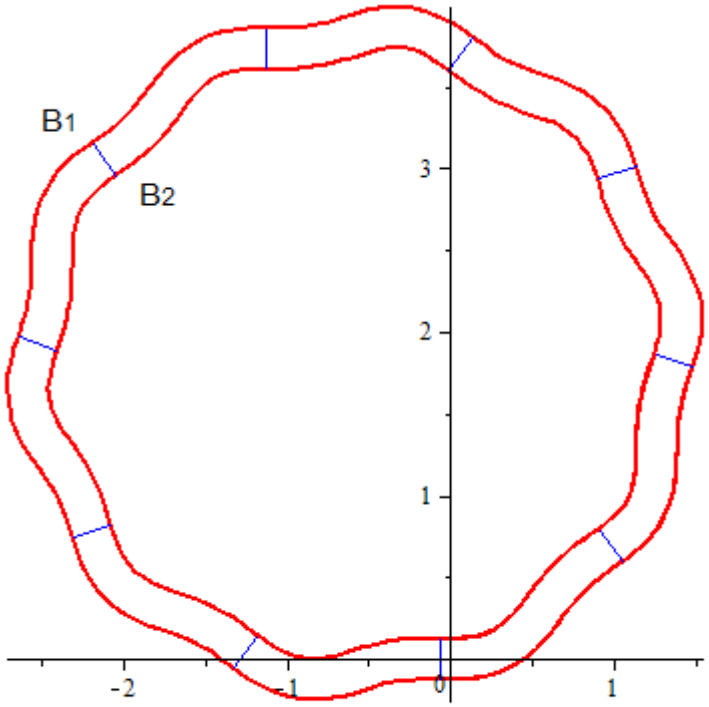



*Рис. 19 Траєкторія руху ведучих коліс*

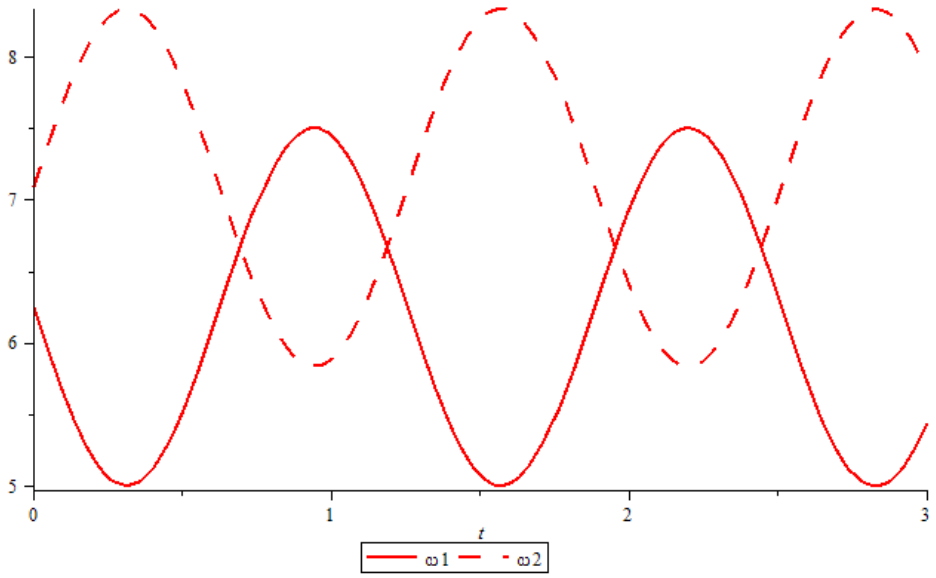



*Рис. 20 Синтез управління кутовими швидкостями ведучих коліс*

Для практичної реалізації руху колісного робота за програмними траєкторіями необхідно також використовувати рівняння динаміки неголономних систем [1, 9–14].

# РОЗДІЛ 3 ПРОЕКТ ПРОГРАМНОЇ СИСТЕМИ КЕРУВАННЯ ТА СТАБІЛІЗАЦІЇ РУХУ КОЛІСНИХ РОБОТІВ

## 3.1 Архітектура системи

### 3.1.1 Загальна архітектура системи

Система поділяється на дві частини:

1. Система керування колісним роботом
2. Система моделювання руху колісного робота

У архітектурі системи керування колісним роботом виділено 3 основні підсистеми (див. Рис. 21). Перша підсистема представляє безпосередньо саму систему колісного робота, яка складається з мобільного застосунку, написаному на мовах програмування Java та Kotlin, та консольного застосунку, написаному на мові програмування C++, який відповідає за різні обчислювання, розпізнавання образів та керує електричним двигуном. Друга підсистема — це сервер застосунків, який керує з’єднаннями роботів та користувачів, надає доступ до керування роботами користувачам, зберігає дані та виконує авторизацію користувачів. Та третя підсистема, яка відповідає за користувацький інтерфейс, взаємодіє з сервером застосунків для віддаленого керування колісним роботом, шляхом передачі команд управління, які інтерпретуються роботом та виконуються.

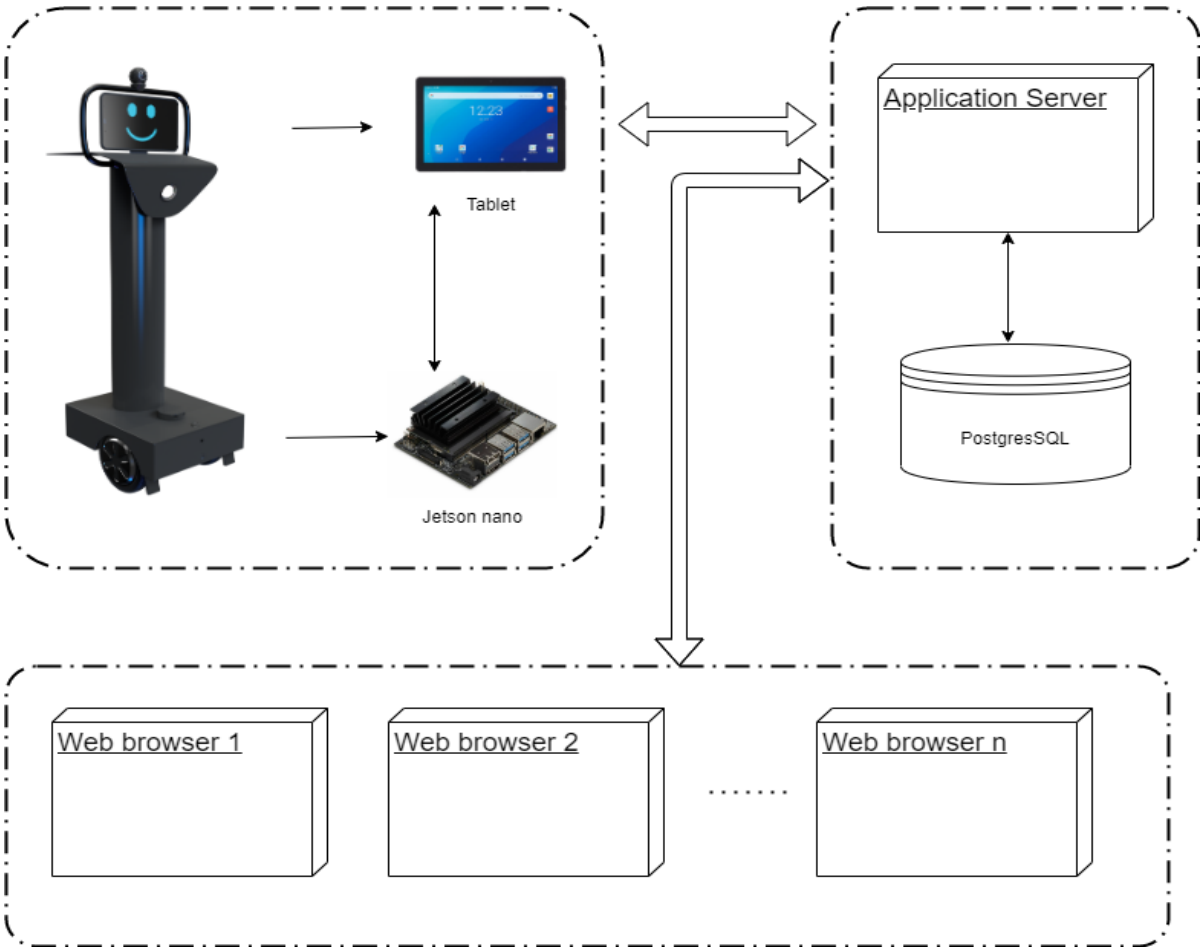



*Рис. 21 Загальна архітектура системи керування колісним роботом*

Архітектура системи моделювання руху та стабілізації колісного робота представлена взаємодіючими модулями, які складаються з набору 3D-сцен, 3D-об’єктів та керуючих скриптів, написаних на C# для Unity 3D. головним об’єктом є макет колісного робота, для якого реалізовано різні алгоритми руху по сцені.

У наступних розділах розглянута кожна система окремо.

### 3.1.2 Архітектура серверу застосунків

Архітектура сервера застосунків побудована за патерном «модель–вигляд–контролер» (model-view-controller, MVC). MVC — це архітектурний шаблон, який використовують під час проектування та розробки програмного забезпечення з метою розділити систему на три частини, що взаємодіють між

собою. Перша частина — це модель (Model). Модель є центральним компонентом патерну MVC, і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель відповідає за пряме керування даними, логікою та бізнес правилами застосунку. Друга частина — вигляд (View). Вигляд може являти собою будь-яке представлення інформації, одержану на виході, наприклад html-сторінку, графік чи діаграму. Третя частина — контролер (Controller). Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду.

Більш широко мету патерну можна сформулювати так: головна мета — гнучка архітектура програмного забезпечення, яка сприяє полегшенню подальшої модифікації програми, а також надавати можливість повторно використовувати компоненти програми. Крім того, використання цього патерна у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

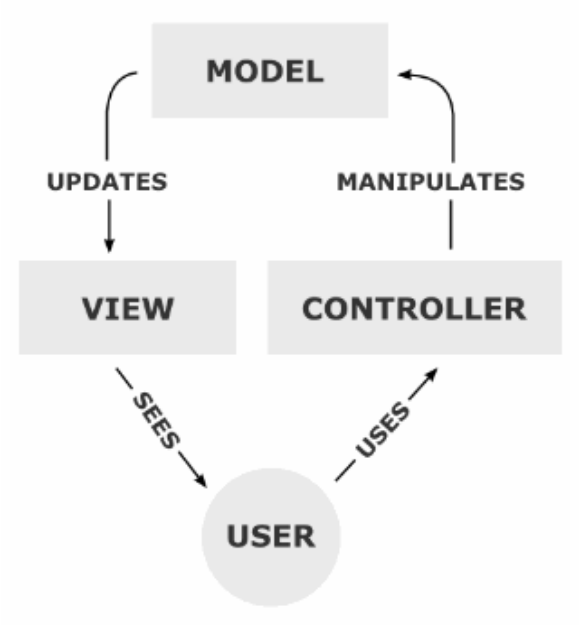



*Рис. 22 Структура патерна MVC*

Також додано додаткові директорії для розділу програмних компонентів за сенсом для покращення розуміння структури (Рис. 23).

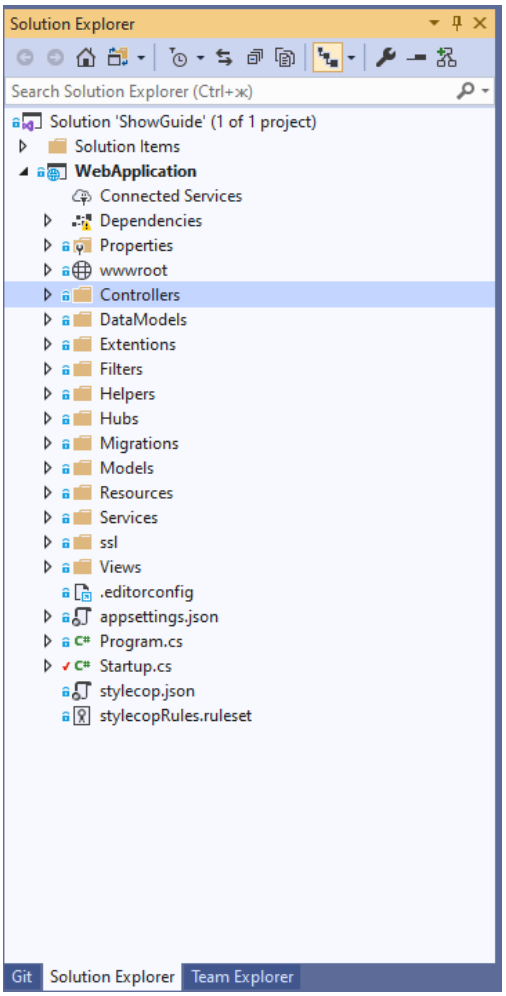



*Рис. 23 Файлова структура проекту серверу застосунків*

**Реалізація патерна**

Для реалізації патерну MVC було використано інтегроване середовище розробки Visual Studio 2019, яке надає можливість створити веб-проект з вбудованим каркасом для MVC, яке включає файлову структуру проекту, спеціальну систему маршрутизації та спеціальні класи для реалізації патерну. Прикладом базових класів є:

1. Microsoft.AspNetCore.Mvc.ControllerBase.
2. Microsoft.AspNetCore.Mvc.Controller.
3. Microsoft.AspNetCore.Mvc.IActionResult.
4. Microsoft.AspNetCore.Mvc.Filters.

### 3.1.3 Архітектура системи колісного робота

Система складається з двох підсистем: мобільний застосунок, який виконує роль зовнішнього інтерфейсу для взаємодії з користувачем, є точкою для взаємодії з сервером застосунків та користувачем шляхом надання доступу до мережі інтернет через вбудований 3G-модем, та консольний застосунок, який керує електродвигуном, відповідає за розпізнавання образів та виконує команди, які надіслані користувачем та передані через мобільний застосунок. Взаємодія системи колісного робота з сервером застосунків та клієнтом показана на Рис. 24.

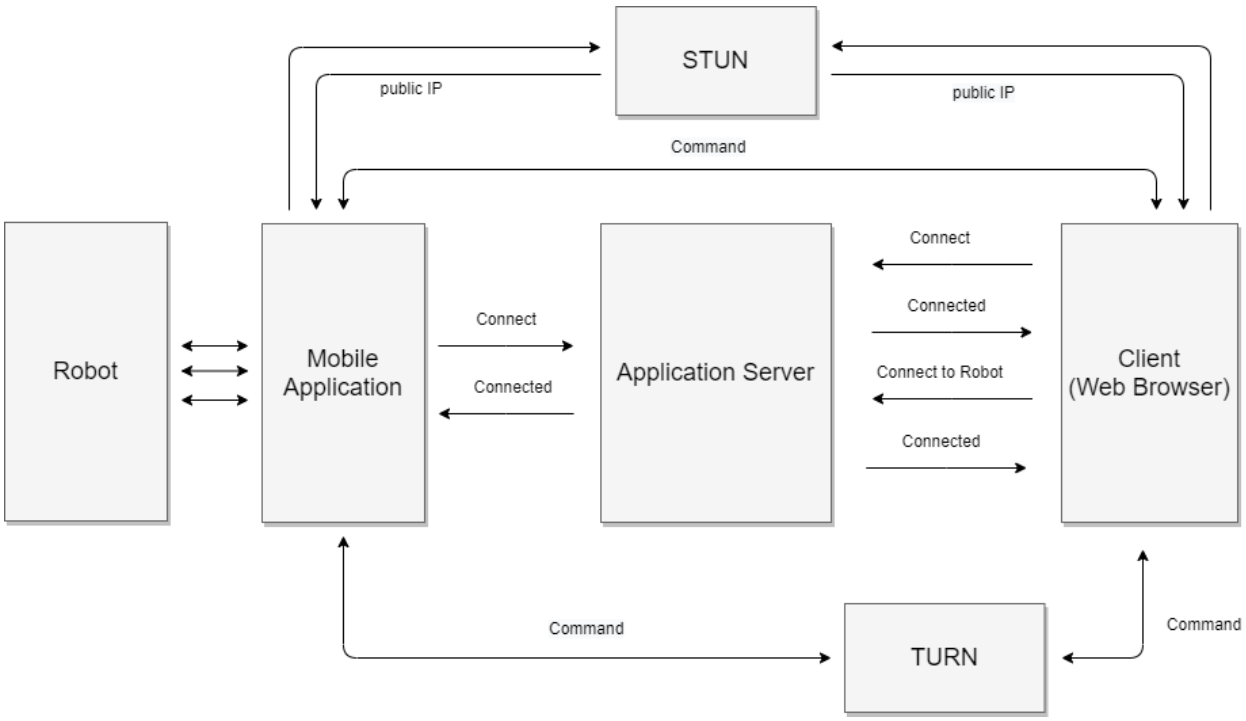



*Рис. 24 Діаграма взаємодії системи колісного робота з сервером застосунків та клієнтом*

Мобільний застосунок спроєктований за патерном Model-View-ViewModel. Model-View-ViewModel (MVVM) — це шаблон проєктування, що застосовується під час проєктування архітектури застосунків ціль якого полегшити відокремлення розробки графічного інтерфейсу від розробки бізнес логіки, представленої у патерні, як «модель». Необхідністю цього є на-

дання можливості змінювати бізнес логіку та графічний інтерфейс незалежно один від одного. Наприклад, розробник працює над логікою роботи з даними, а дизайнер — з користувацьким інтерфейсом. З назви патерну зрозуміло, що він складається з трьох частин:

1. Моделі (Model), яка представляє фундаментальні дані, які необхідні для роботи застосунку.
2. Вигляд (View), який представляє безпосередньо графічний інтерфейс.
3. Модель вигляду (ViewModel), яка з одного боку є абстракцією вигляду, а з іншого є обгорткою даних моделі, які мають бути зв’язаними. Тобто модель вигляду містить модель, яка перетворена до вигляду, а також містить у собі команди, якими може скористатися вигляд для впливу на модель.

ViewModel є частиною, яка відповідає за перетворення даних для їх подальшої підтримки і використання. З цієї точки зору, ViewModel більше схожа на модель, ніж на представлення і оброблює більшість, якщо не всю, логіку відображення даних. Фактично ViewModel призначена для того, щоб

1. Здійснювати зв'язок між моделлю та виглядом.
2. Відслідковувати зміни в даних, що зроблені користувачем.
3. Відпрацьовувати логіку роботи вигляду.

Схема взаємодіє компонентів патерну MVVM зображено на Рис. 25.

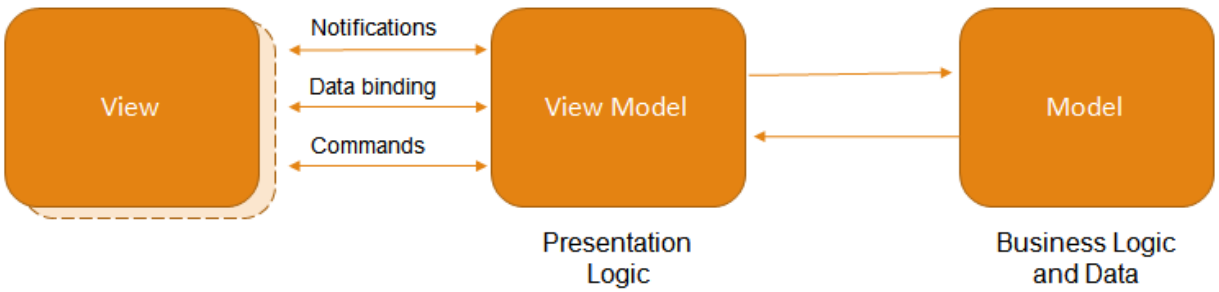



*Рис. 25 Схема взаємодіє компонентів патерну MVVM*

### 3.1.4 Архітектура системи моделювання руху колісного робота

Архітектура системи моделювання руху безпосередньо залежить від інтегрованого середовища розробки Unity. 3D-проєкт складається з набору сцен, 3D-об’єктів та скриптів, які задають поведінку об’єктам та описують взаємодію між об’єктами. Для реалізації варіативності руху робота було обрано патерн стратегія об’єднаний з патерном ітератор. Стратегія — патерн проєктування, який належить до класу патернів поведінки. Головною метою патерну є визначання класу алгоритмів, їх інкапсуляції та забезпечення взаємозамінності. Перекладаючи на систему патерн виділяє алгоритми руху робота об’єднуючи їх інтерфейсом ITravel. Таким чином кожен алгоритм реалізує інтерфейс ITravel, що дає змогу абстрагуватися від конкретного алгоритму та забезпечити взаємозамінність алгоритмів. Діаграма патерну стратегія представлена на Рис. 26.

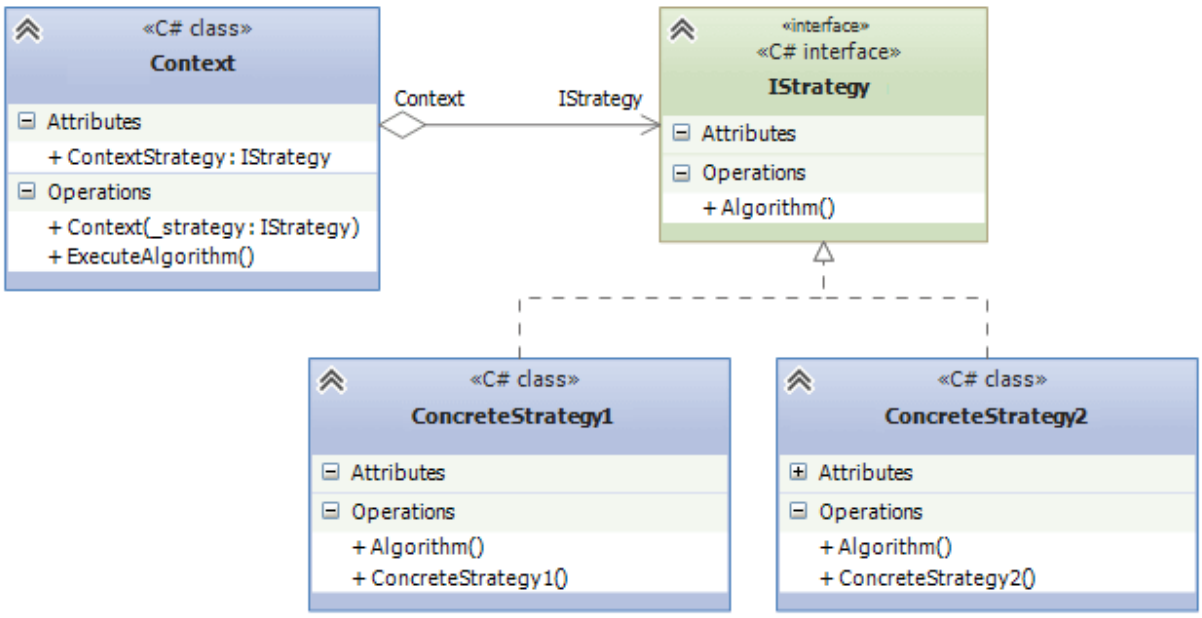



*Рис. 26 Діаграма патерна стратегії*

Отже, з використанням патерну стратегія система є гнучкою до вибору алгоритму руху та легко може замінити один алгоритм на інший. Головною метою кожного алгоритму руху по суті є отримання наступної точки маршруту шляхом обчислень. Для зручного використання кожного з алгоритмів був використаний патерн ітератор. Патерн ітератор є поведінковим патерном

проєктування. По суті він представляє об’єкт, який дозволяє отримати послідовний доступ до елементів об’єкту агрегату. Реалізований за допомоги стандартного інтерфейсу з простору System.Collections — IEnumerable. На практиці поєднання цих двох патернів дає змогу незалежно від алгоритму руху, отримати його об’єкт ітератор та реалізувати рух, як послідовний перебір точок. Де безперервний рух буде означати досягання кожної з точок по черзі поки об’єкт ітератора не поверне останню. Зупинка та відновлення руху також буде простою у реалізації, так як ітератор пам’ятає поточну точку.

## 3.2 Засоби реалізації

### 3.2.1 Microsoft Visual Studio 2019

Microsoft Visual Studio — це інтегроване середовище розробки від компанії Microsoft. Продукт включає в себе інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Ці інструменти допомагають розробляти як консольні застосунки, так і застосункиз з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms та WPF. Також середовище підтримує веб-застосунки та веб-служби. Серед багатьох сильних сторін, треба відзначити підтримку багатьох мов програмування, таких як C#, F#, Visual Basic, C++, Java Script, Python та інші. Також слід відзначити зручний інтерфейс, приклад якого показано на рисунку 27.

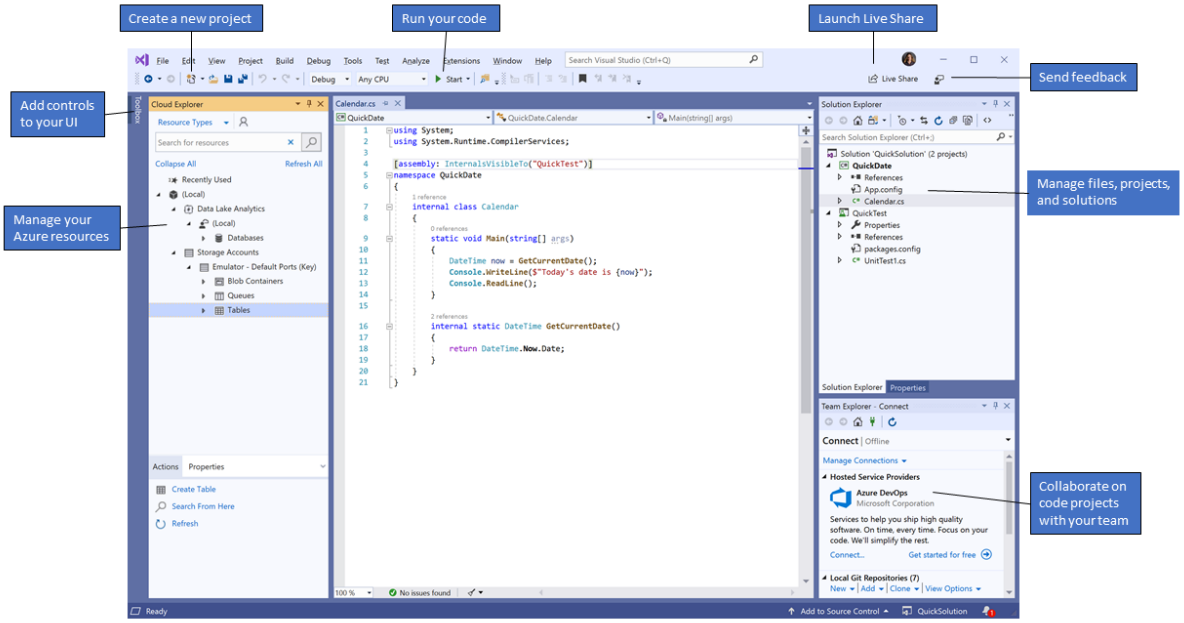



*Рис. 27 Інтерфейс Microsoft Visual Studio*

### 3.2.2 Редактор коду Visual Studio Code

Visual Studio Code — це редактор програмного коду від компанії Microsoft. Visual Studio Code розроблена для операційних систем Windows, Linux та MacOS. Позиціонує себе, як легкий редактор програмного коду для крос-платформної розробки веб та хмарних застосунків. Також Visual Studio Code підтримує безліч мов програмування.

Можна виділити декілька вагомих переваг Visual Studio Code:

1. Сніпети.
2. IntelliSense.
3. Інтегрований термінал.
4. Інструмент перегляду використання сутностей, їх визначення та рефакторинг.

Використання сніпетів — це дуже ефективний спосіб покращення продуктивності програміста. Головна ціль сніпетів — автоматизувати ручну працю. Сніпети дають доступ до каркасних фрагментів коду, які часто потрі-

бні програмісту. Отже використовуючи сніпети програміст більше думає про те, як конструкції коду взаємодіють між собою а не про те, як їх написати.

IntelliSense — це система автозавершення коду. Visual Studio Code дійсно має розумну систему, яка допомагає покращити продуктивність та швидкість розробки програмного забезпечення. Наприклад якщо навести на деяку частину коду, IntelliSense покаже додаткові відомості про типи даних та дасть змогу швидко потрапити до декларування цих даних. Це особливо корисно, коли ви працюєте з об’єктами, які були створені іншими програмістами.

При роботі з великим проектом деякий клас, метод або властивість можуть бути використані у багатьох місцях. Visual Studio Code дозволяє дізнатися де саме використовуються такі сутності, швидко та ефективно зібрав повну інформацію про них.

Наявність інтегрованого терміналу може допомогти в економії часу за рахунок того, що програмісту не треба постійно переключатися між вікнами терміналу та редактору коду. Також це дуже зручно при автоматизованому старті проекту, де ви можете дуже легко помітити помилки.

Редактор включає у себе інструменти відлагодження коду, інструменти для роботи з системами контролю версій, наприклад Git та інструменти рефакторінгу.

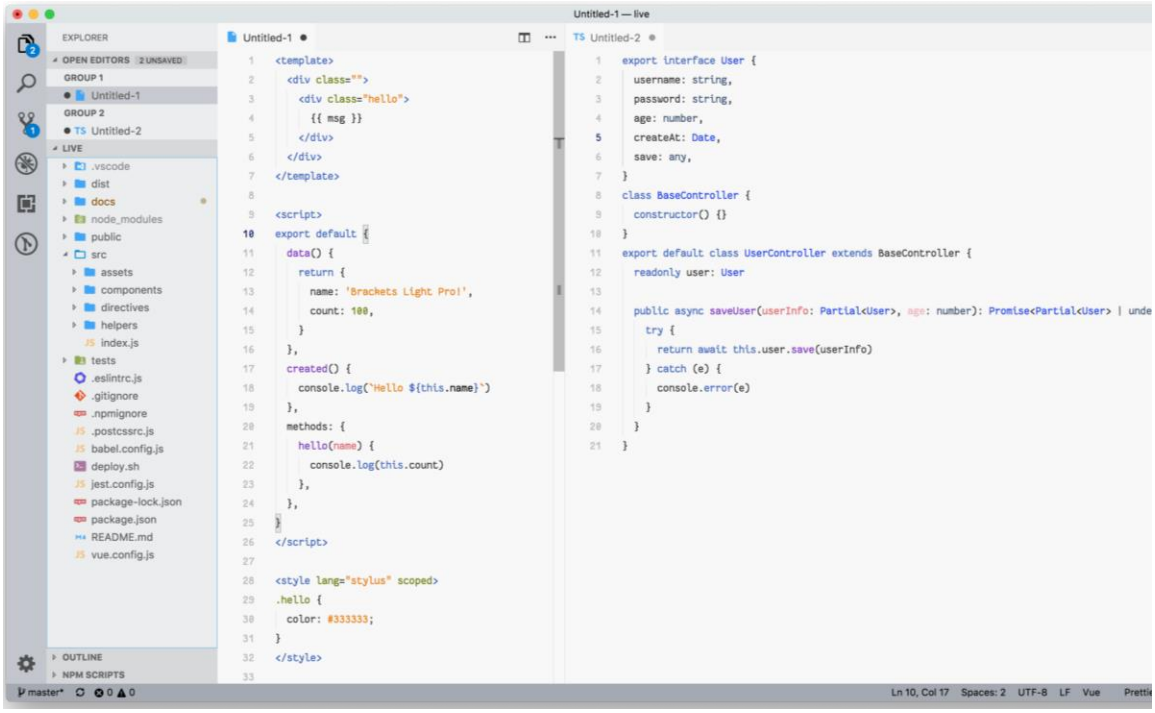



*Рис. 28 Інтерфейс Visual Studio Code*

### 3.2.3 IntelliJ IDEA

Під час розробки  мобільного за стосунку було використано інтегроване середовище розробки IntelliJ IDEA. IntelliJ надає багато можливостей розробнику, який пише програмне забезпечення на Java або Kotlin. Середовище має багато пакетних менеджерів, таких, як Maven, Ant та Gradle. Середовище має багато утиліт, які спрощують процес написання коду, та зручно інтегрує у себе інші плагіни, які можуть спрощувати розробку структур різних баз даних, підключати сторонні сервери для розгортання застосунків, такі як Tomcat, Jetty та інші. Однією з особливостей, які допомогли мені прискорити розробку мобільного застосунку — це можливість транскомпіляції програмного коду Java у програмний код Kotlin. Також середовище може запропонувати вам покращити свій код, шляхом детального аналізу. Аналізатор IntelliJ знаходить код, який може буди оптимізований, може запропонувати відділити однакові частини коду в один програмний модуль, метод або функцію тощо. IntelliJ поділяється на IntelliJ IDEA Ultimate з повним набором всіх моду-

лів та функцій та її обмеженим аналогом IntelliJ IDEA Community, який є безкоштовним для студентів або для некомерційного використання.

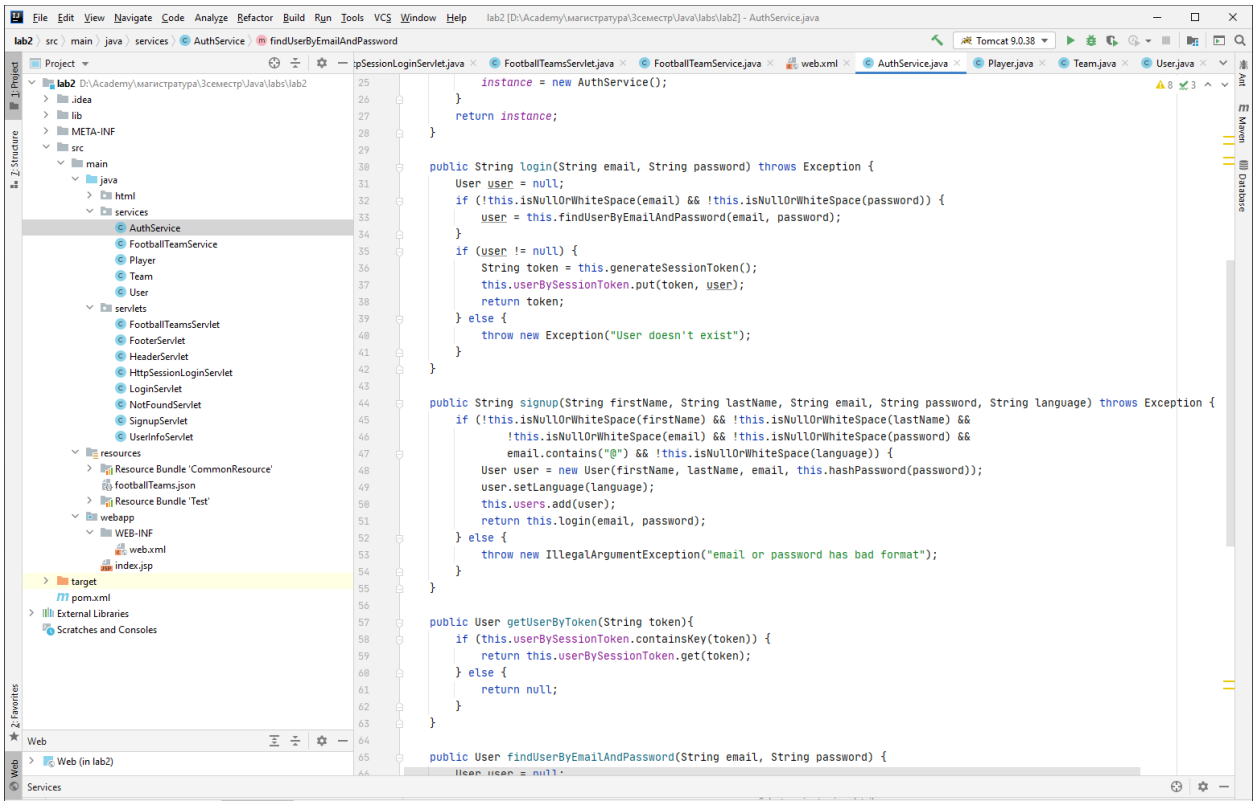



*Рис. 29 Інтерфейс IntelliJ IDEA*

### 3.2.4 Мова програмування Java Script

Java Script — це крос-платформна, скриптова мова програмування. Java Script була спроектована не для створення закінчених, повноцінних застосунків, а скоріше для легкого інтегрування у інші програмні продукти, такі як браузери. Але з часом мова набула розвитку і стала незалежною. Таким чином на 2020 рік за допомоги Java Script можливо створювати серверні застосунки, веб-застосунки та навіть віконні застосунки. Також слід відзначити, що Java Script — це мова програмування, що інтерпретується а не компілюється, тому для її виконання потрібен інтерпретатор. Такі програми часто перетворюються у «двигуни» або «ядра», прикладом таких програм для Java Script є двигун V8 для Google Chrome та Opera та SpiderMonkey для FireFox.

Насправді JavaScript є потужною мовою програмування, яка очолює ринок веб-програмування. Середовище розробки для JavaScript є у кожному комп'ютері та смартфоні, що значною мірою посприяло у зростанні популярності JavaScript. Але мова має і свої недоліки, приклад основних недоліків:

1. Мова компілюється під час виконання.
2. У мові відсутня типізація даних. Це є проблемою усіх скриптових мов. Поки виконання програми не дійде до конкретного фрагменту коду, неможливо дізнатися чи він працює.
3. Доволі не звична об'єктна модель, яка має класи та їх наслідування, але значно відрізняється від звичної реалізації в таких мовах, як C++, Java та C#.

### 3.2.5 Мова програмування C++

C++ — це мова програмування, яка є розширенням мови C. Мова C є потужною і гнучкою мовою програмування, що використовувалася для розробки найбільш важливих програмних продуктів протягом минулих років. Проте, як тільки проект перевищує певні розміри, можливості вживання мови C досягають своїх кордонів. Залежно від проекту, програми розміром від 25000 до 100000 рядків виявляються важкими для розробки і управління тому, що їх важко охопити повністю.

Працюючи в Bell Laboratories в Murray Hill, штат Нью-Джерсі, Б'ярн Страуструп (Bjarne Stroustrup) додав до мови C декілька розширень з метою вирішити цю проблему. Спочатку мова називалася «C з класами». Ця назва була замінена на C++ в 1983 році. Більшість модифікацій зроблених Страуструпом з мовою C підтримують об'єктно-орієнтоване програмування, яке інколи скорочено називають ООП.

Хоча спочатку C++ був націлений на роботу з дуже великими програмами, це не обмежує його вживання. Фактично об'єктно-орієнтовані атрибути мови C++ можуть бути ефективно застосовані фактично до будь-якої за-

дачі програмування. Ця мова часто використовується для таких проектів, як створення редакторів, баз даних, персональних систем роботи з файлами і комунікаційних програм. Завдяки тому, що C++ успадкувала ефективність мови C, з її допомогою розробляється високопродуктивне програмне забезпечення.

Можна виділити декілька основних аспектів мови C++:

1. Мова розроблена як універсальна зі статичними типами даних, високою ефективністю та лаконічністю.
2. Мова підтримує багато різних стилів програмування, таких, як процедурне програмування, абстракція даних, об’єктно-орієнтоване програмування та узагальнене програмування.
3. Має повну сумісність з мовою програмування C.
4. Не вимагає складного середовища програмування.
5. Концепцією є надання програмісту свободу вибору, навіть коли цей вибір неправильний.
6. Намагається уникати платформо-залежних властивостей.

Серед недоліків можна виділити:

1. Недолік інформації про типи під час компіляції.
2. Мова є складною для вивчення.
3. Препроцесор C++ є доволі примітивним.
4. Має погану підтримку модульності.
5. Мова є складною у відлагоджені, через низку можливостей, які суперечать принципам безпечності типів, тому часто візитною карткою мови C++ є ураження типу «переповнення буферу».

### 3.2.6 Фреймворк для створення веб-застосунків ASP.NET Core 3.1

Платформа ASP.NET Core представляє технологію від компанії Microsoft, призначену для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів. З одного боку,

ASP.NET Core є продовженням розвитку платформи ASP.NET, але з іншого боку, це не просто черговий реліз, а самостійне крос-платформне відгалуження. ASP.NET Core може працювати поверх крос-платформного середовища .NET Core, яке може бути розгорнуте на основних популярних операційних системах: Windows, Mac OS, Linux. І таким чином, за допомогою ASP.NET Core ми можемо створювати крос-платформні веб-додатки. ASP.NET Core включає в себе фреймворк MVC, який об'єднує функціональність MVC, Web API і Web Pages.

ASP.NET Core характеризується масштабованістю. Фреймворк побудований з набору незалежних компонентів, що дає можливість використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої компоненти зі своїм функціоналом.

Також було спрощено управління залежностями і конфігурація проекту. Фреймворк тепер має свій легкий контейнер для впровадження залежностей, і більше немає необхідності застосовувати сторонні контейнери, такі як Autofac, Ninject. Хоча при бажанні їх також можна продовжувати використовувати.

### 3.2.7 ORM фреймворк Entity Framework Core

Entity Framework Core — це нова версія Entity Framework після EF 6.X. Це полегшена, крос-платформна версія технології доступу до даних Entity Framework з відкритим вихідним кодом [25].

Entity Framework — це структура об'єктно-реляційного зіставлення (ORM). Це удосконалення ADO.NET, яке надає розробникам автоматизований механізм для доступу і зберігання даних в базі даних. EF Core призначений для використання з додатками .NET Core. Однак його також можна використовувати зі стандартними додатками на базі .NET 4.5+. На рисунку 30 показано підтримувані типи додатків, .NET Framework та ОС.

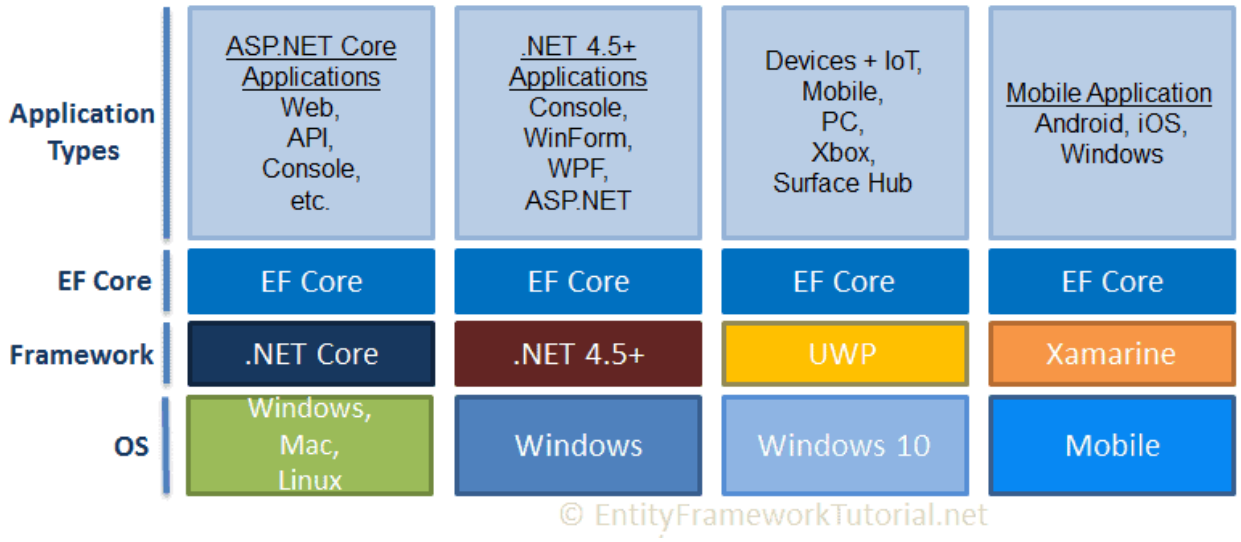



*Рис. 30 Підтримувані типи додатків EF Core*

**Міграції**

Міграція — це спосіб зберегти синхронізацію схеми бази даних з моделлю EF Core шляхом збереження даних. Згідно з рисунком 31, EF Core API будує модель EF Core з класів домена (сутностей), а міграції EF Core створюватимуть або оновлюватимуть схему бази даних на основі моделі EF Core. Кажен раз, коли змінюються класи домена, треба запускати міграцію, щоб підтримувати схему бази даних в актуальному стані.


*Рис. 31 Схема виконання міграції EF Core*

### 3.2.8 Бібліотека для асинхронної комунікації SignalR

ASP.NET SignalR — це бібліотека для розробників ASP.NET, яка спрощує процес додавання веб-функцій в реальному часі в додатки [22]. Веб-функціональність в реальному часі - це можливість змусити серверний код

відправляти контент підключеним клієнтам миттєво, коли він стає доступним, замість того, щоб сервер чекав, поки клієнт запросить нові дані.

SignalR надає простий API для створення викликів віддалених процедур (RPC) від сервера до клієнта, які викликають функції JavaScript в клієнтських браузерах (і на інших клієнтських платформах) з серверного коду .NET. SignalR також включає API для управління з'єднаннями (наприклад, події підключення і відключення) а також групування підключень. SignalR підтримує функцію «проштовхування сервера», в якій код сервера може викликати клієнтський код в браузері за допомогою віддалених викликів процедур (RPC), а не модель запит-відповідь.

Додатки SignalR можуть масштабуватися до тисяч клієнтів за допомогою вбудованих і сторонніх постачальників горизонтального масштабування.

SignalR — це абстракція деяких транспортів протоколів та механізмів, які необхідні для роботи в реальному часі між клієнтом і сервером. SignalR спочатку намагається встановити з'єднання WebSocket, якщо це можливо, а при неможливості встановити WebSocket з'єднання намагається встановити з'єднання у такій послідовності:

1. Web Socket.
2. Forever Frame.
3. Server Send Events.
4. Long Polling.

**Web Socket**

WebSocket — це протокол, що призначений для обміну інформацією між браузером та веб-сервером в режимі реального часу. Він забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет. WebSocket спроектовано для втілення у веб-браузерах та веб-серверах, але може також використовуватись будь-яким клієнт-серверним застосунком. Прикладний програмний інтерфейс WebSocket був стандартизований W3C,

крім того протокол WebSocket стандартизований IETF як RFC 6455. Діаграма порівняння роботи протоколу WebSocket та протоколу HTTP показано на рисунку 32.

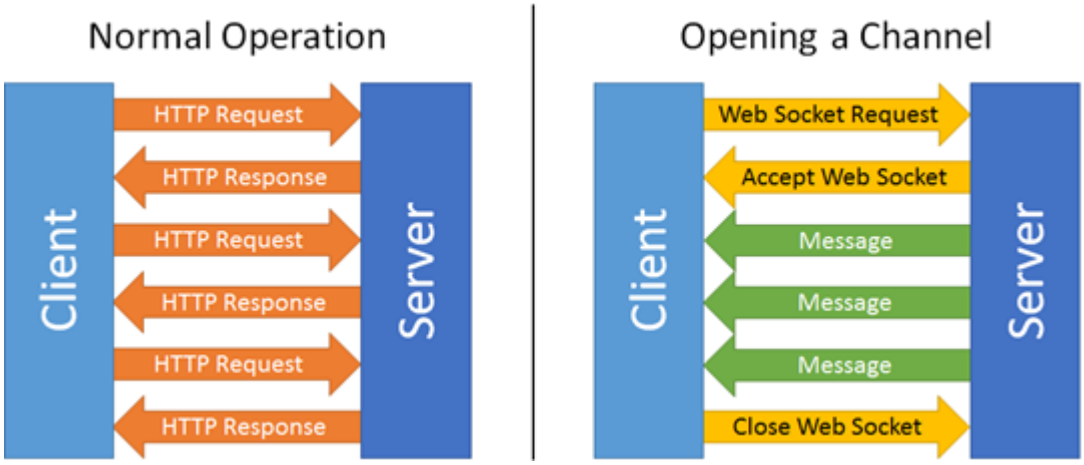



*Рис. 32 Порівняння WebSocket та HTTP*

**Forever Frame**

Forever Frame — це спеціалізована технологія комунікації у реальному часі між клієнтом і сервером, яка призначена тільки для Internet Explorer. При встановленні з'єднання через Forever Frame, на сторінці з'являється прихований IFrame. Сторінка, яка завантажена у IFrame, є особливою, бо з'єднання ніколи не закривається. Так як з'єднання ніколи не закривається, то сервер може використовувати це для безперервної відправки нових скриптів, які завантажуються на головну сторінку та виконуються на ній. При цьому будь-який обмін повідомленнями повинен буди реалізований стандартними викликами AJAX. Діаграма роботи технології показана на рисунку 33.

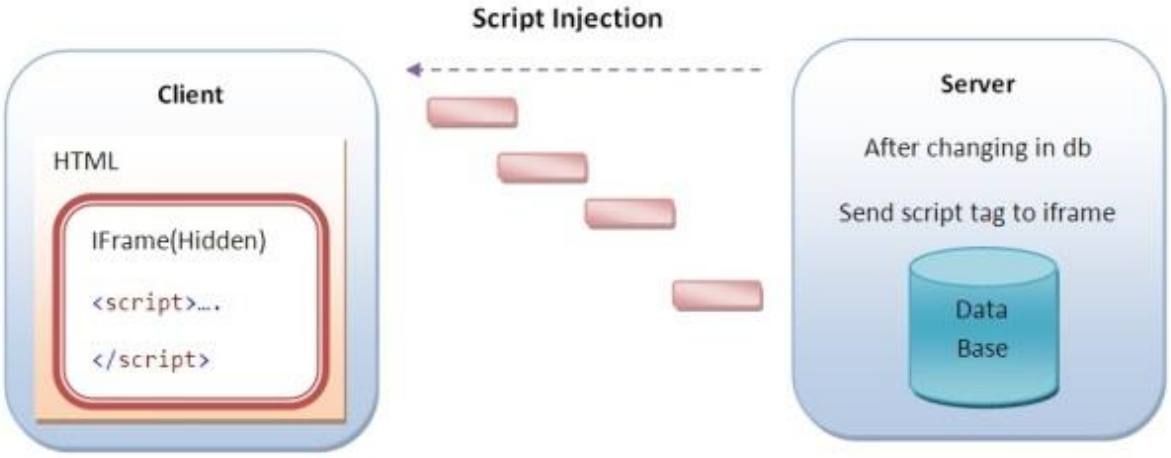



*Рис. 33 Технологія Forever Frame*

**Server Send Events**

Server Send Events — це технологія комунікації у реальному часі між клієнтом і сервером у вигляді DOM-подій. Технологія Server Send Events є стандартом, який описує способи передачі даних клієнтам з моменту організації клієнтом першого з'єднання (див. Рис. 35). Стандарт широко використовується для відправлення повідомлень про оновлення або для відправлення безперервних потоків даних браузеру клієнта. Вона спроєктована для поліпшення крос-браузерної взаємодії за допомогою JavaScript API під назвою EventSource. За допомоги цієї бібліотеки клієнт задає URL для отримання потоку подій, які йому потрібні. Підтримка браузерами зображена у таблиці 2.

Таблиця *2*

*Браузерна підтримка технології Server Send Events*

| Браузер | Підтримка | Примітка |
|---|---|---|
| Internet Explorer | Немає | - |
| Mozilla Firefox | Є | Починаючи з Firefox 6 |
| Google Chrome | Є | Починаючи з Chrome 6 |
| Opera | Є | Починаючи з Opera 11.5 |
| Safari | Є | Починаючи з Safari 5.0 |

| | | |
|---|---|---|
| Microsoft Edge | Є | Починаючи з Edge 79 |

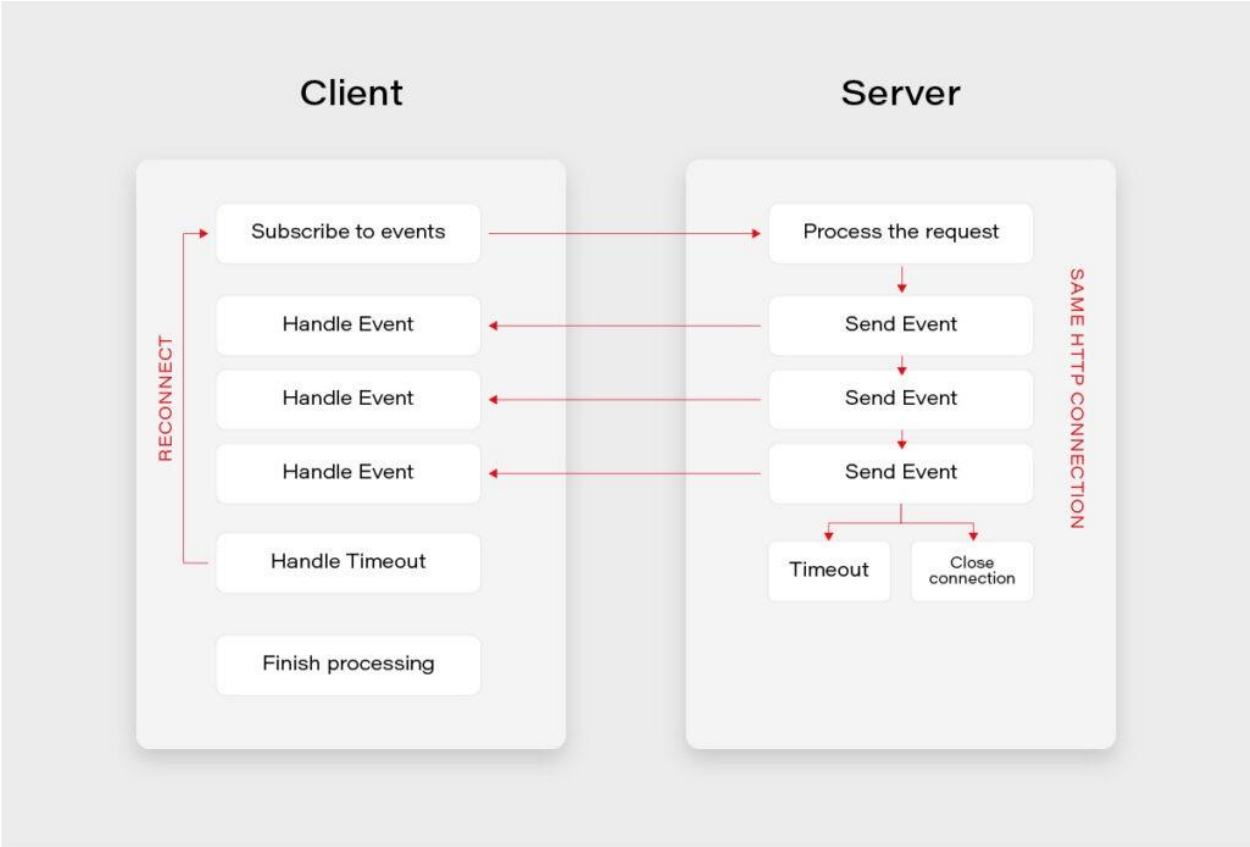



*Рис. 34 Схема роботи Server Send Event**s***

**Long Polling**

Long Polling — це технологія комунікації у реальному часі між клієнтом і сервером, яка базується на максимально дозволеному часі підвисання запиту HTTP. Під час цього процесу ви відкриваєте канал для сервера, щоб використовувати його для можливого майбутнього зв’язку. Щоразу, коли серверу потрібно надіслати повідомлення, можна використовувати наявне з’єднання. Однак, як тільки використовується з’єднання, з’єднання закривається. Для того, щоб продовжувати спілкуватися із сервером, клієнту потрібно буде відновити з’єднання. У найкращому випадку сервер безперервно надсилає дані, тому зв’язок завжди відновлюється. Гірший випадок — з’єднання залишається відкритим до двох хвилин (час очікування за замовчуванням у більшості браузерів). Слід відзначити, що процес безперервного ві-

дкриття та закриття з’єднань може трохи напружити сервер. Long Polling добре працює на старих браузерах, включаючи деякі браузери, які мали загинути багато років тому. Це остаточне резервна технологія, якщо ви повинні підтримувати старі браузери, які можуть працювати лише за даною технологією. Схема роботи Long Polling показана на рисунку 35.

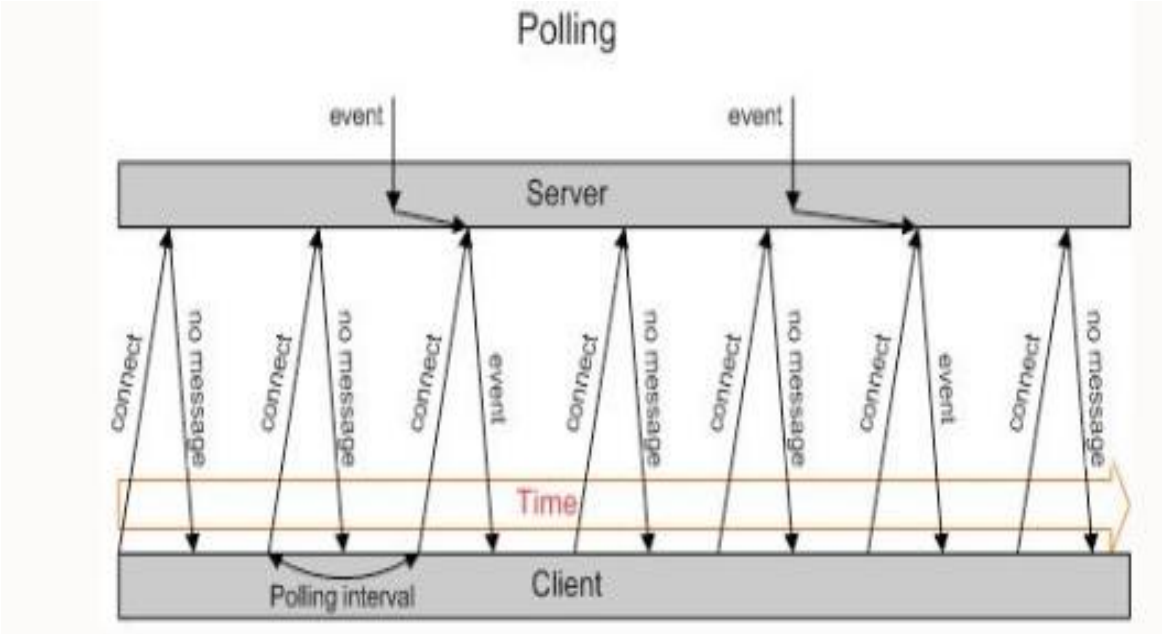



*Рис. 35 Схема роботи Long Polling*

### 3.2.9 Реляційна база даних PostgresSQL

PostgresSQL — це об’єктно-реляційна система керування базами даних. Її вважають альтернативою як комерційним СКБД, так і СКБД з відкритим кодом. Треба наголосити, що PostgresSQL є саме об’єктно-реляційною СКБД, що надає їй деякі переваги перед іншими СКБД з відкритим кодом, таких як MySql, MariaDB та FireBird. Фундаментальною характеристикою об’єктно-реляційних баз даних є підтримка користувацьких об’єктів та їх поведінки, що включає типи даних, функції, операції, домени та індекси. Все робить PostgresSQL гнучкою та надійною.

PostgresSQL підтримує числові типи, типи з плаваючою точкою, текстові, булеві. Також треба відзначити рідку підтримку uuid, грошового, типу

перерахунків (enumerations), геометричного та бінарного типів. PostgresSQL підтримує бітові строки, текстовий пошук, зберігання мережевих адрес, та багато іншого.

### 3.2.10 WebRTC

WebRTC — це браузерна технологія, призначена для передачі потокових даних між браузерами або додатками, з використанням технології p-to-p передачі. Однозначною перевагою цієї технології є можливість встановлювати зв’язок між користувачами, використовуючи тільки браузер. WebRTC не вимагає інсталяції додаткових плагінів. Необхідно просто написати код на JavaScript та використати HTML для відображення.

WebRTC має багато корисних інструментів, тож розглянемо декілька таких.

#### GetUserMedia

API дає змогу керувати користувацькими пристроями з браузера, наприклад:

1. Обрати камеру, з якої буде транслюватися відеопотік.
2. Обрати мікрофон.
3. Конфігурувати якість медіа-даних, що передаються.

Такі налаштування допомагають оптимізувати проєкт. Якщо користувач має погане з’єднання з мережею Інтернет, можна зменшити якість до 360 пікселів. Також необхідно відзначити можливість WebRTC керувати шумом. Технологія вдало видаляє шуми та фонові звуки з аудіопотоку.

#### RTCPeerConnection

Цей API-інтерфейс належить до технології Peer-To-Peer, яка є прямою комбінацією двох браузерів, які не використовують сервер. Отже, відбувається пряма передача відеопотоків з одного пристрою на інший.

У RTCPeerConnection є важлива функція — вбудована підтримка серверів STUN та TURN, які необхідні для обходу провайдера NAT, через якого відеопотоки можуть не доходити до користувачів. Схема RTCPeerConnection показана на рисунку 36.

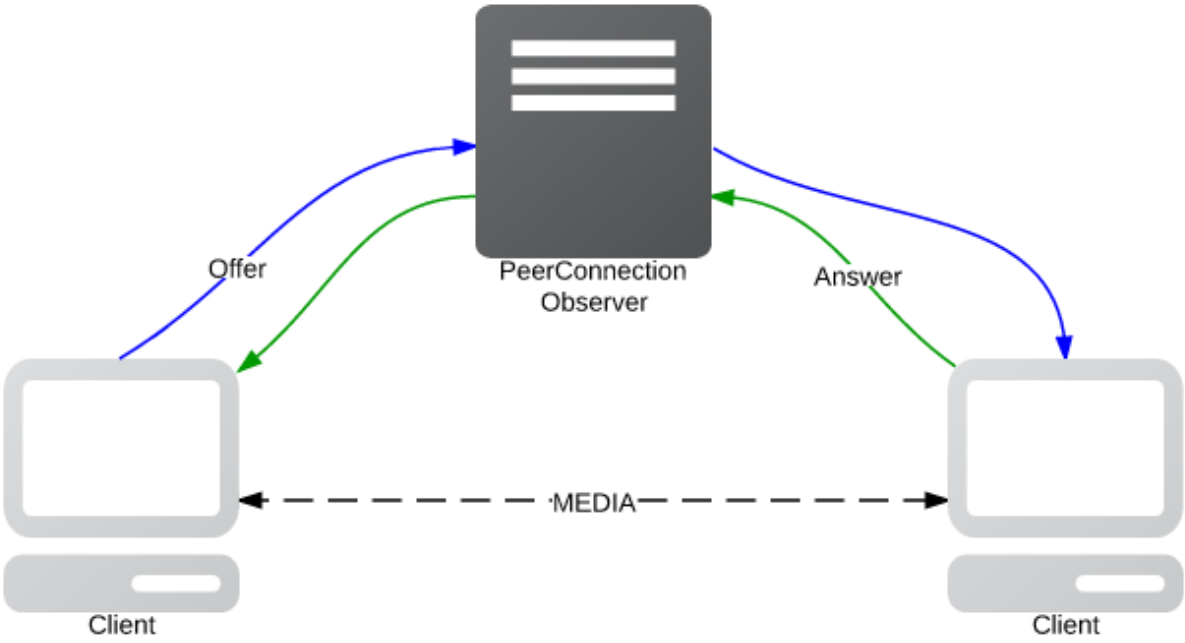



*Рис. 36 Схема RTCPeerConnection*

**Сигнальний сервер**

Однією з особливостей технології WebRtc є відсутність інструменту пошуку клієнтів. Тобто для того, щоб два браузери могли почати обмінюватися пакетами, їм необхідно точно ідентифікувати себе та партнера у глобальній мережі. Для вирішення цієї проблеми використовують сигнальний сервер. Розглянемо приклад з використанням сигнального сервера. Є сигнальний сервер, який тримає постійне з’єднання з нашими клієнтами, наприклад, через веб-сокети або за допомогою HTTP. Перший клієнт формує метаінформацію, та за допомогою веб-сокетів або HTTP пересилає її на сигнальний

сервер. Пересилає також певну частину інформації, з ким саме він бажає з’єднатися, наприклад, штучний ідентифікатор або ще якусь інформацію. Під штучним ідентифікатором мається на увазі ID користувача у системі, login тощо. Сигнальний сервер за цим ідентифікатором встановлює, якому саме клієнтові необхідно переадресувати нашу метаінформацію, та пересилає її. Другий клієнт бере її, використовує, встановлює і формує відповідь та за допомогою сигнального механізму пересилає її на сигнальний сервер, той так само ретранслює її першому клієнтові. Отже, обидва клієнти в цей момент мають усі необхідні дані та метаінформацію, щоб встановити P2P-з’єднання. Метаінформація представлена у вигляді SDP-датаграми (Session Description Protocol). Приклад SDP-датаграми показано на рисунку 37.

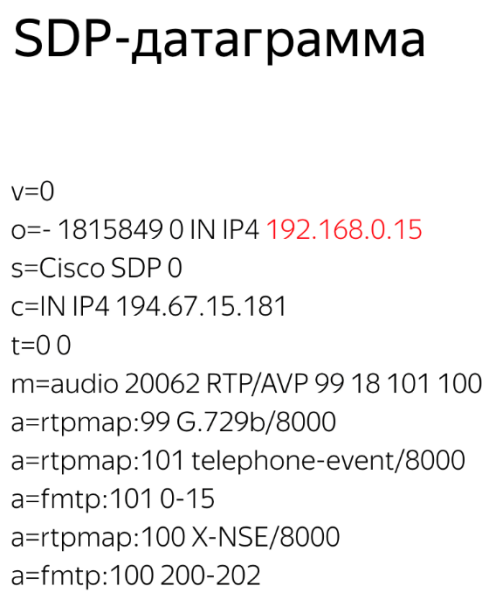



*Рис. 37 SDP-датаграма*

Це, власне кажучи, текстовий файл, який містить усю необхідну інформацію, щоб встановити з’єднання. Там є інформація про IP-адреси, про порти, які використовуються, про те, яка саме інформація передається між клієнтами, який тип інформації (аудіо або відео), які кодеки використовуються. Необхідно звернути увагу на другий рядок датаграми на Рис. 16. Там вказана

IP-адреса клієнта — 192.168.0.15. Очевидно, це IP- адреса комп'ютера, який знаходиться у певній локальній мережі. Однак якщо у нас є два комп'ютера, кожен з яких знаходиться в локальній мережі, кожен з яких знає свою IP-адресу у межах цієї мережі, але вони не знають зовнішніх IP-адрес, то вони не зможуть встановити зв'язок. Для того, щоб вирішити цю проблему, використовують фреймворк ICE.

### ICE фреймворк

ICE (Internet Connectivity Establishment) — це фрейворк, який описує способи встановлення зв'язку, якщо існує NAT. Головною особливістю ICE є використання додаткового STUN-серверу. Це спеціальний сервер, звертаючись до якого, ви можете дізнатися свою зовнішню IP-адресу. Отже, в процесі встановлення з'єднання P-to-P кожен з клієнтів повинен зробити запит до STUN-сервера, щоб отримати свою IP-адресу, та сформувати додаткову інформацію, IceCandidate, і за допомогою сигнального механізму обмінятися нею. Тоді клієнти будуть знати правильні IP-адреси один одного та зможуть встановити з'єднання P-to-P. Схема встановлення зв'язку за допомогою STUN-сервера показана на рисунку 38.

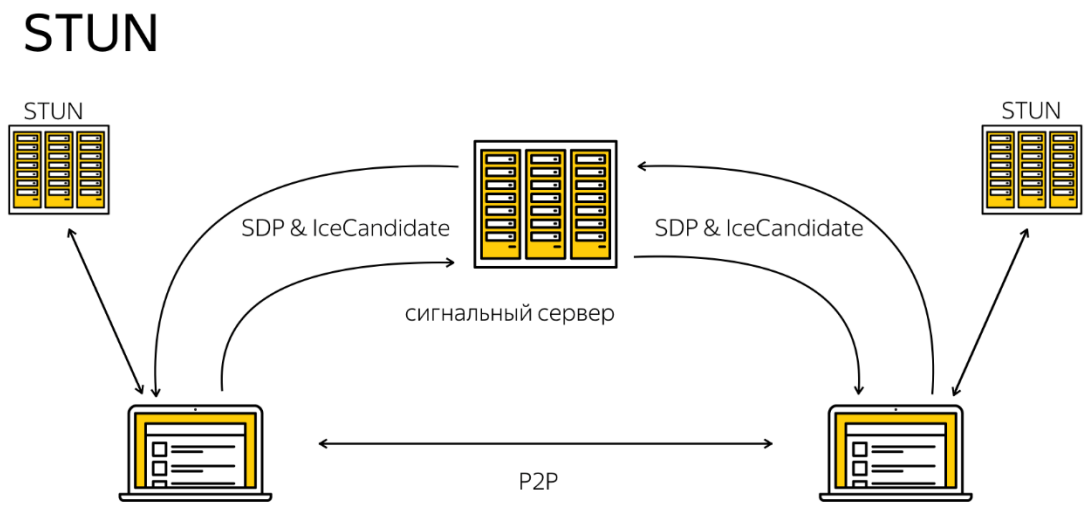


*Рис. 38 Використання STUN-сервера для з'єднання P-to-P*

Проте, бувають складніші випадки. Наприклад, коли комп'ютер прихований за подвійним NAT. В такому разі фреймворк ICE пропонує використання TURN-сервера. Це спеціальний сервер, який перетворює з'єднання «клієнт-клієнт», P-to-P, в з'єднання «клієнт-сервер-клієнт», тобто виступає в ролі ретранслятора. Хороша новина для розробників полягає в тому, що незалежно від того, відповідно до якому з трьох сценаріїв був встановлений зв'язок, чи знаходимося ми в локальній мережі, чи треба звернутися до STUN- або TURN-сервера, API-технологія для нас буде ідентичною. Ми просто на початку вказуємо конфігурацію ICE- та TURN-серверів, вказуємо, як до них звернутися, а після цього технологія використовує певні налаштування залежно від типу встановлення зв'язку. Схема встановлення зв'язку за допомоги TURN-сервера показана на рисунку 39.

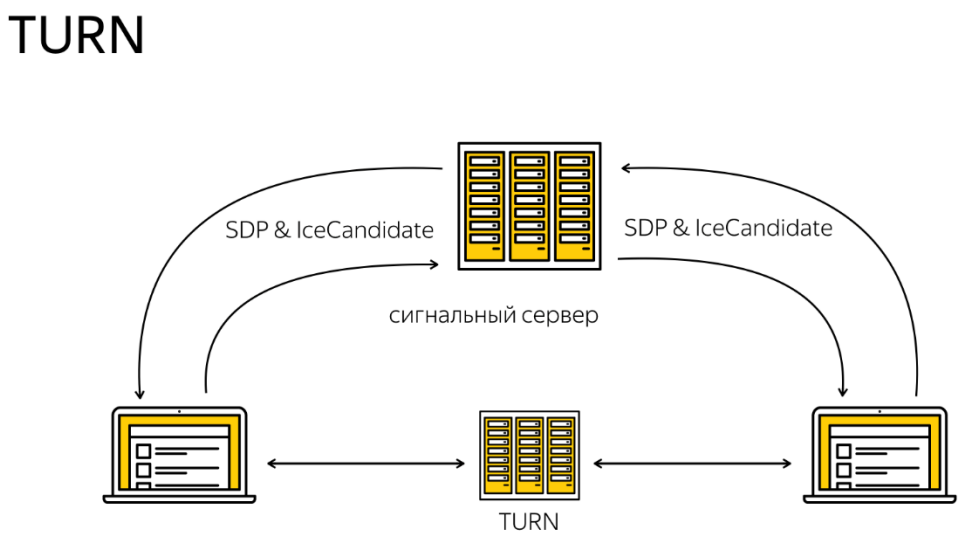



*Рис. 39 Використання TURN-сервера для вирішення проблеми подвійного NAT*

Також варто відзначити, що технологію підтримує багато браузерів. (див. Рис. 40).

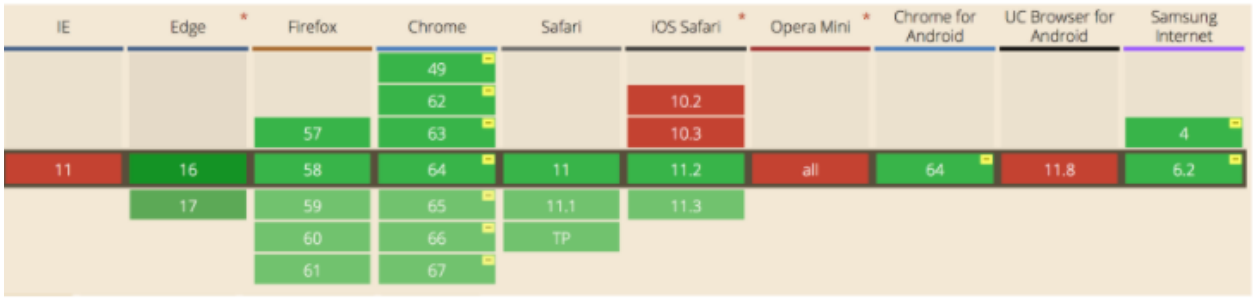



*Рис. 40 Підримка браузерами технології WebRTC*

### 3.3 Модулі та алгоритми

Під час розробки серверу застосунків було розроблено наступні модулі:

1. Модуль доступу до даних.
2. Модуль авторизації.
3. Модуль комунікації.

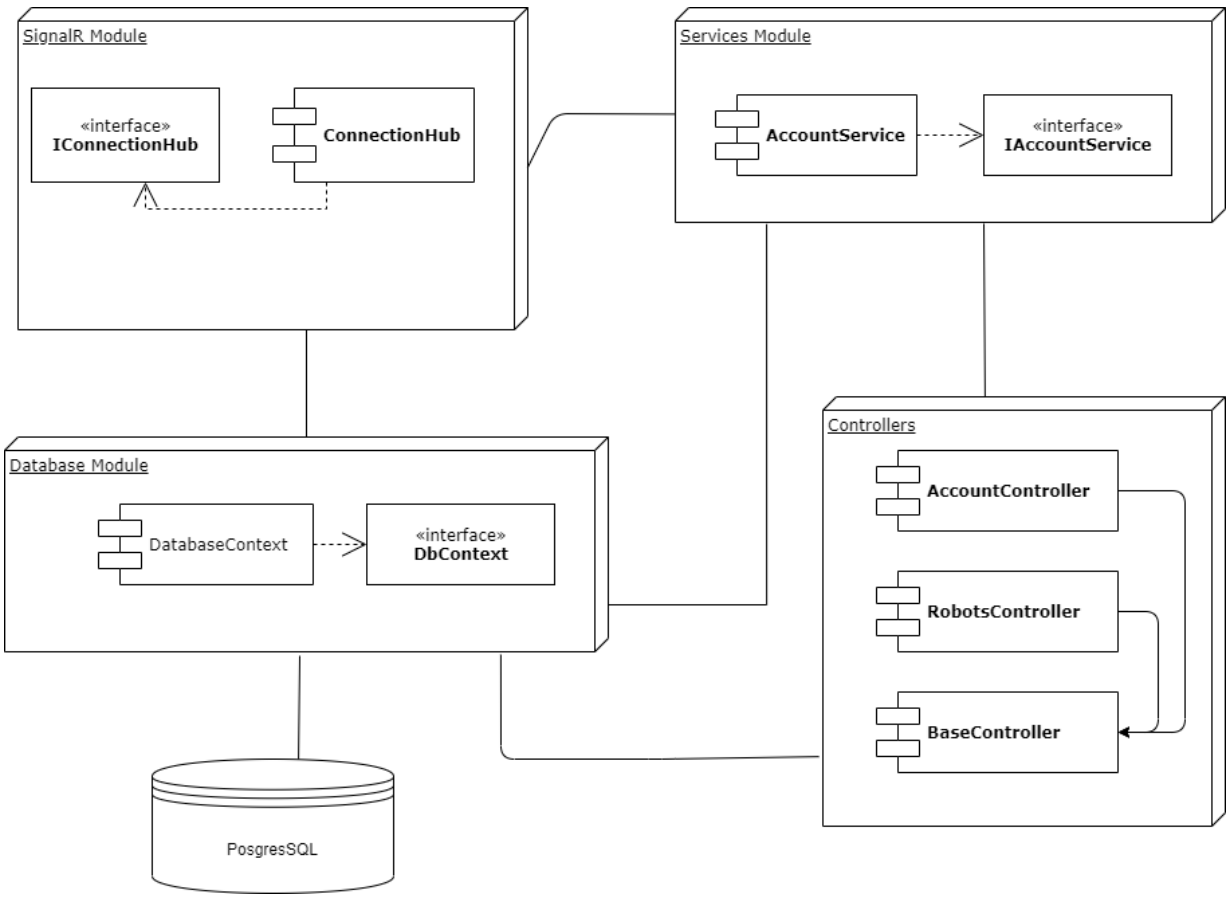



*Рис. 41 Спрощена структура взаємодії програмних компонентів серверу застосунків.*

**Модуль доступу до даних**

Модуль доступу до даних (див. Database Module на Рис. 41) спроектований та створений за допомоги фреймворку Entity Framework Core. Він використовує ORM для співвідношення класів, що мають об’єктно-орієнтовану структуру, до реляційних таблиць PostgresSql.

Entity Framework Core підтримує два підходи розробки:

1. Code First (спочатку код).
2. Database First (спочатку база даних).

При підході Code First, Entity Framework Core створює базу даних та таблиці за допомоги механізму міграцій, яка базується на конфігурації класів домену.

При підході Database First, Entity Framework Core створює класи домену та контексту на основі існуючої бази даних.

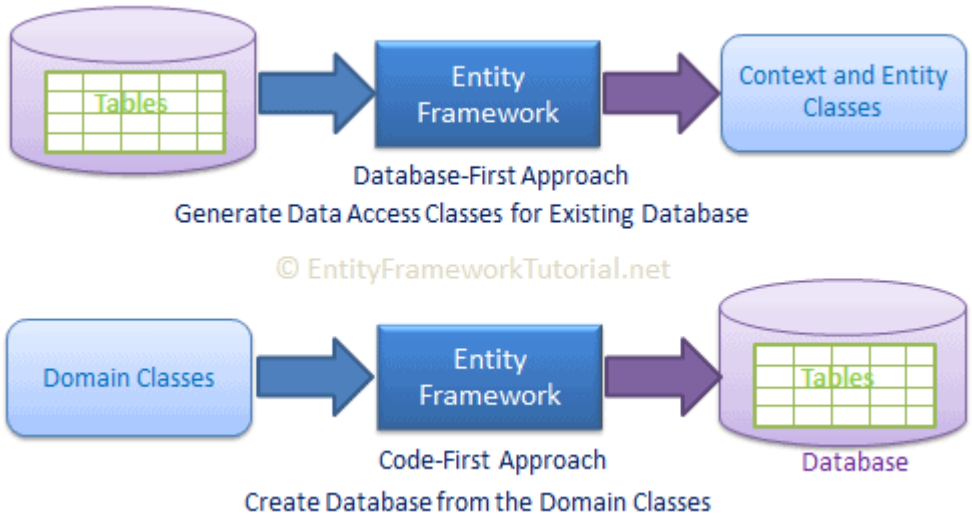



*Рис. 42 Підходи розробки Entity Framework Core*

При розробці серверу застосунків було обрано підхід Code First. У лістингу 5 наведено спрощений вид класу контексту бази даних:

*Лістинг 5 Клас контексту бази даних*

```
/*
```

```
     * Author : Danylo Starshekov
     */
    public class DatabaseContext : DbContext
    {
        private readonly IConfiguration configuration;

        // static constructor for mapping enum types
        // see https://www.npgsql. org/efcore/mapping/enum.
html?tabs=tabid-1
        static DatabaseContext()
        {
            NpgsqlConnection.GlobalTypeMapper.
MapEnum<DeviceStatusEnum>();
            NpgsqlConnection.GlobalTypeMapper.
MapEnum<DeviceTypeEnum>();
            NpgsqlConnection.GlobalTypeMapper.
MapEnum<RoleEnum>();
            NpgsqlConnection.GlobalTypeMapper.
MapEnum<SoftwareTypeEnum>();
        }

        public DatabaseContext(IConfiguration
configuration)
        {
            this.configuration = configuration;
            this.Database.EnsureCreated();
        }

        public DbSet<Customer> Customers { get; set; }

        public DbSet<Session> Sessions { get; set; }

        public DbSet<Device> Devices { get; set; }

        public DbSet<Location> Locations { get; set; }

        public DbSet<Software> Software { get; set; }
  }
```

**Модуль авторизації**

Модуль авторизації (див. AccountService на Рис. 41) представляє механізм авторизації, ідентифікації та аутентифікації користувача у системі. Алгоритм авторизації наступний: користувач вводить свій логін та пароль, після чого дані поступають до сервісу та починається процес ідентифікації, тобто

перевірка чи є у базі користувач з таким логіном. Якщо ідентифікація є успішною починається процес аутентифікації, тобто конвертація паролю до його хеш-вигляду шляхом використання спеціальної хеш-функції та порівняння цього хешу з тим, що відповідає логіну у базі даних. Якщо попередні кроки були успішними, тобто користувач з заданим логіном існує у системі та він підтвердив свою ідентичність шляхом вводу правильного паролю то система дізнається права користувача та но їх основі генерує для нього сесію. Сесія має обмежений час валідності, який дорівнює 24 годинам. Ідентифікатор сесії повертається користувачу у заголовках cookie для подальшого використання. Слід відзначити, що з кожним запитом до серверу застосунків потрібно додавати у заголовок cookie ідентифікатор сесії, якщо він є. Блок-схема алгоритму показана на рисунку 43.

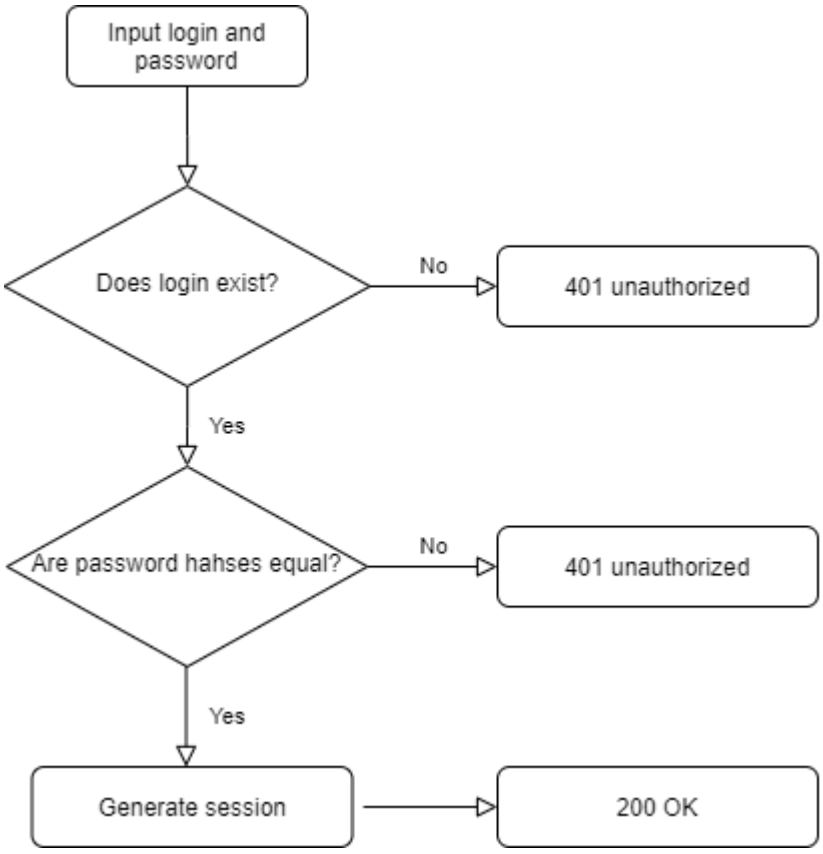



*Рис. 43 блок-схема алгоритму ідентифікації та аутентифікації*

Отже після генерації та отримання сесії користувач включає її ідентифікатор у cookie кожного запиту після чого сервер отримує цей ідентифіка-

тор, намагається знайти сесію з таким ідентифікатором, перевіряє її валідність та приймає рішення чи надавати права до запитуваного ресурсу. Блок-схема алгоритму авторизації показана на рисунку 44.

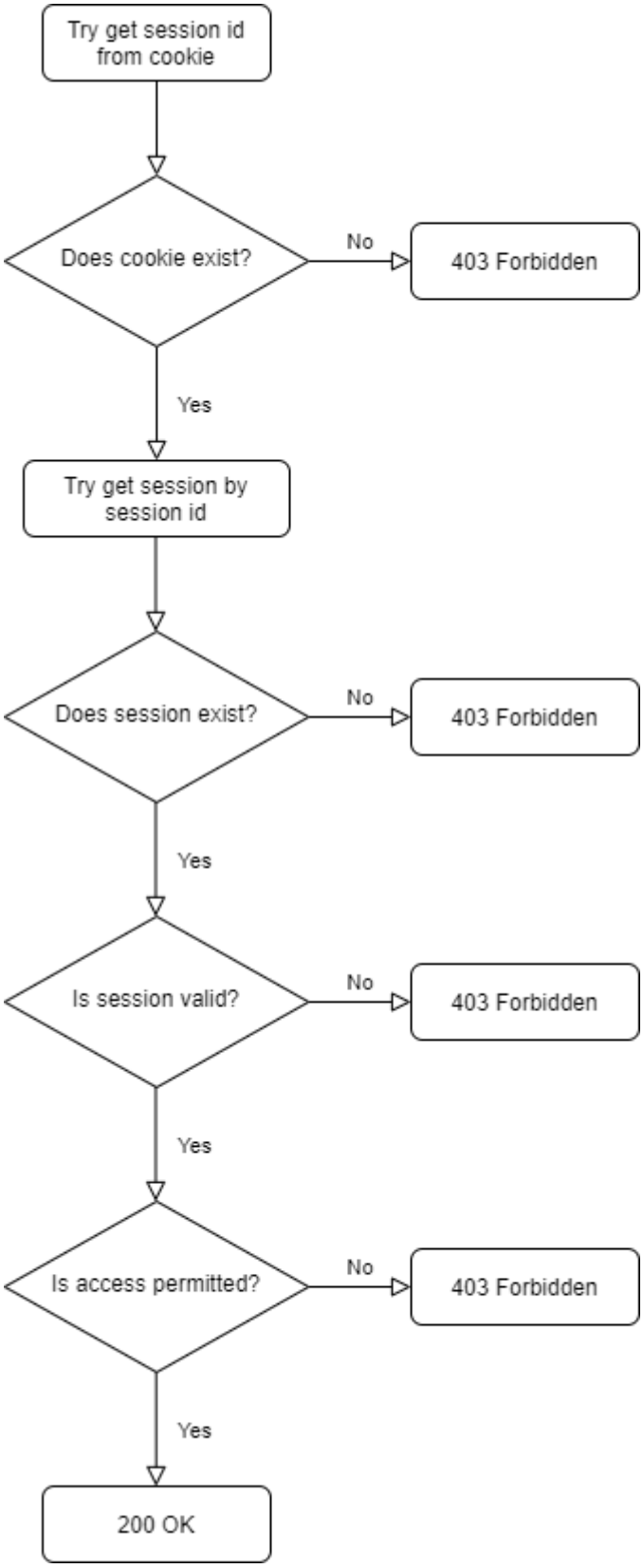


*Рис. 44 Блок-схема алгоритму авторизації*

Для кращої продуктивності пошуку сесії реалізовано зберігання сесії як у основній базі даних так і у «in memory» базі даних, яка розміщує дані у оперативній пам’яті комп’ютера, що значно пришвидшує пошук. Тому на етапі отримання сесії система спочатку шукає сесії у «in memory» базі даних. Якщо сесії не була знайдена, то сесії система шукає її у основній базі даних. На схемі це виглядає так:

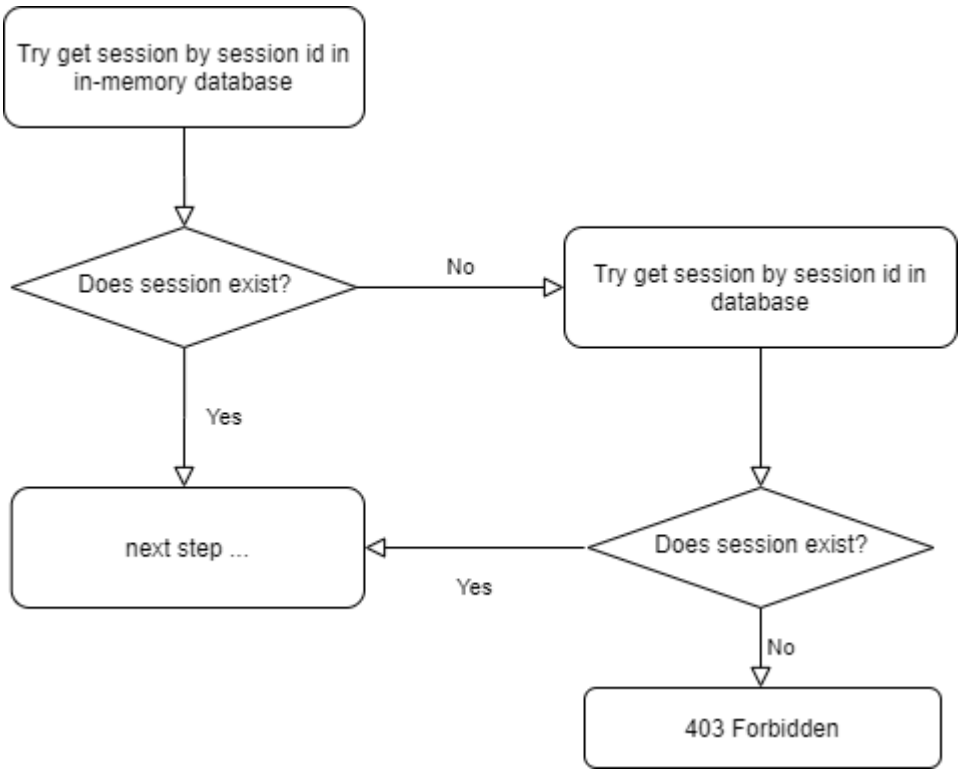



*Рис. 45 Алгоритм пошуку сесії*

Реалізація алгоритму ідентифікації та аутентифікації користувача представлена у лістингу 6.

*Лістинг 6 Алгоритм ідентифікації та аутентифікації*

```
public async Task Login(string email, string password)
{
  if (string.IsNullOrEmpty(email) ||
  string.IsNullOrEmpty(password))
{
```

```
  throw new RestException(HttpStatusCode.BadRequest,
  "Email and password are required");
}

Customer customer = null;
// try get customer's info by email
customer = await
this.databaseContext.FindCustomerByEmailAsync(email.ToLower
());
if (customer != null)
{
 // passwords match check
if(customer.Password.Equals(this.GetPasswordHash(password))
){
 // email and password are valid, so generating the session
 Session session = new Session()
 {
 CustomerId = customer.Id,
 SessionToken = Guid.NewGuid().ToString("N"),
       Role = customer.Role,
       ExpirationDate =
DateTime.Now.AddMinutes(ExpirationDateInMinutes),
 };
if (userLogins.TryGetValue(customer.Id, out byte
loginCount))
{
  if (loginCount >= 5)
  {
    this.logger.LogWarning(
    $"Too many active customer sessions!\n" +
    $"Id: {customer.Id}\n" +
    $"FirstName: {customer.FirstName}\n" +
    $"LastName: {customer.LastName}\n" +
```

```
$"Emai: {customer.Email}");
                        }
                    }
// add session to database
await this.databaseContext.CreateSession(session);
// Add to memory
lock (customerSessionsLocker)
{
 sessionByToken[session.SessionToken] = session;
}
// Increment login
lock (customerSessionsLocker)
{
if (userLogins.ContainsKey(customer.Id))
{
userLogins[customer.Id]++;
}
else
{
userLogins.TryAdd(customer.Id, 1);
                        }
                    }
// set cookie for client
CookieOptions options = new CookieOptions()
 {
     Expires = session.ExpirationDate,
 };

this.httpContextAccessor.HttpContext.Response.Cookies.Appen
d("session_token", session.SessionToken);
                }
else
{
```

```
throw new RestException(HttpStatusCode.Unauthorized,
this.localizer["WrongEmailOrPassword"]);
                }
            }
else
{
throw new RestException(HttpStatusCode.Unauthorized,
this.localizer["WrongEmailOrPassword"]);
            }
        }
```

Код функції хешування наведено у лістингу 7.

*Лістинг 7 Функція хешування*

```
public string GetPasswordHash(string password)
 {
   StringBuilder str = new StringBuilder();
   using (System.Security.Cryptography.SHA256 hash =
System.Security.Cryptography.SHA256.Create())
    {
      Encoding enc = Encoding.UTF8;
      byte[] result =
hash.ComputeHash(enc.GetBytes(password +
PasswordHashSalt));

     foreach (byte b in result)
     {
     str.Append(b.ToString("x2"));
     }
 }
 return str.ToString();
 }
```

**Модуль комунікації**

Модуль комунікації на сервері застосунків реалізований у виді класу Hub бібліотеки Microsoft.AspNetCore.SignalR. Цей модуль відповідає за встановлення зв'язку між клієнтами та роботами. Спочатку до серверу застосунків приєднується застосунок робота. Система надає роботу спеціальний ідентифікатор та зберігає його дані. Після встановлення зв'язку між сервером застосунків та роботом клієнт, який також приєднався, може побачити дані робота та отримує змогу приєднатися до робота. Тобто сервер застосунків представляє у даному випадку сигнальний сервер (частина архітектури технології WebRTC). Частина коду класу ConnectionHub представлено у лістингу 8.

*Лістинг 8 Код класу ConnectionHub*

```
public class ConnectionHub : Hub<IConnectionHub>
    {
        public static readonly List<Device> Devices = new
List<Device>();
        private static readonly List<Customer> Customers =
new List<Customer>();
        private static readonly List<DeviceCall> Calls =
new List<DeviceCall>();

        private readonly DatabaseContext databaseContext;
        private readonly IAccountService accountService;

        public ConnectionHub(DatabaseContext
databaseContext, IAccountService accountService)
        {
            this.databaseContext = databaseContext;
            this.accountService = accountService;
```

```
        }
        //// New customer connects to SignalR
        [Auth(RoleEnum.User, RoleEnum.Admin)]
        public async Task CustomerJoin()
        {
            Session session = await
this.accountService.CheckAuthorization(RoleEnum.User,
RoleEnum.Admin);
            Customer customer = await
this.databaseContext.FindCustomerByIdAsync(session.Customer
Id);
            customer.HubConnectionId =
this.Context.ConnectionId;

            ConnectionHub.Customers.Add(customer);
        }
        //// New Device connects to SignalR
        public async Task DeviceJoin(string id)
        {
            Device device = await
this.databaseContext.FindDeviceByIdAsync(Convert.ToInt32(id
));
            device.Status = DeviceStatusEnum.Available;
            device.HubConnectionId =
this.Context.ConnectionId;
            await
this.databaseContext.UpdateDevice(device);
            ConnectionHub.Devices.Add(device);
            //// Send updated device to all clients
            await this.SendDeviceUpdate(device);
            ////Send Software update list to device
            var software =
this.databaseContext.FindAllSoftwareAsync();
```

```
            await
this.Clients.Client(device.HubConnectionId).SoftwareUpdate(
JsonSerializer.Serialize(software));
        }
        public async Task SendCommand(string
targetConnectionId, string command)
        {
            var call = ConnectionHub.Calls.Find(c =>
c.Customer.HubConnectionId == this.Context.ConnectionId
                                                    &&
c.Device.HubConnectionId == targetConnectionId);
            if (call != null)
            {
                await
this.Clients.Client(targetConnectionId).ReceiveCommand(this
.Context.ConnectionId, command);
            }
        }
}
```

На стороні застосунку робота модуль комунікації представлений, як WebRTC клієнт, який використовує сигнальний сервер для встановлення зв'язку з клієнтом, після чого обмінюється даними з ним вже у режимі P to P.

Приклад підписки на події сигнального серверу та посилання повідомлення наведено у лістингу 9.

*Лістинг 9 Підписка на події клієнтом*

```
this.hubConnection =
HubConnectionBuilder.create("http://192.168.43.217:5000/con
nectionhub").build();
hubConnection.on("IncomingCall", this::incomingCall,
String.class);
private void incomingCall(String conid) {
        runOnUiThread(new Runnable() {
```

```
            @Override
            public void run() {
                try {
                    connectionId = conid;
                    hubConnection.send("AnswerCall", true,
conid);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
```

Для комунікації з веб-клієнтом було також реалізовано WebRTC клієнт. Приклад підписки на події сигнального серверу та посилання повідомлення наведено у лістингу 10.

*Лістинг 10 Підписка на події сигнально сервера та відправлення йому повідомлення*

```
class ConnectionHub {
    constructor() {
        this.peerConnectionConfig = { "iceServers": [{
"url": "stun:stun.l.google.com:19302" }] };
        this.hubUrl = document.location.origin +
'/signalr';

        this.wsconn = new signalR.HubConnectionBuilder()
            .withUrl(this.hubUrl,
signalR.HttpTransportType.WebSockets)

.configureLogging(signalR.LogLevel.None).build();

        this.webrtcConstraints = { video: true, audio: true
};
```

```
        this.devices = [];
        this.connections = [];
        this.localStream = null;
    }

async join(videoElementId) {
        this.VideoElementId = videoElementId;
        try {
            await this.wsconn.start();
            console.log("SignalR: Connected");

            await this.wsconn.invoke(CUSTOMER_JOIN);

            console.log("SignalR: Joined");

            this.wsconn.onclose(e => {
                if (e) {
                    console.log("SignalR: closed with
error.");
                    console.log(e);
                }
                else {
                    console.log("Disconnected");
                }
            });
        }
        catch (error) {
            console.error(error);
            return false;
        }
        return true;
    }
subscribeOn(eventName, callback) {
```

```
        console.log('SignalR subscribes ON ' + eventName);
        switch (eventName) {
            case RECEIVE_SIGNAL:
                this.wsconn.on(RECEIVE_SIGNAL, async
(signalingUser, signal) => {
                    console.log('SignalR ON ' +
RECEIVE_SIGNAL, signalingUser);
                    await
this.receiveSignalCallback(signalingUser, signal);
                    callback(signalingUser, signal);
                });
                break;
            case CALL_ESTABLISHED:
                this.wsconn.on(CALL_ESTABLISHED, async
(signalingUser) => {
                    console.log('SignalR ON ' +
CALL_ESTABLISHED, signalingUser);
                    await this.initiateOffer(signalingUser,
this.localStream);
                    callback(signalingUser);
                });
                break;
            default:
                this.wsconn.on(eventName, (data) => {
                    console.log('SignalR ON ' + eventName,
data);
                    callback(data);
                });
                break;
        };
    }
```

**Модуль інтерфейсу веб-застосунку**

Інтерфейс розроблений за допомоги патерну MVC, використовуючи HTML5, CSS3 та JavaScript для створення безпосередньо веб-сторінок. Кожна сторінка складається з окремого набору розмітки, стилів та скриптів. Маршрутизація реалізована у вигляді middleware та показана у лістингу 11.

*Лістинг 11 Реалізація MVC маршрутизації*

```
app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern:
"{controller=Home}/{action=Index}/{id?}");
                endpoints.MapHub<ConnectionHub>("/signalr",
options =>
                {
                    options.Transports                    =
Microsoft.AspNetCore.Http.Connections.HttpTransportType.Web
Sockets;
                });
            });
```

Отже, видно, що маршрутизація розділяє маршрути до представлень та SignalR модулю. Тепер коли клієнт звернеться за шаблоном /Controller/Action він отримає результат роботи методу контролера, який був указаний. Наприклад якщо користувач звернеться за відносною адресою /Home/Index в результаті він отримає сторінку, яку генерує метод Index контролера Home. Код контролеру за замовчуванням (HomeController) показано у лістингу 12.

*Лістинг 12 Реалізація MVC контроллера*

```
public class HomeController : BaseController
    {
```

```
        public HomeController(IAccountService
accountService, ILogger<HomeController> logger)
            : base(accountService, logger)
        {
        }
        public IActionResult Index()
        {
            return this.View();
        }
        public IActionResult Privacy()
        {
            return this.View();
        }
        public IActionResult Contact()
        {
            return this.View();
        }
}
```

Тут під змінною this.View мається на увазі cshtml-файл, який розташований у папці Views з назвою, яка співпадає з назвою методу. Структура файлів інтерфейсу показана на рисунку 46.

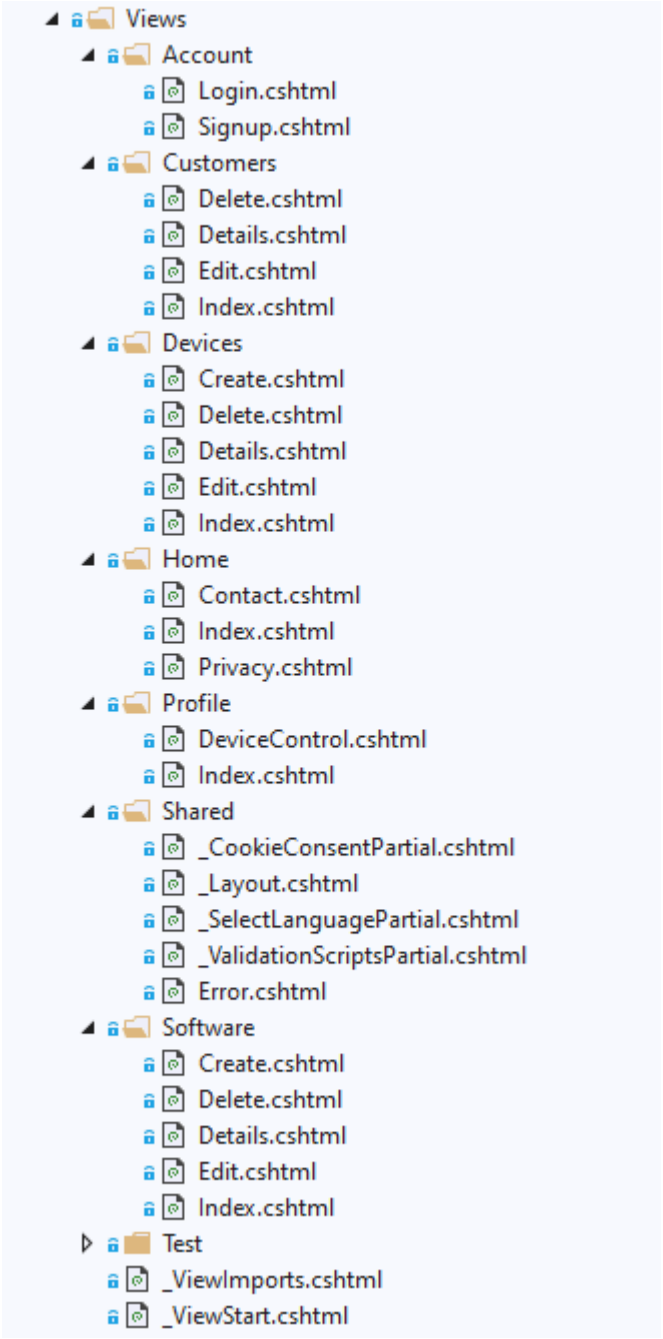



*Рис. 46 Структура файлів інтерфейсу*

**Алгоритми**

Для комп’ютерного 3D-моделювання руху колісного робота було обрано 4 режими:

1. Рух робота по заданим точкам.
2. Рух за кривою, згенерованою алгоритмами наближення.
3. Рух робота за математично заданою кривою.

Одним з алгоритмів наближення є наближення сплайнами.

**Наближення сплайн-функціями**

Взагалі задачі наближення у комп'ютерних системах належать до обчислювальної математики. Основною задачею обчислювальної математики є одержання наближених чисельних рішень, коли це неможливо зробити аналітичними методами, або коли використання аналітичних методів є надто складним. При вирішенні складних задач зазвичай доводиться використовувати одразу декілька методів, наприклад:

1. Чисельне диференціювання.
2. Інтегрування.
3. Визначення власних значень матриць.
4. Розв'язок системи лінійних алгебраїчних рівнянь (СЛАР).

Отже ефективність вирішення задачі повністю визначається характеристиками тих методів, які використовуються для кожної складової.

Існує перелік вимог для методів, які бажано виконувати:

1. Точність.
2. Стійкість.
3. Економічність.
4. Універсальність.

**Точність**

Кількісною мірою точності є величина похибки, яка за своєю природою поділяється на:

1. Таку, що не можна усунути.
2. Обчислювальну похибку.

Похибка, яку неможливо усунути, виникає внаслідок спрощень, які були прийняті при побудові моделі, а також через отримані неточні дані моделі.

Така похибка не може змінитись в процесі вирішення задачі, але обчислити її необхідно для того, щоб не отримати надмірно жорсткі умови точності.

У другому випадку похибка поділяється на похибку методу і похибку округлення. Якщо похибка методу є його індивідуальною характеристикою, то похибка округлення за своєю природою визначається машинною формою зображення числа. Так, у системі з плаваючою крапкою число зображується у вигляді:

$$x = \pm q^{p} \sum_{k=1}^{t} \alpha_k q^{-k},$$

де $q$ — ціле-основа системи числення; $\alpha_1,..,\alpha_t$ — цілі в межах $0 \le \alpha_k \le q$. Довжина мантиси $t$ визначається числом розрядів і є величиною обмеженою $t < t_0$. Очевидно, що при виконанні арифметичних дій довжина мантиси результату буде більше ніж $t_0$, а її частина, починаючи з $\alpha_{t+1}$, повинна бути відкинутою. Граничне значення $t_0$ визначається як типом ЕОМ, так і вимогами до системи зображення числа: звичайна точність, подвійна тощо.

**Стійкість**

Вважається, що метод є стійким, якщо невеликі зміни вхідних параметрів приводять до невеликих змін в результатах.

**Економічність**

Економічність методу визначається обсягом обчислень, необхідним для його реалізації. На практиці пріоритетними є вимоги точності та стійкості. Якщо виникає потреба вибору одного з кількох методів, то при виконанні цих двох умов проводиться порівняння за іншими характеристиками.

**Сплайни**

Сплайни — це група функцій однакової структури, локально визначених на відрізку $(x_i, x_{i+1})$, для яких виконуються умови безперервності як функцій, так і їхніх похідних потрібного порядку. Найбільш поширеними є сплайн-многочлени, а серед них — поліноми третього ступеня. Розглянемо побудову кубічного сплайну. У загальному випадку, щоб задати поліном третього ступеня, необхідно мати чотири коефіцієнти. Якщо таблиця складається з $n$ вузлів, то інтервалів між ними буде $n-1_i$ та треба визначити невідомі $4(n-1)$. Той факт, що функція та її перша та друга похідні безперервні у вузлах $n-2$, вони є рівносильними умовам $3(n-2)$. Крім того, вимога $s(x_i) = f(x_i)$ дає ще $n$ умов. Потрібні ще дві додаткові умови, щоб однозначно визначити сплайн. Якщо ці умови мають вигляд $s''(x_1) = s''(x_n) = 0$, то такий сплайн називається природнім. Отже, задача зводиться до розв'язку СЛАР з матрицею загального виду.

При зміні значення функції хоча б в одному вузлі, а також при вилученні чи додаванні вузлів, систему необхідно розв'язувати знову. На практиці існує ряд задач, у яких наближена функція повинна відповідати вимогам гладкості (безперервності похідних), але інтерполяційна таблиця змінюється настільки швидко, що кожного разу розв'язувати СЛАР навіть спрощеної структури не є технічно можливим. У такому разі використовують так званий згладжуючий сплайн, запис якого у параметричній формі має такий вигляд:

$$x(t) = ((a_3 \cdot t + a_2) \cdot t + a_1) + a_0,$$
$$y(t) = ((b_3 \cdot t + b_2) \cdot t + b_1 \; t \in (0;1),$$

де коефіцієнти обчислюються за такими формулами:

$$a_3 = \frac{1}{6}(-x_{i-1} + 3x_i - 3x_{i+1} + x_{i+2}),$$

$$a_2 = \frac{1}{2}(x_{i-1} - 2x_i + x_{i+1}),$$

$$a_1 = \frac{1}{2}(-x_{i-1} + x_{i+1}),$$

$$a_0 = \frac{1}{6}(x_{i-1} + 4x_i + x_{i+1}).$$

Така форма запису гарантує мінімальну кількість операцій для обчислення многочлена, а визначення коефіцієнтів не потребує розв'язку СЛАР. При зміні однієї точки ($x_i$, $y_i$) таблиці значення коефіцієнтів зміниться тільки на чотирьох відрізках. За безпосередніми обчисленнями можна переконатися, що вимоги безперервності функцій *x(t)*, *y(t)* та їх перших двох похідних будуть виконані, але значення наближеної функції у вузлі не співпадає з табличним. У лістингу 13 показана реалізація алгоритму.

*Лістинг 13 Реалізація алгоритму інтерполяції сплайнами*

```
public class SplineTravel : ITravel
{
    private Vector2[] points;
    private float step;

    public SplineTravel(Vector2[] points, float step)
    {
        this.points = points;
        Array.Sort(this.points, (p1, p2) =>
p1.x.CompareTo(p2.x));
        this.step = step;
    }

    public IEnumerator<Vector2> GetEnumerator()
    {
        int n = points.Length;
        float[] A = new float[n];
        float[] B = new float[n];
        float[] C = new float[n];
        float[] D = new float[n];

        B[0] = 1;
```

```
        C[0] = -1;
        D[0] = 0;

        A[n - 1] = -1;
        B[n - 1] = 1;
        D[n - 1] = 0;

        float[] h = new float[n];
        float[] delta = new float[n];
        for (int i = 0; i < n - 1; i++)
        {
            h[i] = points[i + 1].x - points[i].x;
            delta[i] = (points[i + 1].y - points[i].y) /
(points[i + 1].x - points[i].x);
        }

        for (int i = 1; i < n - 1; i++)
        {
            A[i] = h[i - 1];
            B[i] = 2 * (h[i - 1] + h[i]);
            C[i] = h[i];
            D[i] = delta[i] - delta[i - 1];
        }

        float[] sigma = GetProgonka(A, B, C, D);
        for (int i = 0; i < n - 1; i++)
        {
            for (float xx = points[i].x; xx < points[i +
1].x; xx += step)
            {
                float t = (xx - points[i].x) / h[i];
                float tCr = 1 - t;
```

```
                float yy = points[i + 1].y * t +
points[i].y * tCr + h[i] * h[i] * (sigma[i + 1] *
(Mathf.Pow(t, 3) - t) + sigma[i] * (Mathf.Pow(tCr, 3) -
tCr));
                yield return new Vector2(xx, yy);
            }
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }

    private static float[] GetProgonka(float[] A, float[]
B, float[] C, float[] D)
    {
        int n = D.Length;
        float[] x = new float[n];

        float[] R = new float[n];
        R[1] = -C[0] / B[0];
        float[] Om = new float[n];
        Om[1] = D[0] / B[0];

        for (int i = 1; i < n - 1; i++)
        {
            R[i + 1] = -C[i] / (A[i] * R[i] + B[i]);
            Om[i + 1] = (D[i] - A[i] * Om[i]) / (A[i] *
R[i] + B[i]);
        }
        x[n - 1] = (D[n - 1] - A[n - 1] * Om[n - 1]) / (A[n
- 1] * R[n - 1] + B[n - 1]);
```

```
        for (int i = n - 2; i >= 0; i--)
        {
            x[i] = x[i + 1] * R[i + 1] + Om[i + 1];
        }
        return x;
    }
}
```

### 3.4 Структури даних

Загальна система є комплексним рішенням, яке складається з багатьох підсистем. Центральною підсистемою є сервер застосунків, який серед описаного раніше контролює доступ до даних. Реалізація механізму авторизації потребує дуже частого пошуку сесій, що вплинуло на зберігання сесій не лише у базі а й у оперативній пам’яті. Для зберігання даних було вирішено використати безкоштовну систему керування базами даних PostgresSQL. На рисисунку 48 показана діаграма сутностей бази даних.

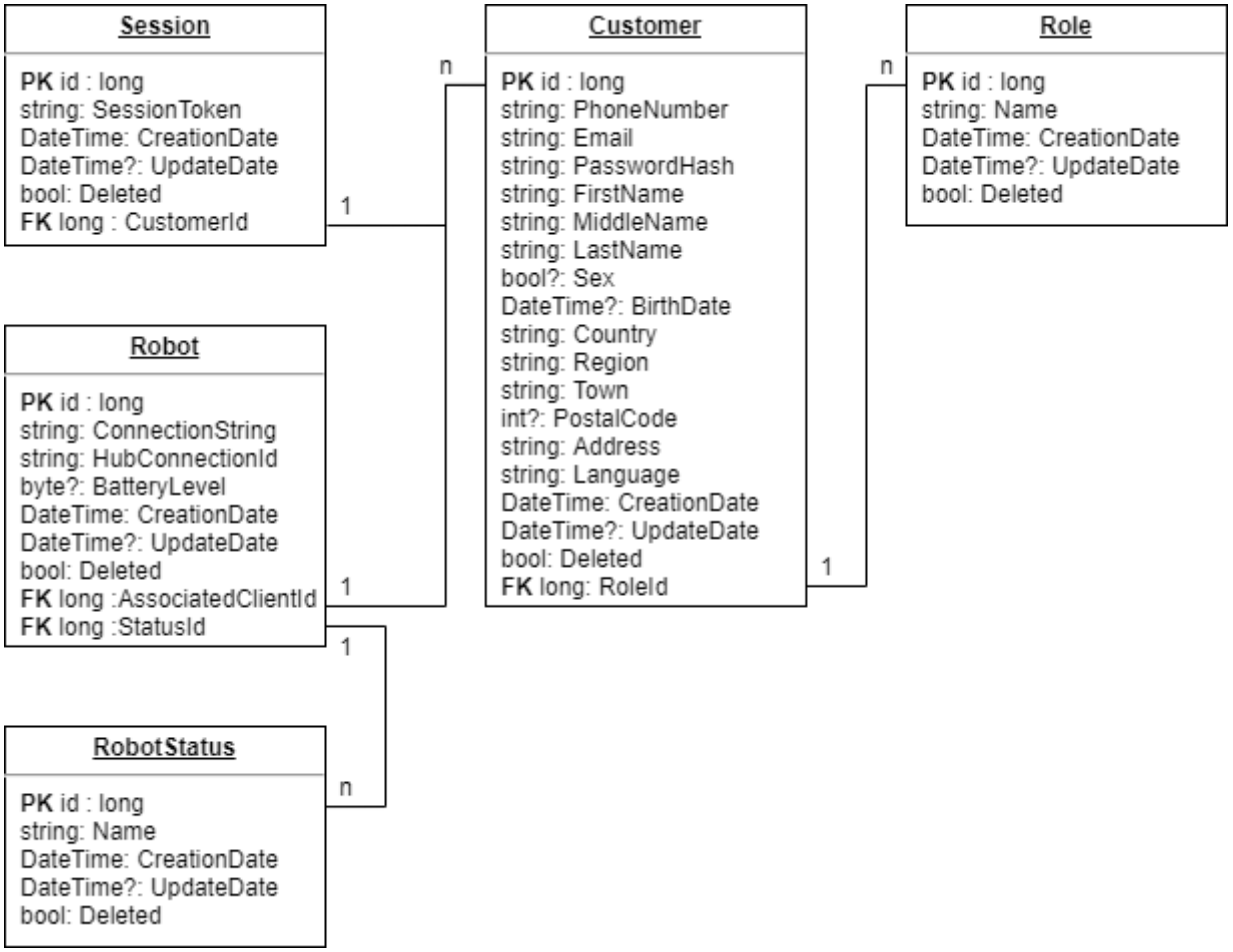



*Рис. 47 Діаграма сутностей бази даних*

Як видно с діаграми сутностей бази даних система має дві основні сутності: робот (Robot) та користувач (Customer). Сутність «користувач» зберігає у собі усі необхідні дані: ідентифікатор, прізвище, ім’я, по-батькові, електронну пошту, номер телефону, хеш паролю, стать, дату народження, адресу, країну, область, місто, поштовий код, мову, роль у системі. Також с діаграми видно, що кожна сутність має три службових поля:

1. CreationDate — дату створення запису у базі даних.
2. UpdateDate — останню дату модифікації.
3. Deleted — мітка, що означає видалений запис чи ні.

Останнє створене для імітації видалення запису з бази даних для того, щоб зберігати дані у базі навіть після їх видалення.

Таблиця Role є статичною, яка містить два ідентифікатора ролі користувача:

1. Admin.
2. User.

Сутність «робот» містить наступну інформацію: ідентифікатор сутності у базі даних, ідентифікатор для мережевого з’єднання, заряд батареї, дані клієнта, що керує роботом та статус. Таблиця статусу робота є статичною та містить такі типи статусу:

1. Доступний (Available).
2. Зайнятий (Occupied).
3. Не у мережі (Offline).
4. Заряджається (Charging).
5. Помилка (Error).

## 3.5 Проект інтерфейсу

### 3.5.1 Проект інтерфейсу веб-застосунку

Веб-застосунок дає змогу користувачу зареєструватися у системі та віддалено керувати роботом. Для реєстрації користувачу потрібно ввести обов’язкові поля (див. Рис. 48):

1. Email.
2. Пароль.
3. Підтвердження паролю.
4. Ім’я.
5. Прізвище.
6. Номер телефону.

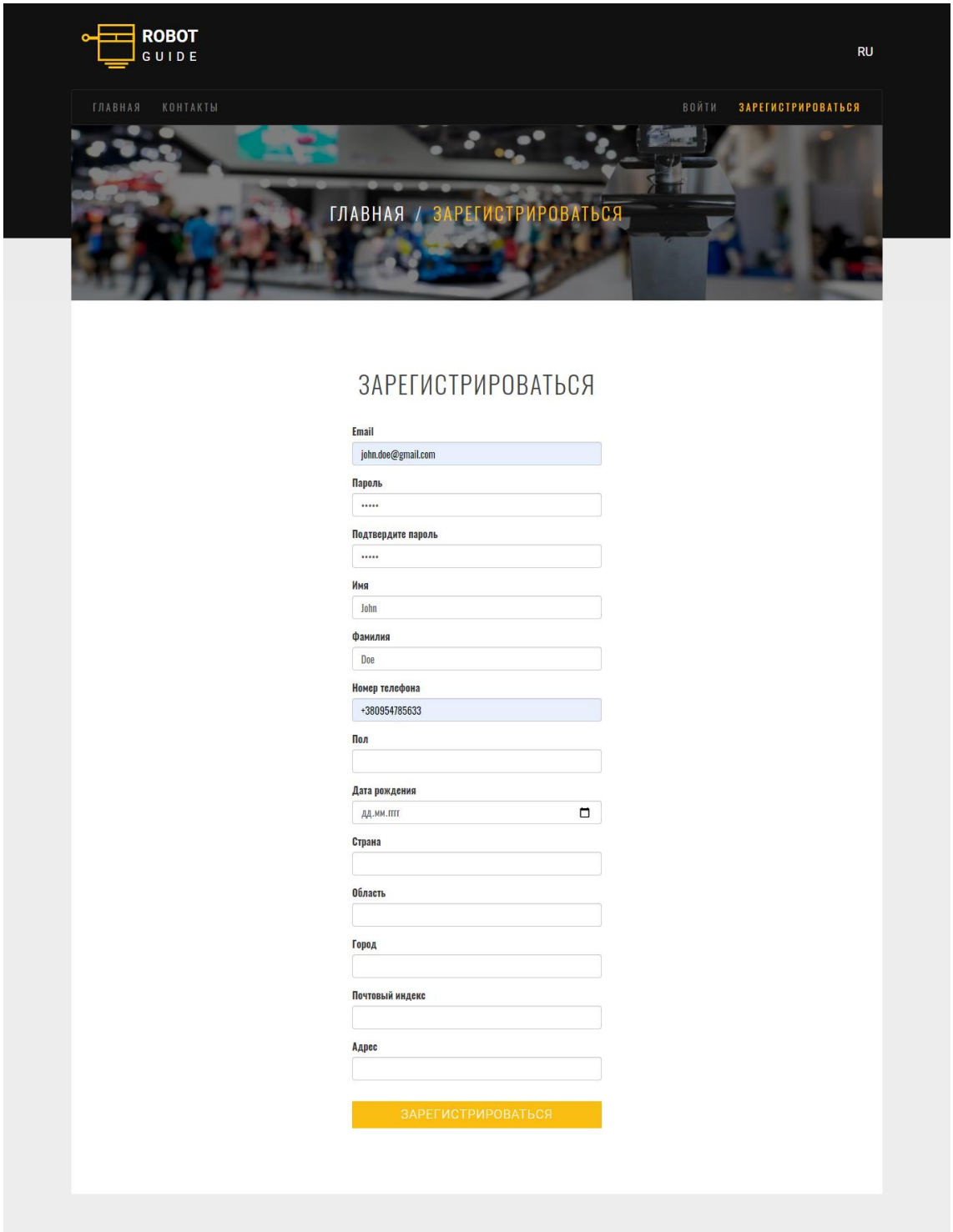



*Рис. 48 Реєстрація користувача у системі*

Після вводу даних треба натиснути кнопку «зареєструватися». При успішній реєстрації користувач попадає на сторінку управління роботами (див. Рис. 49).

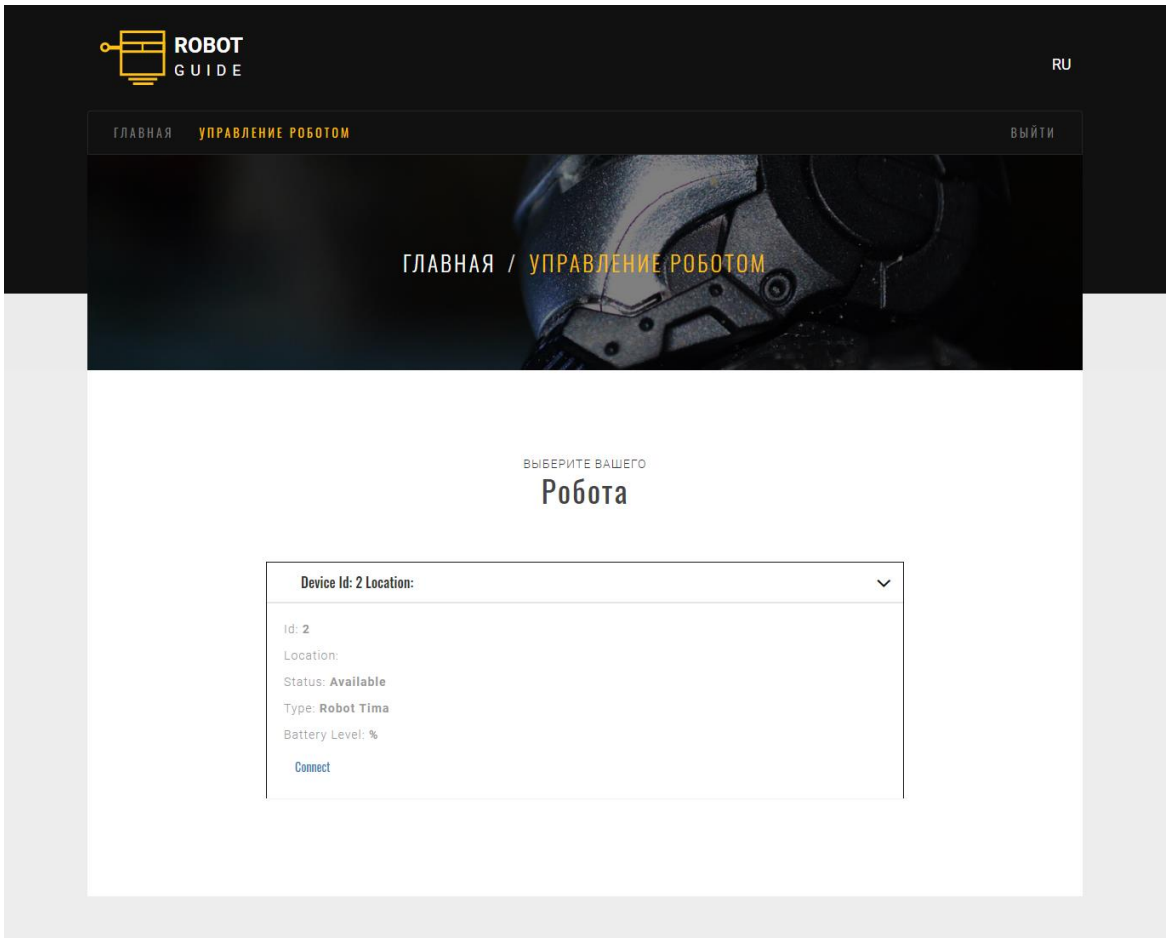



*Рис. 49 Сторінка управління роботами*

На сторінці управління роотами представлений список усіх роботів, які з’єднані з системою. При виборі конкретного робота користувач отримує базову інформацію про робота таку, як ідентифікатор, заряд батареї та статус. Для того щоб почати процес керування роботом треба натиснути кнопку «Connect». Треба відзначити, що кнопка буде активної тільки у тому випадку, якщо користувач має права на керування даним роботом та робот має вільний статус.

При вдалому з’єднанні з роботом, після натиснення кнопки Connect, користувач потрпляє на сторінку керування роботом (див. Рис. 50).

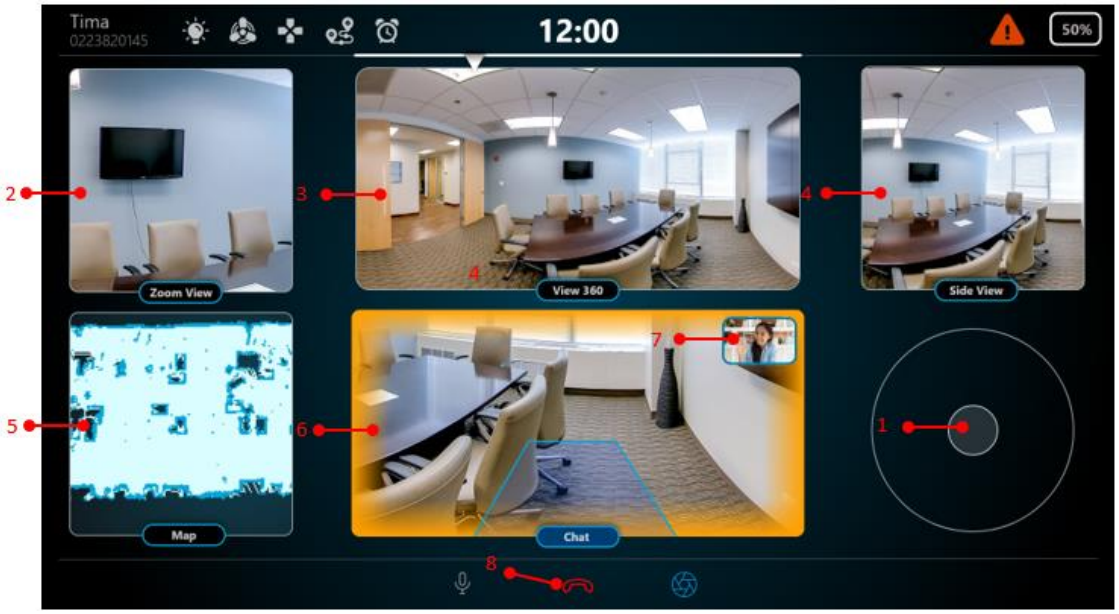



*Рис. 50 Сторінка керування роботом*

На сторінці є нступні елементи керування:

1. Джойстик.
2. Згорнуте зображення з камери 360.
3. Розгорнуте зображення з камери 360.
4. Зображення з додаткової камери.
5. Карта позіціонування робота у просторі.
6. Зображення з основної камери.
7. Зображення камери користувача.
8. Кнопка завершення з’єднання з роботом.

### 3.5.2 Проект інтерфейсу віконного застосунку

Віконний застосунок дає змогу задати варіант руху моделі колісного робота та реалізує рух. На екрані налаштування (див. Рис. 51) можна побачити наступні параметри:

1. Варіанти алгоритму руху робота.
2. Задання маршруту точками.
3. Значення кроку.
4. Швидкість роботу.

5. Задання функції математичної для побудови траєкторії та інтервали для неї.
6. Кнопка початку руху.
7. Кнопка зупинки руху.

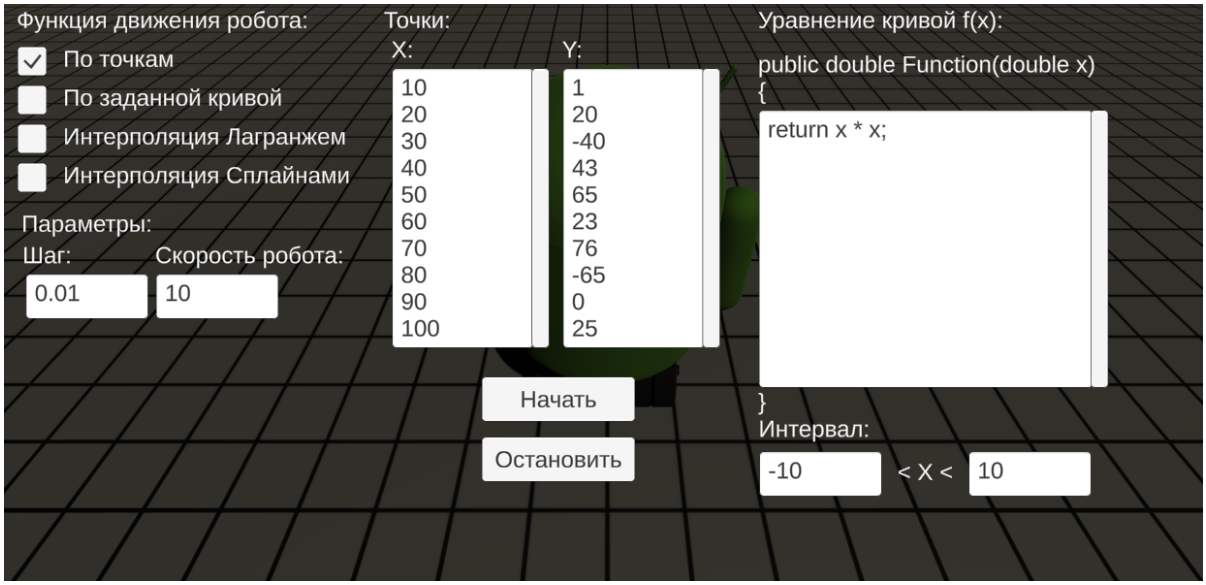



*Рис. 51 Вікно налаштування моделювання руху*

Після обрання алгоритму побудови траєкторії та встановлення всіх необхідних парметрів робот почне рух (див. Рис. 52, 53).

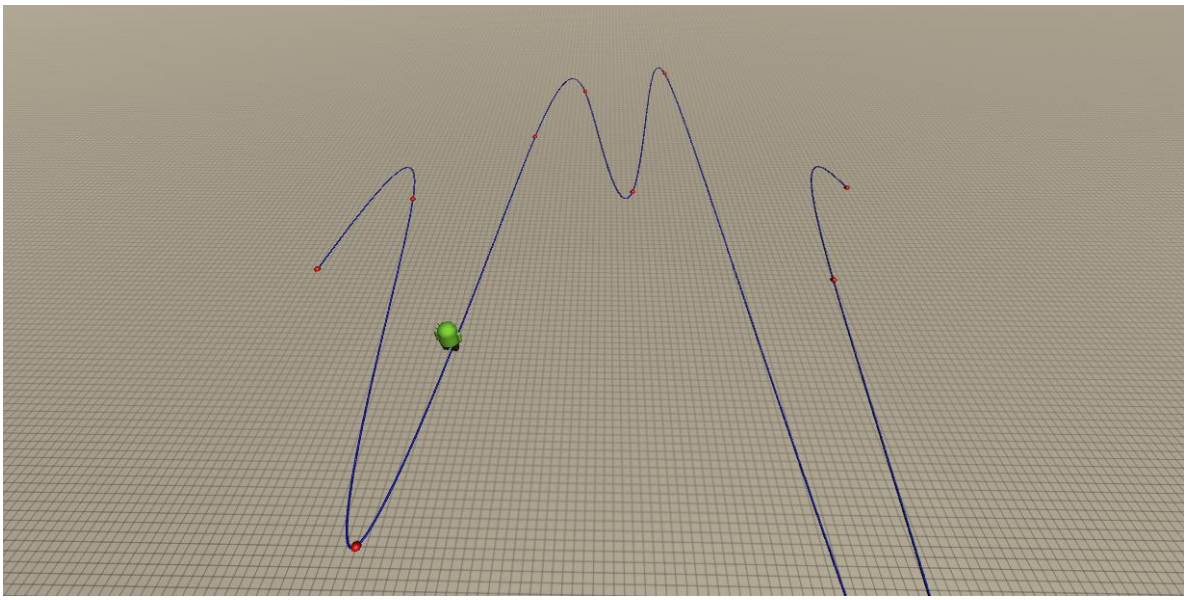


*Рис. 52 Рух робота при виді зверху*

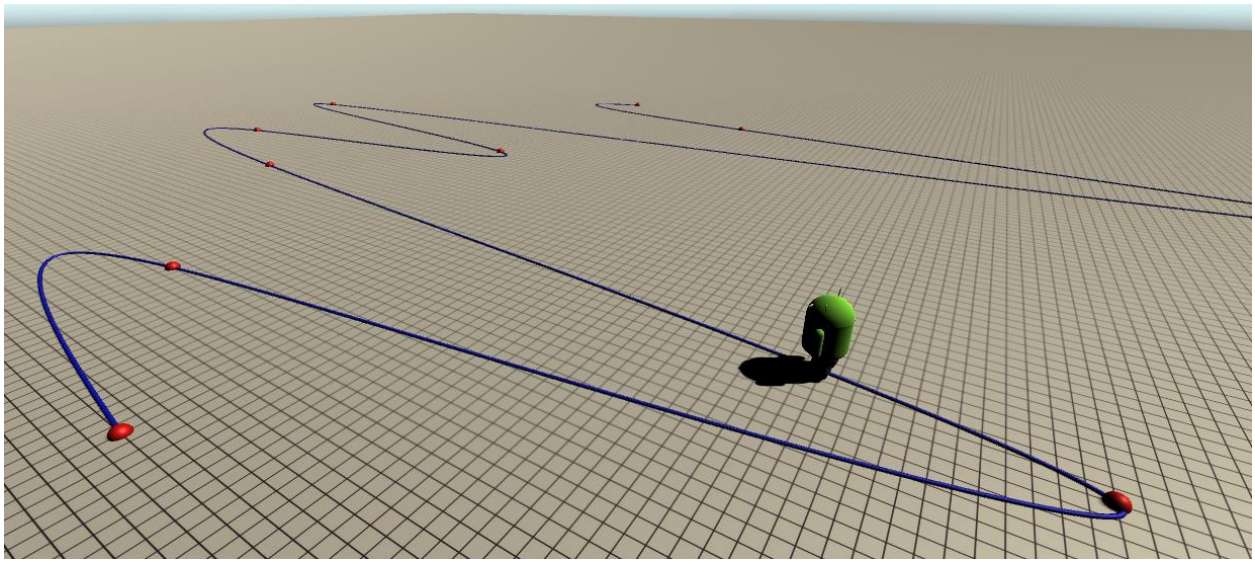


*Рис. 53 Рух робота при виді збоку*

## 3.6 Вимоги до апаратного забезпечення серверу застосунків

### 3.6.1 Вимоги до апаратного забезпечення серверу застосунків

Основною задачею серверу застосунків є втановлювати з’єднання між робот та клієнтом та зберігати дані, як у базі даних так і у оперативній пам’яті.

*Мінімальни вимоги до апаратного забезпечення серверу застосунків:*

1. Одноядерний процесор з базовою тактовою частотою 2.0 GHz.
2. Оперативна пам’ять ємністю не менше 6 Гб.

*Рекомендовані вимоги до апаратного забезпечення серверу застосунків:*

1. Чотирьох-ядерний процесор з базовою тактовою частотою 2.6 GHz.
2. Оперативна пам’ять ємністю не менше 8 Гб.

Так як клієнт використовую тільку браузер, та основним навантаженням є кодування та декодування аудіо та відео потоків з одного робота, то було визначено наступні вимоги до апаратного забезпечення:

*Мінімальни вимоги до апаратного забезпечення серверу застосунків:*

1. Двох-ядерний процесор з базовою тактовою частотою 2.0 GHz.
2. Оперативна пам’ять ємністю не менше 4 Гб.

*Рекомендовані вимоги до апаратного забезпечення серверу застосунків:*

1. Чотирьох-ядерний процесор з базовою тактовою частотою 2.6 GHz.
2. Оперативна пам’ять ємністю не менше 8 Гб.

### 3.6.2 Вимоги до апаратного забезпечення системи моделювання руху

Основною задачею системи моделювання руху є моделювання руху колісного робота у трьох-вимірному просторі. Оскільки застосунок є віконним та використовує 3D-графіку навантажується графічний процесор. Було визначено наступні вимоги до апаратного забезпечення:

*Мінімальни вимоги до апаратного забезпечення серверу застосунків:*

1. Чотирьох-ядерний процесор з базовою тактовою частотою 2.6 GHz.
2. Оперативна пам’ять ємністю не менше 6 Гб.
3. Відео-адаптор NVIDIA GeForce GTX 965M або аналоги.

*Рекомендовані вимоги до апаратного забезпечення серверу застосунків:*

1. Чотирьох-ядерний процесор з базовою тактовою частотою 3.4 GHz.
2. Оперативна пам’ять ємністю не менше 8 Гб.
3. Відео-адаптор NVIDIA GeForce GTX 1050 TI.

## 3.7 Опис функціональних можливостей

### 3.7.1 Опис функціональних можливостей системи керування

Розробле система складається з веб-клієнту, севреру застосунків та модульною системую, яка представляє робота. Система дає змогу зареєструватися користувачу та віддалено керувати роботом залежно від прав користувача. Користувачу транслюється відео з камер робота так само, як і робот отримує відео з веб-камери користувача. Користувач керує роботом шляхом відправки команд. Система створена, як тестова, та слугує для тесутвання фізичного прототипу трьох-колісного робота.

### 3.7.2 Опис функціональних можливостей системи моделювання

Система моделювання руху колісного робота дає змогу задати маршрут для руху колісного робота та відстежити його проходження. Маршрут можна задати координатами точок на площині. Також система може побудувати траєкторію на основі введених точок маршруту. Траєкторію маршруту також можна задати у вигляді математичної функції.

# РОЗДІЛ 4 АНАЛІЗ РЕЗУЛЬТАТІВ МАТЕМАТИЧНОГО ТА КОМП'ЮТЕРНОГО МОДЕЛЮВАННЯ КОЛІСНИХ РОБОТІВ

## 4.1 Дослідження технології для моделювання руху

Для порівняння було обрано дві конкуруючі інтегровані середовища розробки Unity 3D иа Unreal Engine 4. Також існує ряд спеціалізованого програмного забезпечення для моделювання руху, моделювання фізичної взаємодії твердих тіл, але не всі з них мають відкритий код та доступні для дослідження. Порівнювати програмне забезпечення такого типу можна за наступними критеріями:

1. Мови програмування.
2. Орієнтованиість на тип застосунку.
3. Платформи, що підтримуються.
4. Якість документації.
5. Умови комерційного використання.
6. Умови використання розширеної версії.

Таблиця *3*

*Підтримка мов програмування середовищами розробки*

| Мова програмування | Середовище розробки |
|---|---|
| C++ | Unreal Engine |
| C# | Unity |
| UnityScript | Unity |
| Boo | Unity |

Таблиця *4*

*Орієнтованість платформ на типи створюваних застосунків*

| Середовище розробки | Тип застосунку |
|---|---|
| Unreal Engine | Краще працєю для 3D |
| Unity | 2D, 3D |

Таблиця *5*

*Підтримка платформ середовищами розробки*

| Середовище розробки | Платформи |
|---|---|
| Unity | Підтримує 21 платформу, включаючи Web, Mobile, PC та Console |
| Unreal Engine | В основному підтримує PC та Console |

Таблиця *6*

*Якість документації для середовищ розробки*

| Середовище розробки | Якість документації |
|---|---|
| Unity | Детальна документація |
| Unreal Engine | Слабка документація |

Таблиця *7*

*Умови комерційного використання*

| Середовище розробки | Умови комерційного використання |
|---|---|
| Unreal Engine | Безкоштовно при обороті доходу менше 50 тисяч доларів США. |
| Unity | Безкоштовно при обороті доходу менше 100 тисяч доларів США. |

Таблиця *8*

*Умови використання розширеної версії продукту*

| Середовище розробки | Умови використання розширеної версії продукту |
|---|---|
| Unreal Engine | Безкоштовно з часткою доходу 5% |
| Unity | 1500 $ за Pro-версію або 75$ / місяць |

Отже можна зробити висновок, що середовище розробки Unity 3D є більш універсальним у порівнянні з Unreal Engine 4. Можливості Unity досить обширні, що підійде для створення застосунків для специфічних платформ. Детальність документації допоможе краще вивчити цю платформу, на основі чого можна зробити висновки, що Unity 3D краще підходить для розробників, які тільки освоюють розробку ігрових застосунків. В той час як Unreal Engine 4 надає кращу графіку, орієнтована на FPS-застосунки та потребує знань у області ігрової розробки.

### 4.2 Дослідження браузерних технології передачі відео та аудіо даних

Для реалізаії віддаленого керування колісним роботом була поставлена задача передачі аудіо та відео потоку від клієнта роботу та навпаки. Можна констатувати, що віддалене керування фізичним об’єктом потребує мінімальної затримки в передачі пакетів. Були проаналізовано наступні технології:

1. WebRTC.
2. Flash RTMFP.
3. Java Applet.

Таблиця 9

*Підтримка мережевих протоколів та специфікацій*

| | Java Applet | Flash RTMFP | WebRTC |
|---|---|---|---|
| UDP | * | Не підтримує | Не підтримує |
| TCP | * | Не підтримує | Не підтримує |
| RTMP | * | Підтримує | Не підтримує |
| RTMFP | * | Підтримує | Не підтримує |
| RTP | * | Не підтримує | Не підтримує |
| SRTP | * | Не підтримує | Підтримує |
| DTLS | * | Не підтримує | Підтримує |
| Media Encryp-tion | * | Підтримує | Підтримує |
| SIP | * | Не підтримує | Не підтримує |
| ICE | * | Не підтримує | Підтримує |
| STUN | * | Не підтримує | Підтримує |
| TURN | * | Не підтримує | Підтримує |
| DTMF | * | Не підтримує | Не підтримує |
| HOLD | * | Не підтримує | Не підтримує |

Таблиця *10*

*Підтримка відео-кодеків*

| | Java Applet | Flash RTMFP | WebRTC |
|---|---|---|---|
| H.264 | * | Підтримує | Не підтримує |
| H.263, H.263+ | * | Не підтримує | Не підтримує |
| Sorenson Spark | * | Підтримує | Не підтримує |
| VP8 | * | Не підтримує | Підтримує |

Таблиця *11*

*Підтримка аудіо-кодеків*

| | Java Applet | Flash RTMFP | WebRTC |
|---|---|---|---|
| G.711 | * | Підтримує | Підтримує |
| G.729 | * | Не підтримує | Не підтримує |
| Speex | * | Підтримує | Не підтримує |
| NellyMoser | * | Підтримує | Не підтримує |
| Opus | * | Не підтримує | Підтримує |
| iLBC | * | Не підтримує | Не підтримує |

Таблиця *12*

*Підтримка веб-браузерами*

| | Java Applet | Flash RTMFP | WebRTC |
|---|---|---|---|
| *FireFox 18+* | Підтримує | Підтримує | Підтримує |
| *Google Chrome 30* | Підтримує | Підтримує | Підтримує |
| *Microsoft Edge* | Підтримує | Підтримує | Підтримує |
| *Safari 6* | Підтримує | Підтримує | Не підтримує (підтримка з 11 версії) |
| *Opera* | Підтримує | Підтримує | Не підтримує (підтримка з 12 версії) |

Символ * означає, що теоретично кожен з пунктів може бути реалізований на Java, але це може займати багато часу та відповідальність за коректність роботи буде лежати на розробнку. Також слід відзначити, що реалізація може займати досить багато місця.

Порівнюючи технології Java Applet, Flash RTMFP та WebRTC слід відзначити, що Java Applet та Flash RTMFP — це плагіни. Тому, якщо говорити про технології, яка не потребує встановлення плагінів, WebRTC не має аналогів.

## 4.3 Дослідження руху фізичного прототипу при віддаленому керуванні

Фізичного прототип трьох-колісного робота був протестований на отримання та виконання команд керування. Команди керування відправляли кут повороту та статичне значення швидкості. У результаті проведених експериментів колісний робот виконував рух прямо та назад, повороти ліворуч, праворуч та повний на 360 градусів. Було протесовано рух на підйом з нахилом 10% та спуск з нахилом 10%.

Одною з особливостей віддаленого керування колісного робота стала технологія комунікації користувача та робота — WebRTC. При ситуації коли або користувач, або робот знаходився за подвійним NAT, технологія переключається з режиму передачі даних P to P на Client – Server – Client, застосовуючи TURN сервер. При географічно віддаленому розташуванні TURN сервера від робота або користувача виникала помітна затримка передачі відео та аудіо потоку, що значно ускладнювало керування фізичним прототипом.

# ВИСНОВКИ

1. Встаговлено актуальність дослідження проблеми руху колісних роботів. Шляхом аналізу технологій для створення прогрманого забезпечення моделювання руху фізичних об’єктів було створенно програмне забезпечення для моделювання руху колісного робота.

2. Досліджено методи створення програмного забезпечення для моделювання руху.

3. Досліджено інструменти мережевої комунікації у реальному часі.

4. Інструментом для реалізації мережевої взаємодії було обрано технологію WebRTC.

5. Інструментом среалізації сигнального сервера було обрано технологю SignalR.

6. Розроблено програмне забезпечення для віддаленого керування колісним роботом.

7. Розроблено програмне забезпечення для моделювання руху колісних роботів.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мартыненко Ю.Г. Управление движением мобильных колесных роботов. МГУ им. М.В. Ломоносова. Фундаментальная и прикладная математика, 2005, том 11, № 8, С. 29–80.

3.Андрианова О.Г. Моделирование движения колесного робота по заданному пути. МГТУ им. Баумана. Электронное научно-техническое издание Наука и образование, 2011, №10, С. 1–15.

4. Нефедов Г.А. Стабилизация движения двухколесного робота с дифференциальным приводом по заданному пути. Наука и образование. Москва: Научное издание МГТУ им. Н.Э.Баумана, 2013. С. 131–138.

5. Ткачев С.Б. Реализация движения колесного робота по заданной траектории. Вестник МГТУ им. Н.Э.Баумана. Естественные науки. 2008. №2. С. 33–55.

6. Зенкевич С.Л., Назарова А.В. Система управления мобильного колесного робота. Вестник МГТУ им. Н.Э. Баумана. Сер. “Приборостроение”. 2006. №3

7. Девянин Е.А. О движении колесных роботов. Труды конф.“Мобильные роботы и мехатронные системы”. Москва, 1998. С. 169–200.

8. Зенкевич С.Л., Назарова А.В., Лисицын Д.М. Моделирование движения робота по сложному маршруту. Труды конф. “Мобильные роботы и мехатронные системы”. Москва, 2000. С. 14–27.

9. Кобрин А.И., Мартыненко Ю. Г. Неголономная динамика мобильных роботов и ее моделирование в реальном времени. Труды конф. “Мобильные роботы и мехатронные системы”. Москва, 1998. С. 107–123.

10. Неймарк Ю.И., Фуфаев Н.А. Динамика неголономных систем. Москва: Наука, Физматлит, 1967. 520 с.

11. D. Cruz, et al, Decentralized Cooperative Control, IEEE Control Systems Magazine, June 2007, pp. 58-78.

12. Harry G. Kwatny, Bor-Chin Chang, Shiu-Ping Wang Static bifurcation in mechanical control systems. In Book: Bifurcation Control, Springer, Berlin, Heidelberg. 2003, pp. 67–81.

13. Tatievskyi D. Control synthesis for 4WS vehicle-robot model for traffic program motion, "Technology audit and production reserves". Systems and control processes. 2019. – Vol 4, No 2(48) . pp. 16–23.
14. Вербицький В. Г., Безверхий А. І., Татієвський Д. М. Комп'ютерне моделювання вирішення задачі досягнення мети при русі автопоїзда за наявності перешкод. Наукові праці ДонНТУ, серія "Інформатика, кібернетика та обчислювальна техніка". 2018. № 2(27) . С. 53–63.
15. Старшеков Данило, студент магістратури ФЕЕІТ ІІ ЗНУ. Наук. кер.: д-р.ф.-м.н., проф. Вербицький В.Г. « МОДЕЛЮВАННЯ РУХУ КОЛІСНИХ РОБОТІВ НА ПЛАТФОРМІ UNITY». Збірник наукових праць студентів, аспірантів і молодих вчених «Молода наука-2020» : у 5 т. Запорізький національний університет. Запоріжжя: ЗНУ, 2020. Т.5. С. 168.
16. Фрэд Лонг. Руководство для программиста на Java: 75 рекомендаций по написанию надежных и защищенных программ. 2011. 71 с.
17. Кей С., Хорстманн, Гари Корнелл. Java. Библиотека профессионала, том 1. Основы. 9-е издание. 2008. 114 с.
18. James Gosling, Bill Joy, Guy Steele, Gilad Bracha. The Java Language Specification, Third Edition.
19. Жемеров Д., Исакова С. Kotlin в действии. ДМК-Пресс, 2017. 402 с.
20. Молчанова Л. А. Введение в Maple. Учебно-методическое пособие. Издательство Дальневост. ун-та. 2006. 36 с.
21. Слабуха Николай. Введение в Robot Operation System. 2018. 351 с.
22. Introduction to SignalR. URL: https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr (дата звернення 10.12.2020)
23. Хокинг Дж. Unity в действии. Мультиплатформенная разоаботка на C#. Питер. 2016. 336 с.

24. V. I. Kalenova, A. V. Karapetjan, V. M. Morozov, M. A. Salmina. Nonholonomic mechanical systems and stabilization of motion. 2005. – Vol. 11, No 7. pp. 117-158.

25.Entity Framework Tutorial. URL: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx (дата звернення 10.12.2020)

26. Документація C#. URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата звернення 8.12.2020)

27. Патерн стратегія. URL: https://refactoring.guru/uk/design-patterns/strategy (дата звернення 1.12.2020)

28.Патерн ітератор. URL: https://refactoring.guru/pl/design-patterns/iterator (дата звернення 1.12.2020)

29. Основы работы с Robotic Operating System. URL: https://habr.com/ru/post/128024/ (дата звернення 5.11.2020)

30. Опыт использования WebRTC. Лекция Яндекса. URL: https://habr.com/ru/company/yandex/blog/419951/ (дата звернення 1.10.2020)

31. Серверный WebRTC в 2020 году — обзор возможностей. URL: https://habr.com/ru/post/512496/ (дата звернення 16.10.2020)

32. Эффективное управление подключениями SignalR. URL: https://habr.com/ru/post/470299/ (дата звернення 09.9.2020)

33. Практическое руководство по использованию Hilt с Kotlin. URL: https://habr.com/ru/company/otus/blog/532066/ (дата звернення 26.9.2020)

34. Корректный ASP.NET Core. URL: https://habr.com/ru/post/437002/ (дата звернення 21.10.2020)

**Декларація**

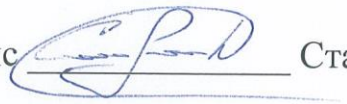
**академічної доброчесності**

**здобувача ступеня вищої освіти ЗНУ**

Я, Старшеков Данило Анатолійович, студент 2 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти sp115-28@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Комп'ютерна система керування та стабілізації руху колісних роботів**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата 30.11.2020 Підпис  Старшеков Данило Анатолійович

(студент)

Дата 30.11.2020 Підпис  Вербицький Володимир Григорович

(науковий керівник)