МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДОДАТКУ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ «УСТАНОВКА ОСУШКИ ПРИРОДНОГО ГАЗУ». МЕРЕЖЕВЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДОДАТКУ

Здобувач освіти гр. ІН-81 Софія ГОНЧАРЕНКО

Науковий керівник,
кандидат фізико-математичних наук,
доцент кафедри комп’ютерних наук Сергій ШАПОВАЛОВ

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

Суми 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2022 р.

ЗАВДАННЯ
до випускної роботи

Студентки четвертого курсу, групи ІН-81 спеціальності 122 -Комп'ютерні науки, денної форми навчання Гончаренко Софії Андріївни.

Тема: Інформаційне та програмне забезпечення додатку віртуальної реальності «Установка осушки природного газу». Мережеве програмне забезпечення додатку.

Затверджена наказом по СумДУ

№_____ от _____2022 р.

Зміст пояснювальної записки: 1) огляд предметної області; 2) огляд подібних рішень; 3) актуальність налаштування мережевого доступу; 4) постановка задачі; 5) вибір програмної реалізації серверу авторизації та бібліотеки мережевого супроводу для Unity; 6) практична реалізація проекту.

Дата видачі завдання "_____"_____2022 р.

Керівник випускної роботи _____ Шаповалов С.П.

Завдання прийняла до виконання _____ Гончаренко С.А.

РЕФЕРАТ

Записка: 65 стор, 14 рис, 2 таблиці, 3 додатки, 23 джерел.

Об’єкт дослідження – Застосування технологій VR в дистанційному освітньому середовищі.

Мета роботи – створення додатку VR «Установка осушки природного газу» та розробка мережевого програмного забезпечення для використання в процесі віртуальної освіти.

Методи дослідження – дослідження аудиторії та порівняння сервісів аналогів.

Результати – розроблено мережеве програмне забезпечення, що відтворює у віртуальній реальності досвід роботи оператора станції осушки природнього газу. Використання мережевих технологій дає можливість викладачам та студентам створювати навчальні групи дистанційно та проводити заняття на відтвореній реальній ситуації. Мережеві налаштування розроблені на Nest.js з використанням Node.js, розробка кімнат та синхронізація рухів налаштована з допомогою Photon Pun.

ВІРТУАЛЬНА РЕАЛЬНІСТЬ, VR ОСВІТА,
VR ДОДАТОК, ДИСТАНЦІЙНЕ НАВЧАННЯ, МЕРЕЖЕВЕ
ЗАБЕЗПЕЧЕННЯ ДОДАТКУ

ЗМІСТ

ВСТУП

Протягом XX століття розвиток техніки набув надзвичайно стрімкості: зросли складність, швидкість, функціональність, точність. З’явилися перші комп’ютери, а разом з ними у наше життя увійшли інформаційні технології і віртуальна реальність (VR).

Віртуальна реальність виявилася тим новим і незвичайно дивовижним світом, який наразі має суперечливий характер: когось VR приваблює, когось лякає, а хтось так глибоко поринув у віртуальний простір, що не може вийти у реальний світ. Використання віртуальної реальності відносять також і до найсучасніших технологій у науці та освіті. У навчальному процесі завдяки VR може здійснюватися моделювання складної чи небезпечної діяльності. У науці VR дозволяє візуалізувати внутрішню будову об’єктів, молекулярних та атомних структур. Використовуючи технології VR, можна створювати проекти у різних освітніх напрямах.

Відповідно до зазначеного, метою дипломної роботи є створення додатку VR «Установка осушки природного газу» та розробка мережевого програмного забезпечення для використання в процесі віртуальної освіти.

Для досягнення поставленої мети необхідно виконати такі задачі:

- провести аналіз предметної області та визначити актуальність дослідження;

- розробити сервер авторизації для забезпечення розділення ролей користувачів та їх ідентифікації.

- додати систему кімнат для одночасного проведення занять декількома групами.

- розробити синхронізацію дій клієнтських додатків.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд предметної області

Дистанційна та змішана форми навчання мають доволі тривалу історію у сфері освіти. Вони з’явилися набагато раніше, аніж люди почали використовувати комп’ютери та інформаційні технології. Раніше студенти різних університетів, які навчалися заочно, мали можливість здобувати вищу освіту засобами так званого «кореспондентського навчання», в межах якого взаємодія викладача зі слухачами відбувалася через листування – поштою надсилалися різні матеріали та завдання. Проте можливості заочної форми навчання були дуже обмежені.

Саме поява і поширення Інтернет-мережі детермінувало розширення меж не тільки заочного, а й очного навчання, зробивши його повноцінним і всеохоплюючим. З винайденням різноманітного технічного обладнання і невпинним розвитком Інтернет-технологій навчальний процес значно полегшився і набув широкого розповсюдження по всьому світові, в тому числі і в Україні. Зокрема, у 2004-2006 роках постановою Кабінету Міністрів України була прийнята і затверджена Програма розвитку системи дистанційного навчання, в якій зазначалося про необхідність прискорення реформування вітчизняної системи освіти в контексті світового процесу переходу до постіндустріального та інформаційного суспільства, а також економічних, політичних і соціальних змін в самій Україні. Саме дистанційне навчання, за задумом законотворців, мало сприяти найбільш ефективному розв’язанню проблем, адже зазначена форма «здійснюється на основі сучасних педагогічних, інформаційних та телекомунікаційних технологій» [1].

Проте, ще протягом деякого часу над дистанційною освітою превалювала змішана форма навчання, яка мала вигляд педагогічно виваженого поєднання технологій традиційного, електронного, дистанційного і мобільного навчання, спрямованого на інтеграцію аудиторного та позааудиторного навчання. Одним із завдань змішаної форми навчання було розв'язання проблем дистанційної освіти. Саме змішані підходи до навчання виявилися на той час одними з найбільш популярних технологій, адже дозволяли скористатися гнучкістю і зручністю дистанційного курсу та перевагами традиційного класу [2].

Пандемія коронавірусу і введення вимушених карантинних заходів кардинально змінили увесь світ. Відтепер надзвичайно важливим в контексті збереження життя і здоров'я населення планети стало зведення до мінімуму контактів між людьми, що передбачало перехід на віддалену роботу, спілкування і навчання на безпечній соціальній відстані. Саме після 2020 року майже усі учбові заклади повністю переробили свої навчальні плани під карантинні обмеження і перевели освітній процес у дистанційну форму [3].

Зараз під дистанційним навчанням розуміють сукупність інформаційних технологій та методів навчання, які можуть забезпечити освіту під час перебування учнів поза школою. Відповідно до Положення про дистанційне навчання (2013 р.), його метою є «надання освітніх послуг із застосуванням сучасних інформаційно-комунікаційних технологій на певному рівні освіти або освіти відповідно до національних освітніх стандартів» [4].

Для вирішення питань, пов'язаних з переходом від очного до заочного навчання, були залучені і спеціально створені різноманітні інноваційні навчальні засоби і методики. Теоретичний матеріал перенесено викладачами в презентації, відеоролики та інтерактивні квізи; практичний матеріал

відпрацьовується учнями і студентами в онлайн форматі під час відеоконференцій, тестувань тощо.

Терміни «віртуальний», «віртуальність» є похідними від латинського слова «virtualis», що означає процес, який відбувається або може відбутися за певних умов, або процес, якого не існує, але він може бути реалізований [5]. Віртуальна реальність (з лат. virtus – потенційний, можливий і лат. realis – дійсний, існуючий; англ. virtual reality) – це створений технічними засобами світ, що передається людині (відвідувачу цього світу) через її органи чуття: зір, слух, нюх тощо. Тобто віртуальна реальність складається лише з нереальних, створених комп'ютером об'єктів; вона імітує дію і реакції на дію.

Варто зазначити, що поведінка об'єктів віртуальної реальності часто близька до поведінки об'єктів фізичної реальності. Так, користувачі VR можуть впливати на ці об'єкти за реальними законами фізики. Незвичайним і цікавим аспектом використання віртуальної реальності є те, що користувачам дозволено більше, ніж нам у реальному житті [6].

За відносно короткий проміжок часу поняття «віртуальна» та «віртуальна реальність» набули міждисциплінарного статусу. Сьогодні вони використовуються не тільки в квантовій механіці, інформатиці, ергономіці, а й у психології, педагогіці, медицині, соціології. Технологія VR також широко використовується в різних сферах людської діяльності: проектуванні та інженерії, гірничодобувній та військовій техніці, будівництві, тренажерах і тренажерах, маркетингу та рекламі, індустрії розваг тощо [7].

Не менш широкого використання VR технології набули в освіті та науці. VR створює ціле цифрове середовище, 360-градусний імерсивний досвід користувача, який здається реальним. В умовах віртуальної реальності учні і студенти можуть взаємодіяти з тим, що вони бачать, якби вони справді

були там. Використання в освіті VR технологій відкриває перед учнівською і студентською аудиторією безліч можливостей:

- у віртуальному просторі можна легко побачити деталі будь-якого процесу чи об'єкта, що набагато цікавіше, ніж розглядати картинки в підручниках (наприклад, вивчення будови тіла, знайомство з підводним світом, занурення у вулкан) ;

- у віртуальному середовищі ніщо не відволікає від зовнішніх подразників, що дозволяє повністю сконцентруватися на навчальному матеріалі;

- технологія віртуальної реальності дає можливість повністю контролювати та змінювати сцену подій (наприклад, учні можуть бути свідками історичних подій, проводити фізичні чи хімічні експерименти, розв’язувати задачі в ігровій та зрозумілій формі тощо);

- за допомогою технології VR ви можете виконувати складні операції, керувати спортивними автомобілями і навіть космічними кораблями, а також експериментувати з небезпечними хімічними речовинами, не завдаючи шкоди собі або навколишньому середовищу;

- використання віртуальної реальності в освіті забезпечує зв’язок між досвідом і навчанням;

- завдяки VR в освіті студенти та студенти надихаються відкривати, що можна зробити помилки, які нікому не зашкодять, тому що все відбувається в окремих сценаріях;

- студенти мають можливість навчатися на практиці, а не пасивно читаючи та запам'ятовуючи [8].

Таким чином, сучасні виклики і карантинні обмеження на тлі стрімкого розвитку науки і техніки призвели до віртуалізації майже всіх сфер суспільного життя людини. Віртуальний простір докорінно змінив людський

спосіб життя, створивши натомість, з одного боку, нове штучне середовище; з іншого, надав безліч можливостей для найрізноманітніших проявів і діяльності людей. У віртуальній реальності можна робити щось багаторазово і не боятися помилитися (саме тому надзвичайно високою є ефективність тренажерів, заснованих на використанні VR технологій). Не менш важливим є те, що зниження цінності об’єктів віртуального світу, які є лише ілюзією, спонукає і мотивує людей творити, пробувати, досліджувати. Нарешті використання VR технологій має велике значення в пізнанні, освоєнні і формуванні сучасного освітнього простору. Перенесення знань, умінь і навичок чи нового досвіду з однієї реальності в іншу може відігравати значну роль у розвиткові людини.

1.2 Огляд подібних рішень

На ринку представлено досить багато VR проектів та рішень, які були створені для освітніх цілей. Усі вони мають різний функціонал, для того, щоб виконувати завдання, для яких були створені. Наприклад, є додатки, які призначені лише для огляду, спостерігання, в інших учні та студенти можуть виконувати якісь дії. Більшість працюють на основі мережевих технологій.

Для наочності було розглянуто три проекти, як уможливили розуміння принципів роботи та отримання прикладу для здійснення власного дослідження.

Застосунок для вивчення анатомії людини 3D Organon VR Anatomy [9]. Вивчення цього додатку показало, що це повнофункціональний атлас з анатомії у віртуальнії реальності, який містить більше, ніж 400 реалістичних анатомічних моделей і структур, описів і деталізацій.

Використовуючи його, учень отримує можливість не тільки роздивлятися окремо або в комплексі частини людського тіла, а також він може «здійснювати» операційні втручання на моделях (рис. 1.1), виконувати різні завдання у різних режимах, створювати квізи, задавати різні параметри навчання тощо.

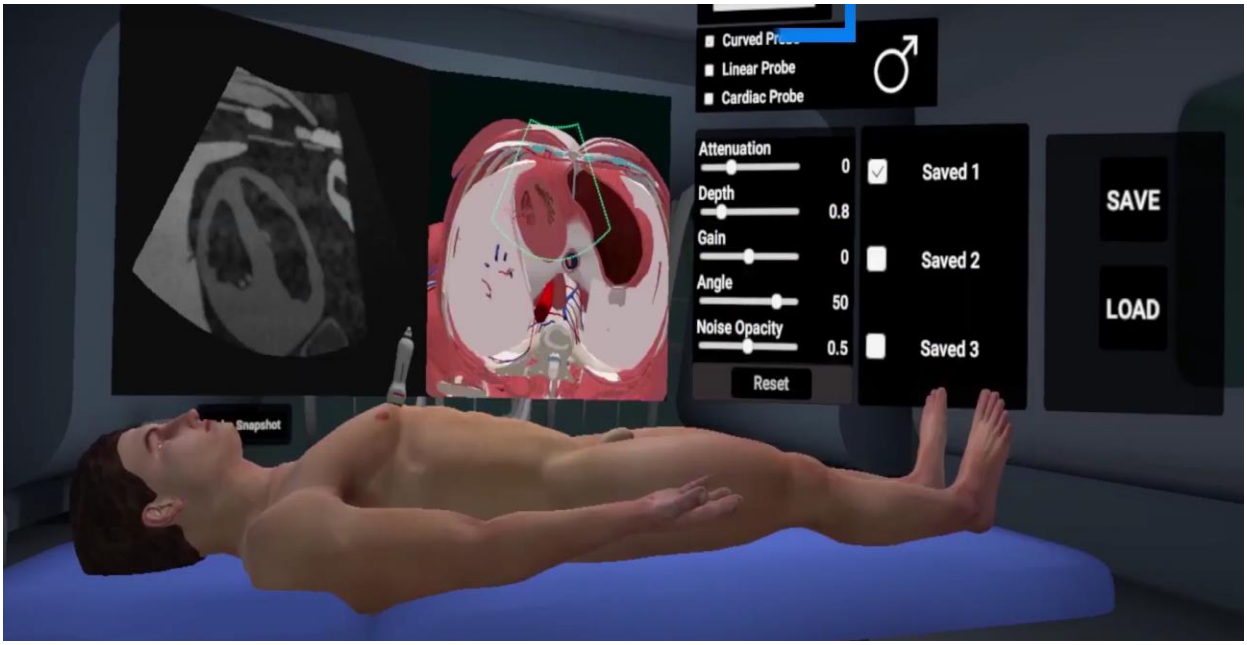

Рисунок 1.1 – Проведення операції на макеті

Саме взаємодія з об’єктами в процесі проходження сценаріїв (приміром, проведення операцій) є аналогом для нашого проєкту – можливість працювати з газовою установкою, створювати ситуації та мати змогу вручну робити налаштування. Також учасник може виступати як учнем, який проходить завдання, так і викладачем, який задає сценарії, слідкує за проходженням (рис. 1.2). Студентів може біти як один, так і декілька, модератор має можливість котролювати усіх. Крім того, додаток має 16 мов

для налаштування та можливість користування через VR окуляри та як 3Д модель.

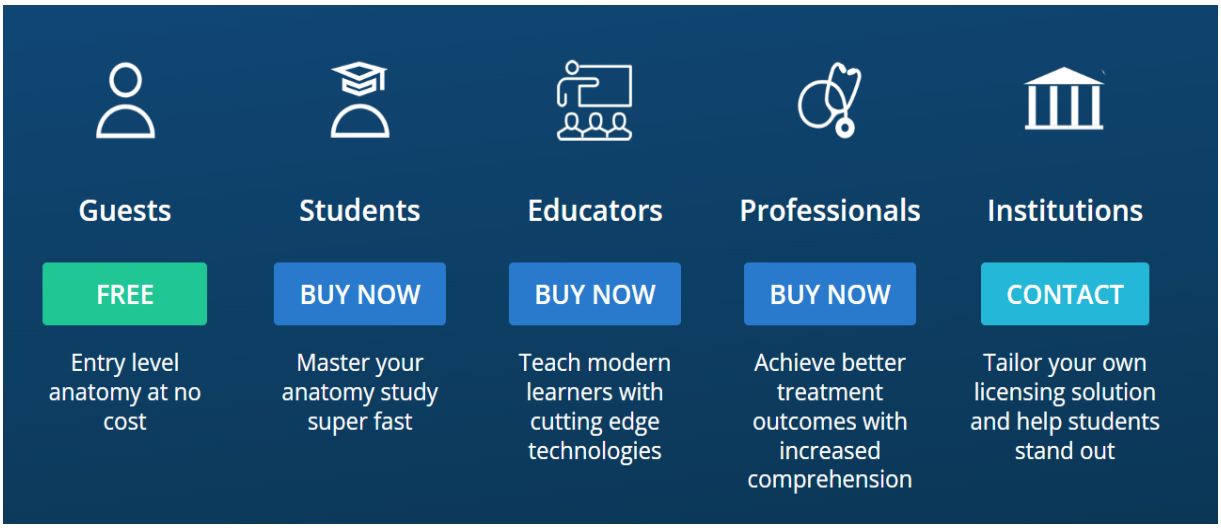

Рисунок 1.2 – Ролі використання додатку

Ще одним корисним веб-проектом є Mix sumdu (рис. 1.3) [10]. Це університетська освітня платформа, яка розроблена на основі мережевих технологій. Окрім того, викладач та студент мають різні права доступу, що дозволяє розмежувати можливості цих ролей, їм не потрібно локально перебувати в одній аудиторії або десь поряд. Мережеві технології дозволяють працювати у будь-якій точці світу і мати доступ до своїх завдань.

Третім проектом, який ми розглянули в якості прикладу, є 4D Interactive Anatomy (рис. 1.4) [11]. Як і перший варіант додатку, це атлас з анатомії, який дозволяє розглядати «вручну» людське тіло. Можливості студентів та викладачів відрізняються, також присутня можливість віддаленої роботи учасників сцен.

Студент має можливість проходити більш, ніж 2000 питань у вікторинах; шукати і досліджувати понад 2000 анатомічних структур; повертати, видаляти шари і виділяти структури у 3D; мати доступ до навчання

у будь-який час з будь-якого екрану. Викладачі можуть створювати власні тести та експортувати результати; користуватися функціоналом для експорту зображень для використання у навчальних заняттях, презентаціях, LMS і статтях; здійснювати роботу в електронному журналі для відстеження оцінок учнів.

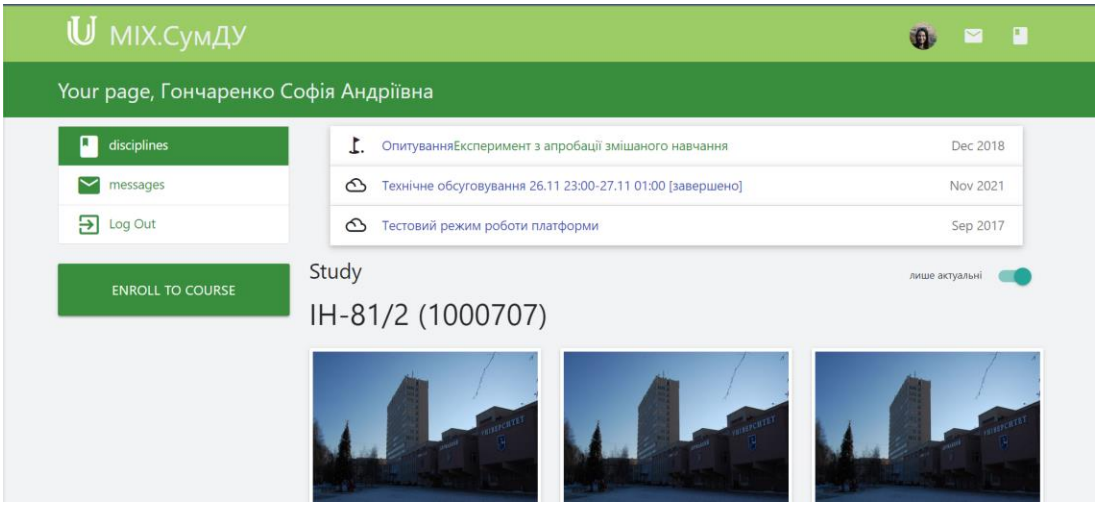

Рисунок 1.3 – Платформа Mix sumdu

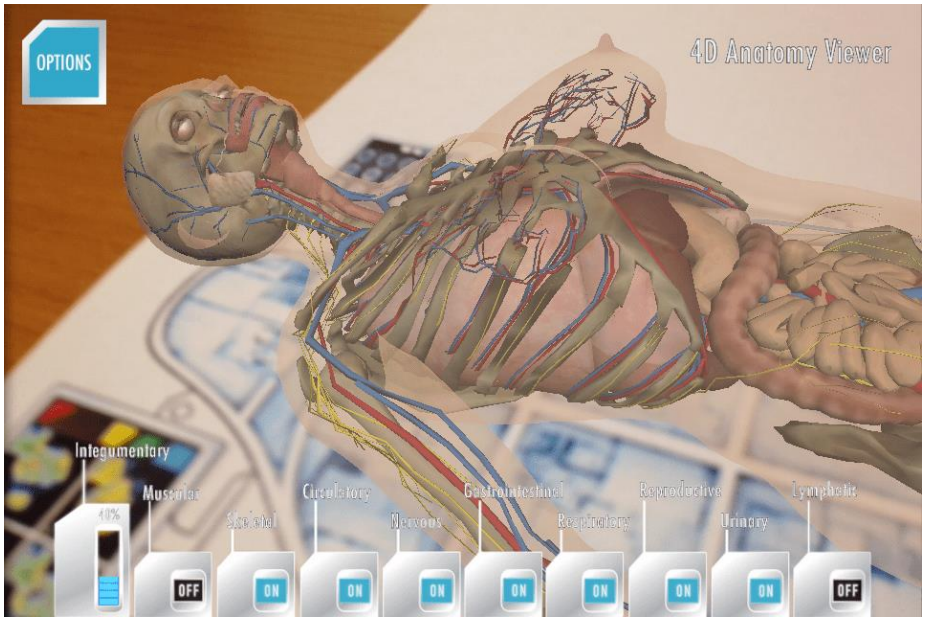

Рисунок 1.4 – 4D Interactive Anatomy

1.3 Актуальність налаштування мережевого доступу

Віртуальна реальність здатна здійснити революцію в освіті, але через високу вартість гарнітур її використання в звичайних класах може бути обмеженим. Проте, варто визнати, що загалом в освіті, яка наразі переходить до більшої кількості онлайн-налаштувань, VR має можливість стати інноваційним доповненням до будь-якого уроку чи практичного / лабораторного заняття. VR Education робить процес навчання цікавим, безпечним і більш привабливим, аніж коли-небудь раніше.

Додавання мережевих сервісів для VR додатку дозволяє збільшити ефективність освіти, об’єднати студентів та викладачів з різних куточків світу в одному віртуальному інформаційному просторі. Також це дозволяє не прив’язуватися до конкретного місця, відвідувати заняття, знаходячись вдома, на роботі тощо. Не треба щоденно їздити до закладу освіти та назад додому, що особливо критично для великих міст та студентів з області, які у дорозі можуть «витрачати» по декілька годин.

Мультиплеєр дозволяє контролювати процес навчання викладачем, який може працювати одночасно з групою студентів. Таким чином підвищується дисципліна та формується атмосфера співпраці між учасниками заняття, яка потрібна для створення соціальної взаємодії.

1.4 Постановка задачі

Необхідно розробити додаток для симуляції роботи установки по очищенню природного газу з використанням можливостей віртуальних технологій.

Для цього необхідно виконати такі **завдання**:

1. Розробити сервер авторизації для забезпечення розділення ролей користувачів та їх ідентифікації.

2. Додати систему кімнат для одночасного проведення занять декількома групами.

3. Розробити синхронізацію дій клієнтських додатків.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Вибір програмної реалізації серверу авторизації

Авторизація – це функція визначення прав/привілеїв користувача або його доступу до ресурсів, що пов’язано із загальною інформаційною безпекою та безпекою комп’ютера. Перед тим як авторизуватися необхідно пройти аутентифікацію, яка ставить на меті перевірку відповідності введеного користувачем логіну та пароля до пароля облікового запису в базі даних [12].

Розробка авторизації, аутентифікації та серверу, який за це відповідає, є важливою частиною роботи над багатьма додатками.

Для створення власного веб-серверу насьогодні існує безліч інструментів. Серед них: Spring Framework в Java, ASP.NET, Node.js та інші. Саме останню технологію ми обрали для розробки нашого продукту, і ось чому [13]:

- кодування в Node.js відносно легко зрозуміти, якщо ви освоїли основи JavaScript та об'єктно-орієнтованого програмування. Найскладніше – мати чітке уявлення про модель клієнт-сервер.

- свій проект легко запустити із відносно невеликими витратами на серверну інфраструктуру.

- Node.js добре підходить для архітектури мікросервісів, що дійсно корисно для проектів, які масштабуватимуться і зростатимуть у майбутньому. Також є можливість створити окремий мікросервіс під будь-який функціонал, а потім масштабувати його окремо від інших частин.

- Node.js дозволяє швидко розробити MVP (мінімально життєздатний продукт) - частина програмного забезпечення з достатньою кількістю функцій

(або однією функцією забій), щоб продукт міг вийти на ринок і задовольнити перших клієнтів. MVP це фундаментальний етап на шляху до повноцінного додатку.

- активне співтовариство, що означає велику підтримку та зворотний зв'язок. Більш того, у величезній спільноті є безліч розробників та власників продуктів, які можуть допомогти вам з більшістю ваших питань.

- велика кількість бібліотек. NPM (менеджер пакетів вузла), екосистема пакетів Node, є найбільшим і найшвидше реєстром програмного забезпечення у світі. Він надає безліч бібліотек і повторно використовуваних шаблонів, які ви можете включити до свого коду, щоб отримати більше користі з меншими зусиллями та часом.

З усіма своїми перевагами Node.js відіграє вирішальну роль у стеку технологій багатьох відомих компаній, які залежать від його унікальних переваг, саме тому ми обрали його для нашого проекту.

Для покращення досвіду роботи із програмним кодом, та запобіганню «випадкових» помилок під час розробки ми будемо також використовувати TypeScript в основі нашого додатку.

TypeScript - це строга синтаксична надмножина JavaScript, що додає до мови необов'язкову статичну типізацію. TypeScript призначений для розробки великих додатків та транспілюється у JavaScript [14].

Для написання веб-сервісів, які будуть відповідати за взаємодію клієнтської та серверної частини будемо використовувати фреймворк Nest.js. Саме він в свої основі використовує TypeScript і включає функції об'єктно-орієнтованого, функціонального і функціонального реактивного програмування. NestJS використовує потужні фреймворки HTTP-серверів, такі як Express та Fastify, які забезпечують певний ступінь абстракції порівняно з основними фреймворками Node.js. Але вони також надають API

безпосередньо розробнику, дозволяючи розробникам використовувати безліч сторонніх модулів, доступних для базової платформи [16].

Крім цього, необхідно подбати про роботу з базами даних. Для цього будемо використовувати Knex – узагальнений інтерфейс для роботи з різними базами даних, який надає доступу до бази даних та виконання дій з нею та її даними. Він підтримує багато систем управління базами даних, такі як MySQL, SQLite, Postgres та інші. З Knex.js можна зберігати всі конфігурації для різних середовищ в одному файлі, використовувати методи бібліотеки для виконання дій з базою даних. Отже, незалежно від того, яку систему управління базами даних ми використовуємо для різних середовищ, ми можемо застосовувати ті самі методи.

2.2 Паттерни

В розробці було використано два паттерни: паттерн репозиторію та паттерн впровадження залежностей (Dependency injection).

Шаблон Controller-Service-Repository (так званий паттерн репозиторій) переважає в багатьох веб-додатках [17]. Одна з важливих причин, чому цей шаблон зручний, полягає в тому, що він відмінно справляється з поділом завдань: рівень контролера у верхній частині рисунку 2.1 несе виняткову відповідальність за надання функціональності, щоб її могли використовувати зовнішні об'єкти. Шар репозиторію знизу цього зображення відповідає за зберігання та вилучення деякого набору даних. На сервісному рівні має бути вся бізнес-логіка. Якщо бізнес-логіка вимагає отримання/збереження даних, вона підключається до репозиторію. Якщо хтось хоче отримати доступ до цієї бізнес-логіки, він проходить контролер, щоб дістатися туди.

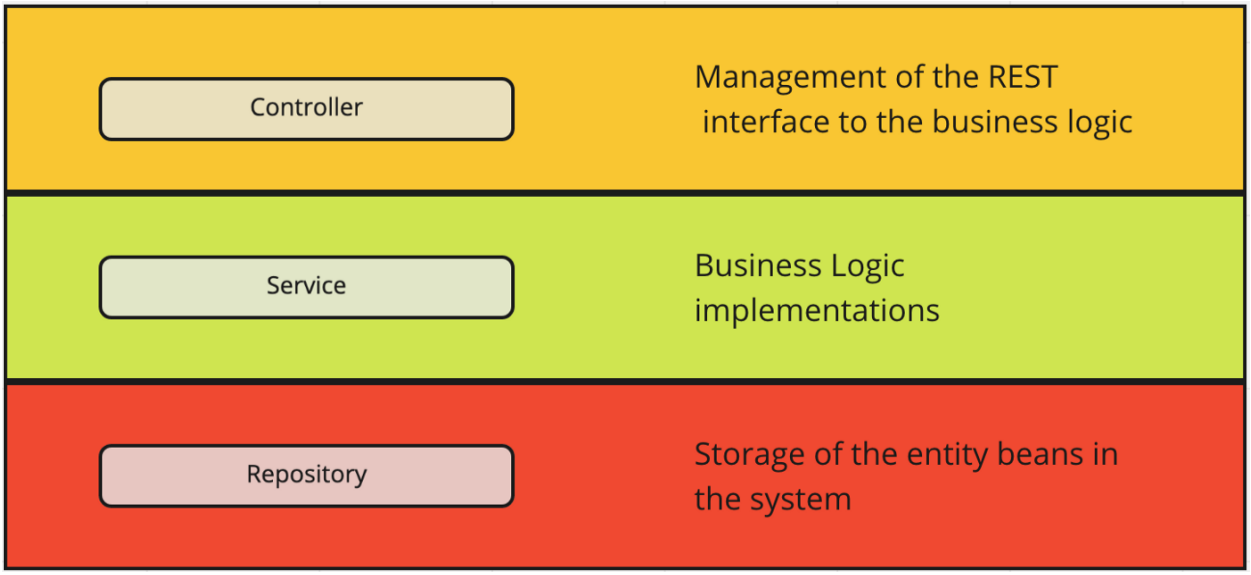

Рисунок 2.1 – Паттерн репозиторій

Якщо код пов'язаний із зберіганням/витягом, він повинен бути поміщений до репозиторію. Якщо йдеться про розкриття функціональності, це відбувається у контролері. Все унікальне в бізнес-логіці буде на рівні сервісу. Репозиторію байдуже, який компонент його викликає; він сліпо робить те, що його просять. Рівень сервісу не піклується про те, як отримати доступ до нього, він просто виконує свою роботу, використовуючи репозиторій, де це необхідно. Контролер просто передає роботу сервісному шару, тому він може залишатися красивим і компактним.

Другий паттерн – використання залежностей (DI) – це шаблон проектування, у якому об'єкт отримує інші об'єкти, від яких він залежить. Форма інверсії управління, впровадження залежностей спрямовані на поділ проблем створення об'єктів та їх використання, що призводить до слабозв'язаних програм. Шаблон гарантує, що об'єкт, який хоче використовувати цю службу, не повинен знати, як створювати ці служби.

Натомість приймаючий об'єкт (або «клієнт») отримує свої залежності від зовнішнього коду («інжектора»), про який він не знає.

Основною перевагою dependency injection є зменшення зв'язку між класами та їх залежностями. Позбувши клієнта знання того, як реалізуються його залежності, програми стають більш придатними для повторного використання, тестування і супроводу. Це також призводить до підвищеної гнучкості: клієнт може діяти на все, що підтримує внутрішній інтерфейс, який очікує клієнт.У загальному плані використання залежностей скорочує обсяг шаблонного коду, оскільки створення залежностей обробляється одним компонентом [18].

2.2 Вибір бібліотеки мережевого супроводу для Unity

В реаліях сьогоднішнього різноманіття програмних продуктів та поставлених задач не існує універсального для впровадження мережевого функціоналу у середовище Unity. Швдкий та динамічний геймплей буде мати зовсім інші запити до мережевого коду аніж додаток з покроковою взаємодією. На жаль, єдиного рішення не існує. Розробникам неоюхідно оцінювати різні варіанти та обрати яке рішення або їх комбінація найкраще підійдуть для реалізації ідеї.

Для реалізації наших завдань було обрано Photon Pun. Вся мережева структура покладена на цей сервіс. Таким чином не треба перейматися за налаштування власного виділеного серверу, його налаштуванням та захистом його від різного виду мережевих атак. Цей плагін дозволяє просто налаштувати з’єднання між двома клієнтами, синхронізувати їх переміщення та створювати спільні події. Останні і є основним принципом роботи даної

системи. Сервер приймає подію від клієнта, за необхідності оброблює її та надсилає всім іншим клієнтам, які знаходиться в одному лобі.

Дане рішення є безкоштовним за умови одночасного підключення менше ста користувачів. В результаті вище описаних систем отримуємо модель взаємодії необхідного ПЗ.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Сервер авторизації

Розпочнемо роботу з розробки серверу авторизації. Його будемо реалізовувати за допомогою фреймворку Nest.js у середовищі Visual Studio Code, тому що даний редактор коду є зручним у використанні і ми можемо налаштувати його для підсвітки змінних та перевірки помилок.

Для початку необхідно створити проект. Для цього скористаємося командою ***nest new <project-name>***. Автоматично підвантажиться шаблон проекту та усі необхідні залежності.

Для повноцінної роботи нашого серверу необхідно створити базу даних, в якості якої використаємо MariaDB. В ній буде міститися дві таблиці: tb_users з полями PK_userId: number; FK_PK_roleId: number; login: string; hashPassword: string; name: string; – та tb_roles з полями PK_roleId: number; roleName: string.

Пароль до нашої БД будемо зберігати у хешованому вигляді для запобігання несанкціонованого доступу до даних користувача.

Далі додаємо потрібні бібліотеки, які знадобляться нам у майбутньому:

- class validator для автоматичної валідації вхідних запитів;
- nestjs, knex, mysql2 для роботи з базами даних;
- cache-manager для зберігання даних у кеші;
- crypto для хешування даних.

Для початку створимо файл змінних оточення (environment variables), де ми розмістимо всі змінні, які будемо використовувати в рамках нашої роботи:

```
DB_DRIVER=mysql2
DB_HOST=localhost
DB_PORT=3306
DATABASE_USERNAME=root
DATABASE_SCHEMA=vr
JWT_SECRET=;2qQq^`<"N^5p7wE
JWT_EXPIRED_IN=12h
JWT_REFRESH_EXPIRED_IN=7d
```

Відповідно, тут ми вказуємо параметри підключення до бази даних, а також властивості майбутніх jwt токенів.

Наступним етапом налаштовуємо з’єднання з базою даних. Для цього створюємо файл конфігурації knex-option.factory:

```
import { KnexOptions, KnexOptionsFactory } from '@nestjsplus/knex';

export class KnexOptionsFactoryImpl implements KnexOptionsFactory {
    createKnexOptions(): KnexOptions {
        return {
            client: process.env.DB_DRIVER,
            debug: false,
            connection: {
              host: process.env.DB_HOST,
              user: process.env.DATABASE_USERNAME,
              database: process.env.DATABASE_SCHEMA,
              port: +process.env.DB_PORT,
            },
          };
    }
}
```

Створюємо клас, який наслідується від KnexOptions та перевизначає функцію createKnexOptions(), де налаштовуємо під’єднання до БД, використовуючи параметри, отримані зі змінних оточення.

Наостанок, для повноцінної роботи сервісу knex, включаємо в основному файлі app.module в якості імпорту модуль KnexModule, реєструючи в ньому створені раніше налаштування :

```
KnexModule.registerAsync({
      useClass: KnexOptionsFactoryImpl,
    }),
```

Так як ми слідували методології Controller-Service-Repository, необхідно створити інтерфейси та їх реалізації, а також контролер для процесу авторизації. Далі будуємо необхідні нам запити.

Для авторизації у нашому додатку ми будемо використовувати систему refresh та access (auth) токенів. Таким чином, максимально убезпечимо користувачів від шахраїв, які будуть намагатися отримати доступ до даних. Access токен має відносно короткий термін існування та зберігає в собі інформацію про користувача, його привілеї тощо. Даний токен є багаторазовим та використовується для доступу до API. Refresh токен має, у свою чергу, довший термін життя та використовується для отримання нової пари access/refresh токенів. Він є одноразовим. Тож, якщо злоумисник отримав доступ до access токенів, він може виконувати запити на той термін, що був у викраденого токену і поновити його знов не зможе.

Для реалізації обраного нами механізму авторизації в нашому сервері необхідно створити наступні кінцеві точки:

- Post auth/login потрібний для авторизації по логіну та паролю і видачі первинної пари refresh та access токенів. В якості тіла запиту приймає JSON файл із двома полями login та password типу string.

- Post auth/refresh використовується для поновлення даних та отримання нової пари токенів. Для даного запиту необхідно передати у тілі значення користувацького refresh токену.

Відповідно кінцевий вигляд нашого контролеру матимете такий вигляд:

```
@Controller('auth')
export class AuthController {
  constructor(@Inject(AUTH_SERVICE_TOKEN) private readonly
authService:IAuthService) {}

  @Post('/login')
  @HttpCode(200)
  public async login(
    @Body() userCredential: UserCredentialsForm,
  ): Promise<any> {
    return await this.authService.login(userCredential.username,
userCredential.password);
  }

  @Post('/refresh')
  @HttpCode(200)
  public async refresh(
    @Body() refreshForm: RefreshForm,
  ): Promise<any> {
    return await
this.authService.refreshAccessToken(refreshForm.refreshToken);
  }
}
```

Наступним завданням є створення сервісів, які будуть використовуватися у контролері. Нам необхідні два методи login та refreshAccessToken. Створюємо інтерфейс, який описуватиме ці два методи:

```
export interface IAuthService{
    login(login: string, password: string): Promise<any>;

    refreshAccessToken(refreshToken: string): Promise<any>;
}
```

Робимо відповідний сервіс, що реалізує створений інтерфейс. Розглянемо роботу методів детальніше. Метод login представлений нижче:

```
public async login(login: string, password: string): Promise<any> {
    const userData = await this.authRepository.checkAuth(
      login,
      hashPassword(password),
    );
    if (!userData) {
```

```
      throw new BadRequestException('Invalid login or password');
    }
    return {
      data: { role: userData.FK_PK_roleId, userName: userData.name },
      accessToken: await this.generateAccessToken(userData),
      refreshToken: await this.generateRefreshToken(userData),
    };
  }
```

Алгоритм роботи даного методу такий: знайти по парі логін/хеш паролю інформацію про користувача у БД через репозиторій. Так як паролі зберігаються у захешованому вигляді, перед тим як шукати відповідність у БД, необхідно виконати деяку обробку. У нашому вигляді пароль хешується п’ять разів з алгоритму sha256:

```
export const hashPassword = (password) => {
  let result = password;
  for (let i = 0; i < 5; i++) {
    result = crypto.createHash('sha256').update(result,
'utf8').digest('hex');
  }
  return result;
};
```

Якщо відповідність знайдена, то генеруємо access токен, в якому зберігаємо параметри користувача, refresh токен, тримаючи у ньому ідентифікатор (його зберігаємо у кеші як ключ до userID для того, щоб зробити його одноразовим) та повертаємо дані контролеру.

Розглядаючи метод refresh/access токен, першим кроком проводиться валідація refresh токену. Якщо він відповідає усім вимогами, ми можемо використати його ідентифікатор для отримання userId з кешу пам’яті. Таким чином, можемо безперешкодно поновити інформацію про користувача з БД На основі неї, відповідно до методології, генеруємо нову пару refresh-access токенів:

```
public async refreshAccessToken(refreshToken: string): Promise<any> {
    const decodedToken = await this.jwtService.verifyAsync(refreshToken);
    const userId = await this.cacheManager.get(decodedToken.refreshId);
    if(!userId){
      throw new UnauthorizedException;
    }
    const userData = await this.authRepository.getUserDetailsById(userId);
    return {
      data: { role: userData.FK_PK_roleId, userName: userData.name },
      accessToken: await this.generateAccessToken(userData),
      refreshToken: await this.generateRefreshToken(userData),
    };
  }
```

Як ми бачимо, наші сервіси використовують доступ до БД через репозиторій, а саме отримання інформації про користувача через логін та хеш паролю або ж через userId. Тому необхідно створити інтерфейс майбутнього репозиторію з потрібними методами:

```
export interface IAuthRepository {
  checkAuth(login: string, hashPassword: string): Promise<UserDTO>;

  getUserDetailsById(userId: number): Promise<UserDTO>;
}
```

Відповідно треба створити реалізацію даного інтерфейсу:

```
export class AuthRepository implements IAuthRepository {
  constructor(@Inject(KNEX_CONNECTION) private readonly knex: Knex) {}

  public async getUserDetailsById(userId: number): Promise<UserDTO> {
    return this.knex('tb_users').select('*').where('PK_userId',
userId).first();
  }

  public async checkAuth(
    login: string,
    hashPassword: string,
  ): Promise<UserDTO> {
    return this.knex('tb_users')
      .select('*')
      .where('login', login)
      .andWhere('hashPassword', hashPassword)
```

```
        .first();
    }
}
```

Тут за допомогою knex робимо запити до БД для отримання необхідної нам інформації.

3.2 Мережевий супровід додатку віртуальної реальності

Для побудови певної поведінки системи ми повинні передбачити різні варіанти подій. Для цього необхідно розробити Use case діаграму, яка буде демонструвати головних акторів нашої системи (користувачів та зовнішніх акторів, таких як сервер) та залежності й функціонал, який має бути присутнім. Таким чином ми зможемо відобразити дані відношення в проекті, правильно побудувавши взаємозв'язки. Use case діаграма представлена на рисунку 3.1, опис акторів системи та варіанти використання описані у Додатку Б (таблиці Б.1 та Б.2 відповідно).

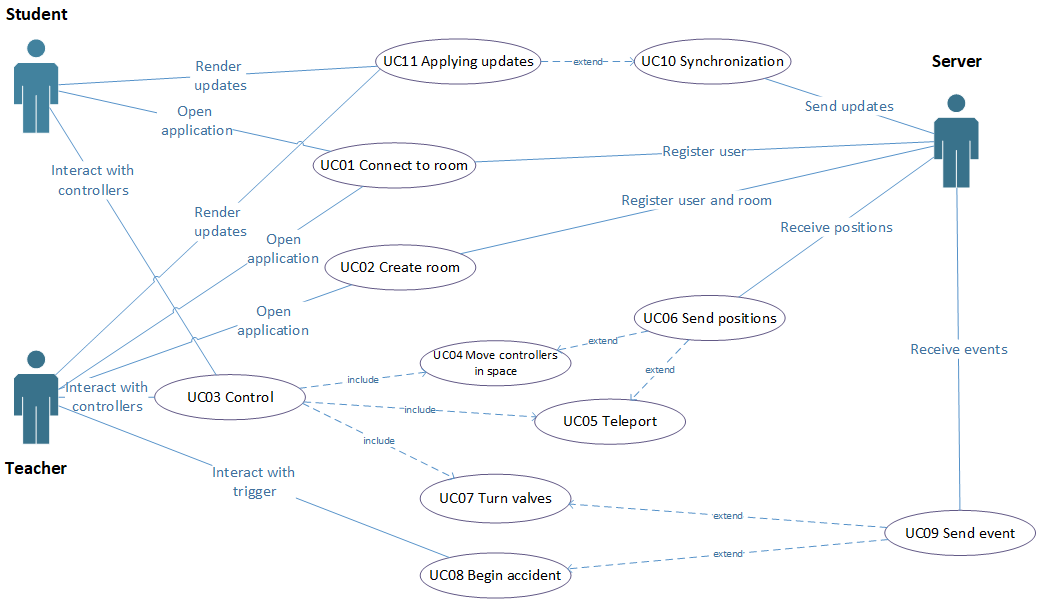

Рисунок 3.1 – Use case діаграма

Як бачимо з рисунку ми маємо три актори – студента, викладача та сервер. Синіми стрілками вказано можливості, які можуть застосовувати дані учасники.

Перш за все, нам необхідно розробити форму авторизації нашого додатку. Створюємо два текстових поля логіну та паролю, де користувач має ввести свої дані, та кнопку Login, яка буде надсилати запит на сервер (рис. 3.2). Для поля пароль ставимо тип password у вікні Unity Editor, для того щоб символи були автоматично замінені на зірочки, для захисту від перегляду.

Рисунок 3.2 – Форма авторизаці

По натисканню на кнопку Login ми формуємо JSON строку, яка дістає дані зі створених полів, конвертує їх у потрібний формат та надсилає як тіло запиту на сервер авторизації.

Запускаємо корутину, яка надсилає запит з потрібними нам параметрами, а також очікує на результат виконання:

```
private IEnumerator onResponse(string url, string jsonData)
    {
        var req = new UnityWebRequest(url, "POST");
        byte[] jsonToSend = new
System.Text.UTF8Encoding().GetBytes(jsonData);
        req.uploadHandler = new UploadHandlerRaw(jsonToSend);
```

```
        req.downloadHandler = new DownloadHandlerBuffer();
        req.SetRequestHeader("Content-Type", "application/json");

        yield return req.SendWebRequest();

        if (req.isNetworkError)
        {
            Debug.Log("Error While Sending: " + req.error);
        }
        else
        {
            if (req.responseCode == (int)StatusCodes.OK)
            {
                AuthResponse authData =
JsonUtility.FromJson<AuthResponse>(req.downloadHandler.text);
                UserStore.instance.user = authData.data;
                PlayerPrefs.SetString('refreshToken',
authData.refreshToken);
                SceneManager.LoadScene("Lobby");
            }
            else
            {
                ResponseView response =
JsonUtility.FromJson<ResponseView>(req.downloadHandler.text);
                errorContainer.text = response.message;
            }
        }
    }
```

У випадку, коли ми отримуємо помилку, наприклад no internet connection або statusCode 400, виводимо користувачу відповідне повідомлення (рис. 3.3).

Якщо запит був виконаний успішно, парсимо отриману відповідь та зберігаємо дані користувача в спеціальний об’єкт що реалізує паттерн Singleton , щоб мати до нього доступ з будь-якого місця нашого додатку. Refresh токен зберігаємо в об’єкті PlayerPrefs. Таким чином, ми його зберігаємо у локальному сховищі і при наступному запуску додатку можемо дістати його та надіслати запит на оновлення пари access/refresh токенів, пропустивши етап авторизації та оновивши дані про користувача.

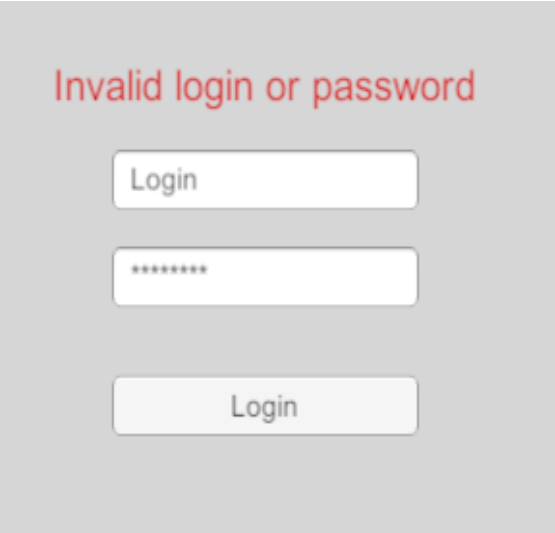

Рисунок 3.3 – Повідомлення про помилку

Наступним механізмом, який нам треба розробити, є система кімнат. Це необхідно для того, щоб паралельно могли займатися декілька груп студентів, не заважаючи одне одному. До того ж, маємо надати можливість користувачам з роллю 1 (teacher) створювати кімнати для навчання. Увійти у створену кімнату можуть користувачі з будь-якою роллю. Тому необхідно додати перелік кімнат. Інструментом, який дозволяє виконати ці дії, є вбудований інструмент Photon Pun (рис. 3.4).

Для цього розроблюємо інтерфейс, який реалізує наші потреби, наприклад кнопки для створення кімнат, а також перелік самх кімнат (рис. 3.5). Додатково додамо вікно відладк для того щоб можна було контролювати процес на будь-якому етапі розробки.

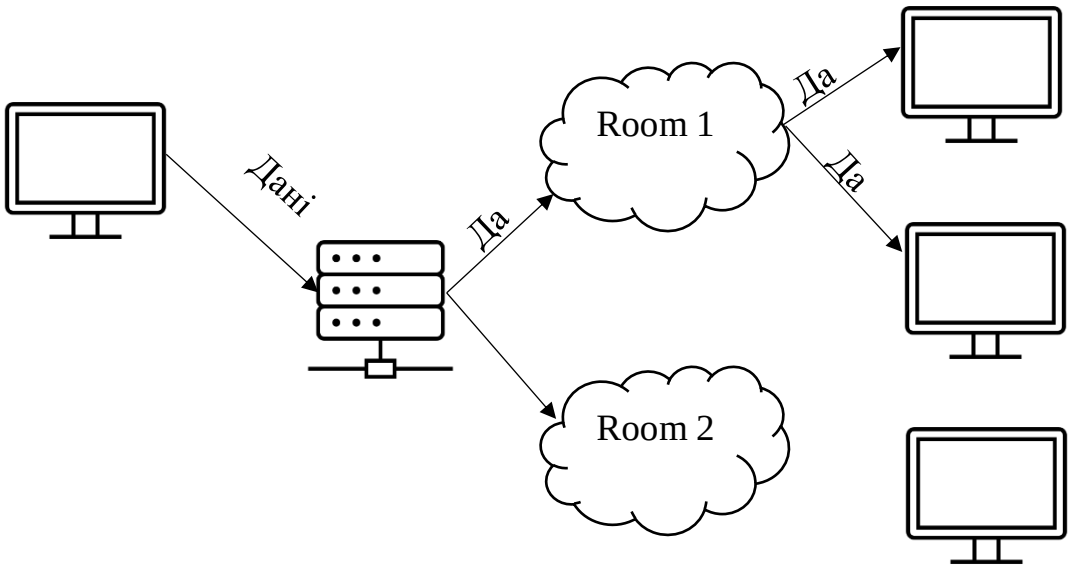

Рисунок 3.4 – Модель роботи системи Photon Network

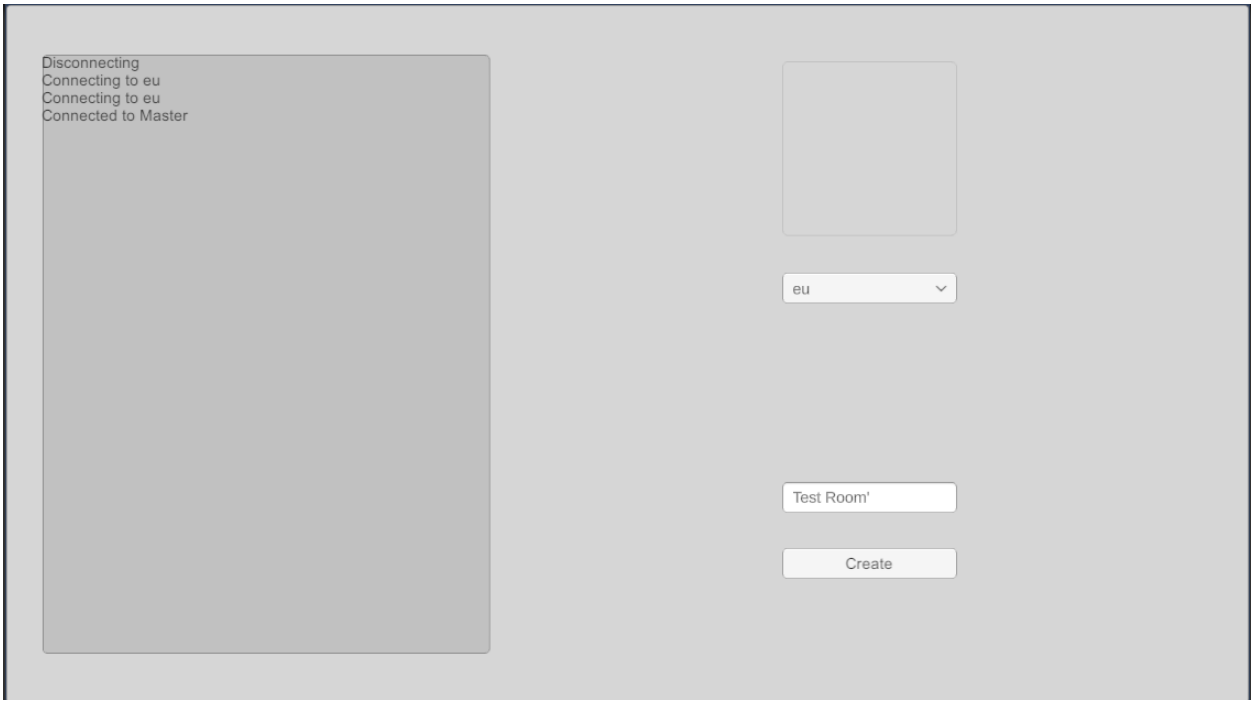

Рисунок 3.5 – Вікно з переліком кімнат

Для створення кімнати необхідно ввести її назву, по якій потім можна під’єднатися, та натиснути Create. Після чого, за допомогою методу photonNetworkCreateRoom, з передачею таких параметрів як назва,

максимальні кількість користувачів (20) та типу кімнати додаємо запис у систему Photon та автоматично переносимо користувача на головну сцену нашого додатку, тобто станцію осушки природного газу. Реалізація методу photonNetworkCreateRoom:

```
public void CreateRoom()
    {
        if ( room.text == "")
        {
            return;
        }

        PlayerPrefs.SetString("Room", room.text);
        PhotonNetwork.CreateRoom(room.text, new Photon.Realtime.RoomOptions
{ MaxPlayers = 20 }, Photon.Realtime.TypedLobby.Default);
    }
```

Після цих маніпуляцій у всіх інших користувачів з’явиться кнопка з назвою кімнати, яка дозволить підключится до викладача (рис. 3.6) . В цей час сервер викличе колбек, який відслідкує зміни у переліку кімнат системи Photon та оновить список на екрані користувачів.

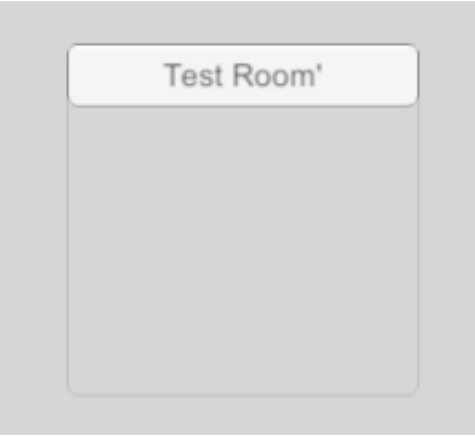

Рисунок 3.6 – Поява кімнати у студента

Натискаємо на кнопку, здійснюємо переміщення до кімнати, де будуть чекати усі клієнти прив’язані до даної сцени. Так як сервери Photon

розташовані у різних куточках світу ми маємо можливість перемикатися між серверами для зменшення затримки у користувачів. Це буде корисно коли викладач та студенти знаходяться в різних країнах. Обравши оптимальний регіон серверу, можливо багатократно підвищити якість досвіду, отриманого під час використання нашого додатку. Тому додаємо випадаючий список, де можна обрати потрібний нам регіон (рис. 3.7).

Рисунок 3.7 – Обрання необхідного регіону

Однак, потрібно розуміти, що користувачі, які знаходяться на різних серверах, не зможуть взаємодіяти один з одним.

Тепер, коли усі користувачі під’єднані до однієї кімнати, слід попіклуватися про їх взаємодію. Першочергово необхідно відобразити аватар користувача на всіх інших клієнтських додатках, щоб інші могли ідентифікувати один одного, як з точки зору юзерів, так і Photon. Тому створюємо префаб аватару, який ми будемо інсталювати на сцену при заході нового користувача. Усі люди, під’єднані до однієї кімнати матимуть власний контролер, через який буде відбуватися взаємодія із сервером.

Для того, щоб інсталювати аватар, треба при завантаженні головної сцени викликати метод PhotonNetwork.Instantiate (), в який передаємо префаб та координати спавну нашого об’єкту. Проте, просто створити аватари недостатньо, потрібно синхронізувати їх рухи між усіма користувачами. Для

цього накладаємо на префаб аватару компонент PhotonTransformView, який надасть нашому об’єкту спеціальний ідентифікатор та виконає синхронізацію положення і повороту. Але, так як ми додаємо мережеві функці для додатку віртуальної реальності, у якого є три об’якти для відстеження: шолом або голова та два контролери (руки). Компонент PhotonTransformView дозволить налаштувати синхронізацію положення лише для шолому, тому, у зв’язку зі складністю відстеження позиції контролерів, передача даних про положення контролерів лягає на наші плечі.

Для стабільної роботи мережі необхідно коректно реагувати на затримки передачі та мінімізувати кількість обміну даних, наприклад зменшити частоту відправки. Проте, у такому випадку потрібно скористатися деякими механізмами згладжування переміщення. У нашому випадку ефективними алгоритмами будуть передбачення на стороні клієнта та інтерполяція сутності.

Сутність першого полягає у тому, що ми використовуємо ігрові події, які є продовженими в часі, тобто виконується певну кількість секунд. Таким чином, на стороні клієнта ми можемо запустити анімацію, яка покаже певну дію клієнту, наприклад піде дим із контрольної панелі при неправильних налаштуваннях, хоча сервер ще не встиг обробити даний запит. Отже, для клієнта навколишній світ буде без затримки (яка неминуча при роботі із мережею) реагувати на його дії (рис. 3.8)

Інтерполяція сутностей (рис. 3.9) у свою чергу допоможе зробити переміщення користувачів плавними. Ідея цього методу полягає у затримці відображення нового положення. Наприклад ми отримали позицію користувача у час T = 2. Раніше ми вже отримали позицію у час T = 1, тобто ми знаємо приблизний шлях руху гравця та час, за який було здійснено

переміщення. Єдиним нашим завданням залишається знайти проміжні точки між цими координатами на відобразити результат на екран [23].

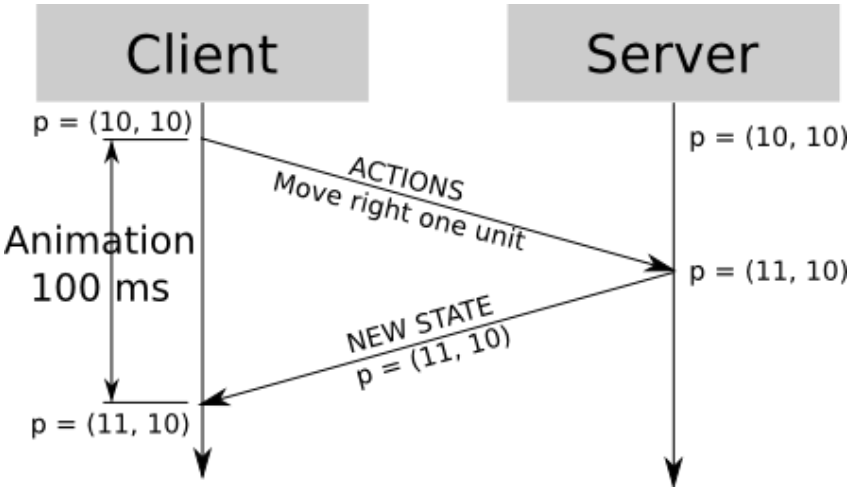

Рисунок 3.8 – Анімація в той час як сервер підтверджує дію

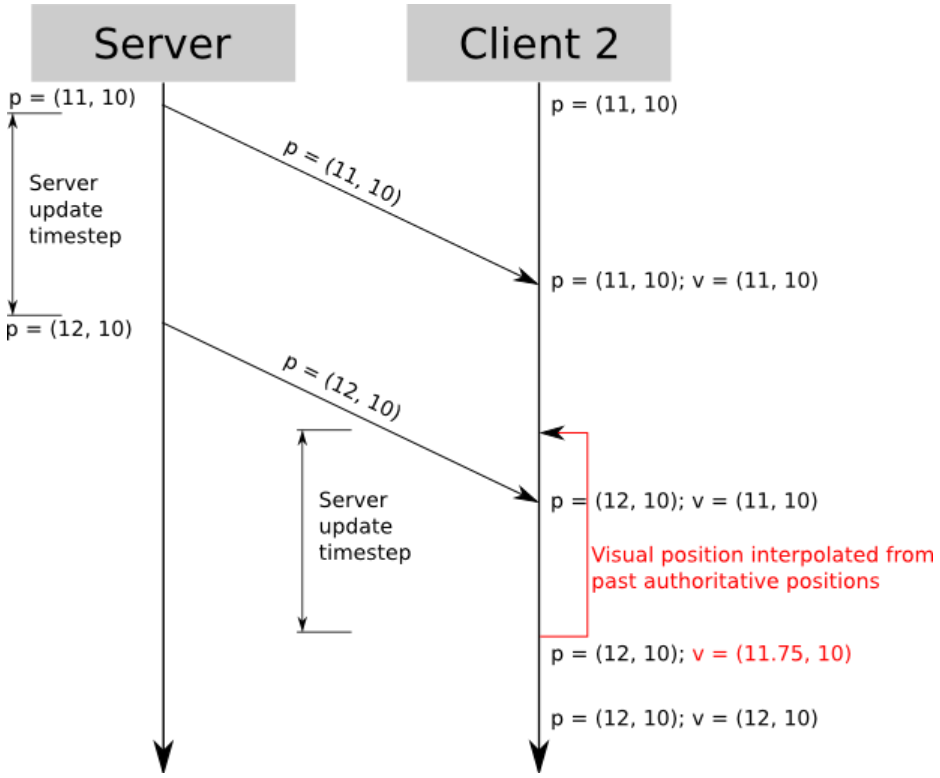

Рисунок 3.9 – Клієнт 2 відображає Клієнта 1 у минулому

Звісно, при такому підході позиції відображатимуться з затримкою, яка дорівнює інтервалу відправки даних, проте, це не критично у додатках, в яких не потрібна велика точність. У противагу ми нівелюємо затримку на передавання даних та їх кількість.

Окрім руху користувачів в нашій системі існують різні події. Їх синхронізація також необхідна. Для виконання поставленої задачі будемо використовувати систему івентів, яку нам надає Photon Pun. Для цього в компонентах, які відповідають за елементи, що мають реагувати на такі події, як контролер ліфту та аварії. Треба додати наслідування від інтерфейсу IonEventCallback. Після цього нам стане доступна для перевизначення функція OnEvent(EventData photonEvent). У якості параметра даної функції ми отримаємо об’єкт типу EventData, в якому міститься ідентифікатор івенту. Відповідно до цього ідентифікатора ми можемо відрізнити одну подію від іншої.

Так, для синхронізації руху ліфту необхідно очікувати коли всі користувачі стануть на потрібну платформу. Коли це відбудеться, то спрацює івент, який означатиме початок руху платформи ліфта у всіх клієнтів.

Аналогійчним способом синхронізуємо початок аварійної ситуації. Після натискання кнопки «Початок аварії» викладачем у всіх інших клієнтів спрацює івент, що позначатиме початок аварійної ситуації. Відповідно таймер зворотнього відліку запуститься.

Робота вентилів також адаптована. Для запобігання плутанини, одночасно кожен такий об’єкт може використовуватись лише одним користувачем.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра виконано аналіз предметної області, визначено актуальність проблеми. Додавання мережевих сервісів для VR додатку дозволяє збільшити ефективність освіти, об’єднати студентів та викладачів з різних куточків світу в одному віртуальному інформаційному просторі. Також проведено аналіз додатків та платформ аналогів для визначення необхідного функціоналу нашого проекту. Після цього було сформовано мету роботи та поставлено завдання до виконання під час роботи над дипломною роботою.

У другому розділі ми виконали аналіз існуючих програмних продуктів для реалізації серверу авторизації та щупинили свій вибір на Nest.js з використанням Node.js. Крім того, у другому розділі докладно описали використані паттерни: шаблон репозиторій та dependency injection. У якості бібліотеки мережевого спроводу для Unity обрано Photon Pun.

У третьому і останньому розділі ми розібрали покроково процесс роботи над розробкою серверу авторизації, системи кімнат та синхронізацією рухів та подій.

Даний проект є кінцевим продуктом, але подальший розвиток та робота над ним є можливою, адже буде тільки на краще додати новий функціонал, який буде новою версією додатку.

СПИСОК ЛІТЕРАТУРИ

1. Про затвердження Програми розвитку системи дистанційного навчання на 2004-2006 роки: Постанова Кабінету Міністрів України від 23 вересня 2003 р. №1494. Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/1494-2003-%D0%BF#Text

2. Муращенко Т. В. Змішане та дистанційне навчання як спосіб доступу до якісної освіти. *Відкрите освітнє е-середовище сучасного університету*. 2019. № 3. С. 283-287.

3. Деякі питання організації дистанційного навчання: наказ Міністерства освіти і науки України від 8 вересня 2020 р. №1115, зареєстрований у Міністерстві юстиції України 28 вересня 2020 року №941/35224. Режим доступу до ресурсу: https://mon.gov.ua/ua/npa/deyaki-pitannya-organizaciyi-distancijnogo-navchannya-zareyestrovano-v-ministerstvi-yusticiyi-ukrayini-94735224-vid-28-veresnya-2020-roku

4. Положення про дистанційне навчання (затверджене наказом Міністерства освіти і науки України від 25.04.2013 р. за №466; зареєстроване в Міністерстві юстиції України 30.04.2013 р. за №703/23235). Режим доступу до ресурсу: http://ru.osvita.ua/legislation/Dist_osv/2999/

5. Jason J. The VR Book: Human-Centered Design for Virtual Reality. Morgan & Claypool, 2019. 636 p.

6. Що таке VR. Поняття віртуальної реальності [Електронний ресурс]// Adobe, 2022. – Режим доступу до ресурсу: https://www.adobe.com/ua/products/substance3d/discover/what-is-vr.html

7. Сальник І. В. Віртуальність як складова сучасного середовища. Режим доступу до ресурсу:: https://phm.cuspu.edu.ua/nauka/mizhnarodnyi-

naukovyi-tsentr-prykladnykh-doslidzhen/anonsy-diialnist/2111-rozshirenij-oglyad-zmistu-naukovogo-doslidzhennya-profesorki-i-v-sal-nik-virtual-nist-yak-skladova-suchasnogo-seredovishcha.html

8. Віртуальна реальність: принципи роботи та переваги для навчання. Режим доступу до ресурсу: https://teach-hub.com/virtualna-realnist/

9. 3dorganon [Електронний ресурс]// 3dorganon, 2022. – Режим доступу до ресурсу: https://www.3dorganon.com/

10. Mix Sumdu [Електронний ресурс]// SUMDU, 2022. – Режим доступу до ресурсу: https://mix.sumdu.edu.ua/

11. 4danatomy [Електронний ресурс]// 4danatomy, 2022. – Режим доступу до ресурсу: https://www.4danatomy.com/

12. Що таке авторизація? Визначення з техопедії [Електронний ресурс]// 2022. – Режим доступу до ресурсу: https://uk.theastrologypage.com/authorization

13. Why The Hell Would I Use Node.js? A Case-by-Case Tutorial [Електронний ресурс]// Noptal, LLC, 2010-2022. – Режим доступу до ресурсу: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js#:~:text=js%3F-,Node.,not%20run%20within%20a%20browser.

14. 7 Advantages of Node.js for Startups/ [Електронний ресурс]// Relevant Software LLC, 2022. – Режим доступу до ресурсу: https://relevant.software/blog/7-benefits-of-node-js-for-startups/#Why_is_Nodejs_great_for_startups

15. What is TypeScript? [Електронний ресурс]// TypeScript, 2022. – Режим доступу до ресурсу: https://www.typescriptlang.org/

16. NestJS vs. Hapi [Електронний ресурс]// LogRocket, 2022. – Режим доступу до ресурсу: https://blog.logrocket.com/nestjs-vs-hapi/#nest-js-overview

17. Controller-Service-Repository. [Електронний ресурс]// 2021. – Режим доступу до ресурсу: https://tom-collings.medium.com/controller-service-repository-16e29a4684e5

18. Mark Seemann. Dependency Injection is Loose Coupling [Електронний ресурс]// 2021. – Режим доступу до ресурсу: https://blog.ploeh.dk/2010/04/07/DependencyInjectionisLooseCoupling/

19. Арестов А. В. Особливості застосування технології віртуальної реальності у навчальному процесі. - магістерс. дисертація.- Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського». Київ – 2018 р. – 112 с.

20. Трач Ю. VR – технології як метод і засіб навчання// Освітологічний дискурс: Київський університет Б. Грінченка, 2017, № 3-4(18-19). - Київ, 2017. – с. 309-322.

21. Гончаренко С. А., Кузьмук Д. А., Шаповалов С. П. Додаток доповненої реальності «Augmented reality education»// МА:: ІМА – 2020. Матеріали та програма науково-технічної конференції СумДУ. 2020. - С. 36.

22. Гончаренко С. А., Кузьмук Д. А., Шаповалов С. П. Впровадження додатків віртуальної реальності в процеси університетського навчання// Матеріали науково-технічної конференції «Інформатика, математика, автоматика», Суми, СУмДУ, 2021. – с. 65-66.

23. Гончаренко С. А., Кузьмук Д. А. Застосування технологій віртуальної реальності в навчальному процесі університету. Наукова робота, Мелітополь – 2021. – 30 с.

ДОДАТОК А

Лістинг серверу авторизації

Файл app.module.ts

```
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { KnexModule, KnexOptions } from '@nestjsplus/knex';
import { KnexOptionsFactoryImpl } from './config/knex-option.factory';
import { ControllersModule } from './controllers/controllers.module';
import { AuthModule } from './modules/auth/auth.module';

@Module({
  imports: [
    ConfigModule.forRoot(),
    KnexModule.registerAsync({
      useClass: KnexOptionsFactoryImpl,
    }),
    AuthModule,
    ControllersModule,
  ],
})
export class AppModule {}
```

Файл knex-option.factory.ts

```
import { KnexOptions, KnexOptionsFactory } from '@nestjsplus/knex';

export class KnexOptionsFactoryImpl implements KnexOptionsFactory {
    createKnexOptions(): KnexOptions {
        return {
            client: process.env.DB_DRIVER,
            debug: false,
            connection: {
              host: process.env.DB_HOST,
              user: process.env.DATABASE_USERNAME,
              database: process.env.DATABASE_SCHEMA,
              port: +process.env.DB_PORT,
            },
```

```
        };
    }
}
```

Файл controllers.module.ts

```
import { Module } from '@nestjs/common';
import { AuthController } from './auth.controller';

@Module({
  imports: [],
  controllers: [AuthController],
})
export class ControllersModule {}
```

Файл auth.controller.ts

```
import { Body, Controller, Get, HttpCode, Inject, Post } from '@nestjs/common';
import { RefreshForm } from 'src/models/from-models/refresh.form';
import { UserCredentialsForm } from 'src/models/from-models/user-credentials.form';
import { AUTH_SERVICE_TOKEN, IAuthService } from
'src/modules/auth/auth.service.interface';

@Controller('auth')
export class AuthController {
  constructor(@Inject(AUTH_SERVICE_TOKEN) private readonly authService:IAuthService)
{}

  @Post('/login')
  @HttpCode(200)
  public async login(
    @Body() userCredential: UserCredentialsForm,
  ): Promise<any> {
    return await this.authService.login(userCredential.username,
userCredential.password);
  }

  @Post('/refresh')
  @HttpCode(200)
  public async refresh(
    @Body() refreshForm: RefreshForm,
  ): Promise<any> {
    return await this.authService.refreshAccessToken(refreshForm.refreshToken);
  }
}
```

Файл auth.repository.interface.ts

```
import { UserDTO } from 'src/models/DTO/user.dto';

export const AUTH_REPOSITORY_TOKEN = Symbol.for('authRepository');

export interface IAuthRepository {
  checkAuth(login: string, hashPassword: string): Promise<UserDTO>;

  getUserDetailsById(userId: number): Promise<UserDTO>;
}
```

Файл auth.repository.ts

```
import { Inject } from '@nestjs/common';
import { Knex } from 'knex';
import { UserDTO } from 'src/models/DTO/user.dto';
import { IAuthRepository } from './auth.repository.interface';
import { KNEX_CONNECTION } from '@nestjsplus/knex';

export const AUTH_REPOSITORY_TOKEN = Symbol.for('authRepository');

export class AuthRepository implements IAuthRepository {
  constructor(@Inject(KNEX_CONNECTION) private readonly knex: Knex) {}

  public async getUserDetailsById(userId: number): Promise<UserDTO> {
    return this.knex('tb_users').select('*').where('PK_userId', userId).first();
  }

  public async checkAuth(
    login: string,
    hashPassword: string,
  ): Promise<UserDTO> {
    return this.knex('tb_users')
      .select('*')
      .where('login', login)
      .andWhere('hashPassword', hashPassword)
      .first();
  }
}
```

Файл auth.service.interface.ts

```
export const AUTH_SERVICE_TOKEN = Symbol.for('authService');

export interface IAuthService{
    login(login: string, password: string): Promise<any>;

    refreshAccessToken(refreshToken: string): Promise<any>;
}
```

Файл auth.service.ts

```
import {
  BadRequestException,
  CACHE_MANAGER,
  ForbiddenException,
  Inject,
  UnauthorizedException,
} from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { UserDTO } from 'src/models/DTO/user.dto';
import { hashPassword } from 'src/utils/password';
import {
  AUTH_REPOSITORY_TOKEN,
  IAuthRepository,
} from './auth.repository.interface';
import { IAuthService } from './auth.service.interface';
import { v4 as uuid } from 'uuid';
import Cache from 'cache-manager';

export class AuthService implements IAuthService {
  constructor(
    @Inject(AUTH_REPOSITORY_TOKEN)
    private readonly authRepository: IAuthRepository,
    @Inject(CACHE_MANAGER) private cacheManager: Cache,
    private readonly jwtService: JwtService,
  ) {}

  public async refreshAccessToken(refreshToken: string): Promise<any> {
    const decodedToken = await this.jwtService.verifyAsync(refreshToken);
    const userId = await this.cacheManager.get(decodedToken.refreshId);
    if(!userId){
      throw new UnauthorizedException;
    }
    const userData = await this.authRepository.getUserDetailsById(userId);
    return {
      data: { role: userData.FK_PK_roleId, userName: userData.name },
```

```
    accessToken: await this.generateAccessToken(userData),
    refreshToken: await this.generateRefreshToken(userData),
  };
}

public async login(login: string, password: string): Promise<any> {
  const userData = await this.authRepository.checkAuth(
    login,
    hashPassword(password),
  );
  if (!userData) {
    throw new BadRequestException('Invalid login or password');
  }
  return {
    data: { role: userData.FK_PK_roleId, userName: userData.name },
    accessToken: await this.generateAccessToken(userData),
    refreshToken: await this.generateRefreshToken(userData),
  };
}

private async generateAccessToken(userData: UserDTO): Promise<string> {
  return await this.jwtService.signAsync(
    {
      userId: userData.PK_userId,
      role: userData.FK_PK_roleId,
      userName: userData.name,
    },
    {
      secret: process.env.JWT_SECRET,
      expiresIn: process.env.JWT_EXPIRED_IN,
    },
  );
}

private async generateRefreshToken(userId): Promise<string> {
  const refreshId = uuid();
  await this.cacheManager.set(refreshId, userId);
  return await this.jwtService.signAsync(
    {
      refreshId,
    },
    {
      secret: process.env.JWT_SECRET,
      expiresIn: process.env.JWT_REFRESH_EXPIRED_IN,
    },
```

```
    );
  }
}
```

Файл password.ts

```
import * as crypto from 'crypto';

export const hashPassword = (password) => {
  let result = password;
  for (let i = 0; i < 5; i++) {
    result = crypto.createHash('sha256').update(result, 'utf8').digest('hex');
  }
  return result;
};
```

ДОДАТОК Б

Лістинг мережевого супроводу додатку віртуальної реальності

Файл AuthorizationService.cs

```
using System;
using System.Collections;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

[Serializable]
class AuthRequestForm
{
    public string username;
    public string password;
}


public class UserDetail
{
    public Roles role;
    public string userName;
}

class AuthResponse
{
    public UserDetail data;
    public string accessToken;
    public string refreshToken;
}

class ResponseView
{
    public int statusCode;
    public string message;
    public string error;

    public override string ToString()
    {
```

```
        return "statusCode:" + statusCode + " message:" + message + " error:" +
error;
    }
}

enum StatusCodes
{
    OK = 200,
    BadRequest = 400
}

public enum Roles
{
    Student = 1,
    Teacher = 2
}


public class AuthorizationService : MonoBehaviour
{
    [SerializeField]
    private string authServiceURL;
    [SerializeField]
    private InputField loginField;
    [SerializeField]
    private InputField passwordField;
    [SerializeField]
    private Text errorContainer;


    public void Request()
    {
        errorContainer.text = "";
        AuthRequestForm authRequest = new AuthRequestForm();
        authRequest.username = loginField.text;
        authRequest.password = passwordField.text;
        String requestJson = JsonUtility.ToJson(authRequest).ToString();
        StartCoroutine(onResponse(authServiceURL + "auth/login",requestJson));

    }

    private IEnumerator onResponse(string url, string jsonData)
    {
        var req = new UnityWebRequest(url, "POST");
        byte[] jsonToSend = new System.Text.UTF8Encoding().GetBytes(jsonData);
```

```
        req.uploadHandler = new UploadHandlerRaw(jsonToSend);
        req.downloadHandler = new DownloadHandlerBuffer();
        req.SetRequestHeader("Content-Type", "application/json");

        yield return req.SendWebRequest();

        if (req.isNetworkError)
        {
            Debug.Log("Error While Sending: " + req.error);
        }
        else
        {
            if (req.responseCode == (int)StatusCodes.OK)
            {
                AuthResponse authData =
JsonUtility.FromJson<AuthResponse>(req.downloadHandler.text);
                UserStore.instance.user = authData.data;
                PlayerPrefs.SetString("refreshToken", authData.refreshToken);
                SceneManager.LoadScene("Lobby");
            }
            else
            {
                ResponseView response =
JsonUtility.FromJson<ResponseView>(req.downloadHandler.text);
                errorContainer.text = response.message;
            }
        }
    }
```

Файл LobbyManager.cs

```
using Photon.Pun;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.UI;

using Photon.Realtime;

public class LobbyManager : MonoBehaviourPunCallbacks
{
    public TMP_Dropdown regions;
    public TMP_InputField room;
    public GameObject panelWithRooms;
```

```
public GameObject buttonToConnect;
public Text debugText;
public GameObject adminObj;


private SetAdmin admin;

private void Start() {
    if(GameObject.FindGameObjectsWithTag("Admin").Length == 0)
    {
        Instantiate(adminObj, Vector3.zero, Quaternion.identity);
    }

    admin = GameObject.FindGameObjectWithTag("Admin").GetComponent<SetAdmin>();
    if (PlayerPrefs.HasKey("Region"))
    {
        regions.value = PlayerPrefs.GetInt("Region");
    }
    if (PlayerPrefs.HasKey("Room"))
    {
        room.text = PlayerPrefs.GetString("Room");
    }
    if (!admin.restart)
    {
        ConnectToServer();
    }
    PlayerPrefs.SetInt("Screenmanager Resolution Width", 800);
    PlayerPrefs.SetInt("Screenmanager Resolution Height", 600);
    PlayerPrefs.SetInt("Screenmanager Is Fullscreen mode", 0);

}

public override void OnConnectedToMaster()
{
    Log("Connected to Master");
    if (!admin.restart)
    {
        PhotonNetwork.JoinLobby();
    }
    else
    {
        RestartRoom();
    }
}
```

```
    public void CreateRoom()
    {
        if ( room.text == "")
        {
            return;
        }

        PlayerPrefs.SetString("Room", room.text);
        PhotonNetwork.CreateRoom(room.text, new Photon.Realtime.RoomOptions {
MaxPlayers = 20 }, Photon.Realtime.TypedLobby.Default);
    }

    public void ConnectToServer()
    {
        Log("Connecting to " + regions.options[regions.value].text);
        PlayerPrefs.SetInt("Region", regions.value);
        PhotonNetwork.AutomaticallySyncScene = true;
        PhotonNetwork.GameVersion = "1";
        PhotonNetwork.PhotonServerSettings.AppSettings.FixedRegion =
regions.options[regions.value].text;
        PhotonNetwork.ConnectUsingSettings();
    }


    public void JoinRoom(string nameOfRoom)
    {
        PhotonNetwork.JoinRoom(nameOfRoom);
    }

    public override void OnJoinedRoom()
    {
        Log("Joined the room");
        PhotonNetwork.LoadLevel("Main");
    }
    private void Log(string logText)
    {
        Debug.Log(logText);
        debugText.text += logText + "\n";
    }

    public void Reconnect()
    {
        Log("Disconnecting");
        PhotonNetwork.Disconnect();
        ConnectToServer();
```

```
    }

    public override void OnRoomListUpdate(List<RoomInfo> roomList)
    {

        foreach(RoomInfo room in roomList)
        {
            if (room.RemovedFromList)
            {
                Destroy(panelWithRooms.transform.Find(room.Name).gameObject);
            }
            else
            {
                GameObject btn = Instantiate(buttonToConnect, Vector3.zero,
Quaternion.identity);
                btn.transform.SetParent(panelWithRooms.transform);
                btn.name = room.Name;
                btn.GetComponent<Button>().onClick.AddListener(() =>
JoinRoom(room.Name));
                btn.GetComponentInChildren<Text>().text = room.Name;
            }
        }
    }
    public void RestartRoom()
    {
        room.text = admin.roomNameToRestart;
        PhotonNetwork.JoinOrCreateRoom(admin.roomNameToRestart, new
Photon.Realtime.RoomOptions { MaxPlayers = 20 }, Photon.Realtime.TypedLobby.Default);
    }
}
```

Файл PlayerMultControll.cs

```
using UnityEngine;
using Photon.Pun;

public class PlayerMultControll : MonoBehaviour, IPunObservable
{
    public Transform parentTramsform;

    private Vector3 TargetLeftPosition;
    private Vector3 TargetRightPosition;
    private Quaternion TargetLeftRotation;
```

```
    private Quaternion TargetRightRotation;

    private PhotonView photonView;
    public GameObject leftHand;
    public GameObject rightHand;


    private GameObject[] valves;
    private bool[] valvesActivity = new bool[6];
    private Quaternion[] valvesRotation = new Quaternion[6];

    public void OnDisable()
    {
        PhotonNetwork.RemoveCallbackTarget(this);
    }

    void OnEnable()
    {
        valves =
GameObject.FindGameObjectWithTag("Head").GetComponent<Info>().valves;
        PhotonNetwork.AddCallbackTarget(this);
        photonView = GetComponent<PhotonView>();
        if (photonView.IsMine)
        {
            foreach (Transform child in this.transform)
            {
                child.gameObject.SetActive(false);
            }

this.transform.SetParent(GameObject.FindGameObjectWithTag("Head").GetComponent<Info>(
).getParams.currPosition);
        }
    }

    void IPunObservable.OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo
info)
    {

        if (stream.IsWriting)
        {

stream.SendNext(GameObject.FindGameObjectWithTag("Head").GetComponent<Info>().getPara
ms.leftHand.transform.position);
```

```
stream.SendNext(GameObject.FindGameObjectWithTag("Head").GetComponent<Info>().getPara
ms.leftHand.transform.rotation);

stream.SendNext(GameObject.FindGameObjectWithTag("Head").GetComponent<Info>().getPara
ms.rightHand.transform.position);

stream.SendNext(GameObject.FindGameObjectWithTag("Head").GetComponent<Info>().getPara
ms.rightHand.transform.rotation);
            foreach (GameObject gn in valves)
            {
                stream.SendNext(gn.transform.rotation);
                stream.SendNext(gn.GetComponent<ActivityOfValve>().is_in_active);
            }
        }
        else
        {
            TargetLeftPosition = (Vector3)stream.ReceiveNext();
            TargetLeftRotation = (Quaternion)stream.ReceiveNext();
            TargetRightPosition = (Vector3)stream.ReceiveNext();
            TargetRightRotation = (Quaternion)stream.ReceiveNext();
            for (int i = 0; i < 6; i++)
            {
                valvesRotation[i] = (Quaternion)stream.ReceiveNext();
                valvesActivity[i] = (bool)stream.ReceiveNext();
            }
        }
    }

    // Update is called once per frame
    void FixedUpdate()
    {
        if (!photonView.IsMine)
        {
            this.transform.GetChild(0).transform.position =
Vector3.Lerp(this.transform.GetChild(0).transform.position, TargetLeftPosition,
0.1f);
            this.transform.GetChild(1).transform.position =
Vector3.Lerp(this.transform.GetChild(1).transform.position, TargetRightPosition,
0.1f);
            this.transform.GetChild(0).transform.rotation =
Quaternion.RotateTowards(this.transform.GetChild(0).transform.rotation,
TargetLeftRotation, 720f * Time.deltaTime);
```

```
            this.transform.GetChild(1).transform.rotation =
Quaternion.RotateTowards(this.transform.GetChild(1).transform.rotation,
TargetRightRotation, 720f * Time.deltaTime);
            for (int i = 0; i < 6; i++)
            {
                if (valvesActivity[i])
                {
                    valves[i].GetComponent<ActivityOfValve>().other_activity = true;
                    valves[i].transform.rotation =
Quaternion.Lerp(valves[i].transform.rotation, valvesRotation[i], 0.1f);
                    valves[i].GetComponent<Rigidbody>().detectCollisions = false;
                }
                else
                {
                    valves[i].GetComponent<Rigidbody>().detectCollisions = true;
                }
            }
        }
    }
}
```

ДОДАТОК В

Дані таблиці є результатом наукової роботи на тему «Застосування технологій віртуальної реальності в навчальному процесі університету» [23].

Таблиця Б.1 Опис акторів системи

Роль	**Можливості**
Student	Актор користувач системи. Має наступні можливості: - підключатися до вже існуючої кімнати; - рухатися кімнатою, тими зонами в ній, які доступні користувачу Student; - керувати станцією, натискати на кнопки та рухати вентилі.
Teacher	Актор користувач системи. Аналогічний функціонал як у користувача Student плюс наступні додаткові можливості: - створювати нові кімнати; - перезавантажувати сесію; - запускати аварійну ситуацію.
Server	Як зовнішній актор системи, Server відповідає за те, щоб здійснювати обмін інформацією між користувачами, які знаходяться в кімнаті, повідомляє їх про преміщення та дії.

Таблиця Б.1 Опис усіх варіантів використання системи

Назва ВВ	**Сценарій**

UC01 Connect to room	**Customer та Teacher** зацікавлені у підключенні до кімнати. **Server** зацікавлений у реєстрації сесії нового користувача. **X.1 Передумови** Програма працює, відкрите вікно лобі. **Customer або Teacher** натискає кнопку із обраною кімнатою. **X.2 Процес** **Customer або Teacher** обирає з переліку потрібну кімнату та натискає на відповідну кнопку. **Server** отримує ідентифікатор користувача й кімнати, зв'язує отримані дані та надає доступ на входження . **X.3 Альтернативні потоки (обробка помилок)** Щоб закрити додаток потрібно натиснути на кнопку «хрестик» у правому верхньому куті вікна. При збою зв'язку виникає відповідне повідомлення у консолі. **X.4 Післяумови** Після надходження дозволу **Customer** або **Teacher** потрапляє на сцену обраної кімнати.
UC02 Create room	**Teacher** зацікавлений у створенні нової кімнати. **Server** зацікавлений у реєстрації сесії нового користувача та кімнати.

	X.1 Передумови **Teacher** вводить назву нової кімнати та натискає кнопку «Create room». **X.2 Процес** **Techer** вводить у відповідне поле назву майбутньої кімнати, натискає кнопку «Create room» та надсилає ці дані до **Server**. Останній створює кімнату із отриманою назвою, прив’язує користувача та надає доступ на входження. **X.3 Альтернативні потоки (обробка помилок)** Щоб закрити додаток потрібно натиснути на кнопку «хрестик» у правому верхньому куті вікна. При збою зв’язку виникає відповідне повідомлення у консолі. **X.4 Післяумови** Після находження дозволу **Teacher** потрапляє на сцену обраної кімнати.
UC03 Control	**Customer або Teacher** зацікавлені у передачі даних із органів керування до додатку. **X.1 Передумови** **Customer або Teacher** натискає клавіши на контролерах або рухаються у просторі. **X.2 Процес**

	Customer або Teacher взаємодіють із гарнітурою віртуальної реальності, додаток відслідковує ці дії та надає відповідну реакцію. **X.3 Альтернативні потоки (обробка помилок)** При втраті зв'язку із обладнанням виводиться відповідне повідомлення на екран окулярів та у консоль. **X.4 Післяумови** **Customer та Teacher** бачать коректну реакцію додатка на їх дії.
UC04 Move controllers in space	Частина **UC03** **X.1 Передумови** **Customer або Teacher** рухає шолом або контролери у просторі. **X.2 Процес** При зміні положення гарнітури додаток вносить відповідні поправки до «аватару» **Customer або Teacher.** **X.3 Альтернативні потоки (обробка помилок)** Вихід «за текстури» супроводжується вібрацією контролерів. При втраті зв'язку із обладнанням виводиться відповідне повідомлення на екран окулярів та у консоль. **X.4 Післяумови**

	Customer та Teacher змінює точки обзору та положення віртуальних кінцівок.
UC05 Teleport	Частина **UC03** **X.1 Передумови** **Customer або Teacher** натискають на «circle pad» та обирають потрібний напрям. **X.2 Процес** Після натиску на кнопку наводиться контролер у потрібну сторону(напрям вказує зелений луч). Для завершення переміщення відпускається кнопка. **X.3 Альтернативні потоки (обробка помилок)** Якщо **Customer або Teacher** наводить у заборонену зону луч стає червоним, а телепортація не відбувається. **X.4 Післяумови** **Customer або Teacher** телепортується у обрану точку.
UC06 Send positions	Розширює **UC04** та **UC05**. **Server** зацікавлений в отримані даних про положення **Customer або Teacher.** **X.1 Передумови** **Customer або Teacher** рухаються у просторі. **X.2 Процес** Додаток зчитує рухи **Customer або Teacher** та передає їх до **Server.** **X.3 Альтернативні потоки (обробка помилок)**

	При збою зв’язку виникає відповідне повідомлення у консолі та на екран. **X.4 Післяумови** **Server** володіє інформацією про переміщення **Customer** та **Teacher**
UC07 Turn valves	Частина **UC03** **X.1 Передумови** **Customer або Teacher** натискають клавішу «trigger» поряд із вентилем. **X.2 Процес** Після натискання клавіші **Customer** або **Teacher** захоплює вентиль та можуть його вільно обертати для контролю установки. **X.3 Альтернативні потоки (обробка помилок)** Якщо вентиль захоплений іншим користувачем, неможливо ним керувати. **X.4 Післяумови** Вентиль обертається відповідно до руху контролеру, при виставлення правильних позицій завершається аварійна ситуація
UC08 Begin accident	**Teacher** зацікавлений у початку навчальної аварійної ситуації. **X.1 Передумови** **Teacher** натискає «контекстну» кнопку контролера.

	X.2 Процес Додаток реагує на натискання кнопки на контролері користувача **Teacherм** починає симуляцію аварійної ситуації. **X.3 Альтернативні потоки (обробка помилок)** При втраті зв'язку із обладнанням виводиться відповідне повідомлення на екран окулярів та у консоль. **X.4 Післяумови** Починається аварійна ситуація, вмикається сирена та зворотній відлік по закінченню якого відбудиться вибух труб.
UC09 Send event	Розширює **UC07** та **UC08**. **Server** зацікавлений в отримані даних про дії **Customer або Teacher.** **X.1 Передумови** **Customer або Teacher** повертає вентилі або **Teacher** починає аварійну ситуацію. **X.2 Процес** Після виконання відповідних дій додаток надсилає до **Server** дані для обробки та подальшого використання. **X.3 Альтернативні потоки (обробка помилок)**

	При збою зв’язку виникає відповідне повідомлення у консолі та на екран. **X.4 Післяумови** **Server** володіє інформацією про дії **Customer** або **Teacher**
UC10 Synchronization	**Server** зацікавлений у відправці оброблених даних до клієнтів. **X.1 Передумови** **Server** отримує запит на нові дані. **X.2 Процес** **Server** обирає клієнтів, які мають отримати конкретні дані та надсилає їх. **X.3 Альтернативні потоки (обробка помилок)** При збою зв’язку виникає відповідне повідомлення у консолі та на екран. **X.4 Післяумови** **Customer** або **Teacher** отримує дані про дії інших користувачів.
UC11 Applying updates	**Customer або Teacher** зацікавлений в оновленні даних інших користувачів. **X.1 Передумови** **Customer** або **Teacher** отримує дані від **Server.** **X.2 Процес**

	Після отримання даних додаток **Customer або Teacher** встановлює позиції «аватарам» інших користувачів, або запускає відповідні події **X.3 Альтернативні потоки (обробка помилок)** При збою зв’язку виникає відповідне повідомлення у консолі та на екран. **X.4 Післяумови** Зміни внесені іншими користувачами відображаються на екрані **Customer або Teacher**