Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Моделі комунікацій учасників освітнього процесу та їх програмна реалізація

Виконав: студент 4 курсу
групи 1ПІ-18б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Козлюк Я.В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ
Коваленко О.О.
(прізвище та ініціали)

Рецензент: к.т.н., ст. викл. каф. КН
Озеранський В.С.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«____» _____ 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
25 березня 2022 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Козлюку Ярославу Віталійовичу

1. Тема роботи – «Моделі комунікацій учасників освітнього процесу та їх програмна реалізація»

Керівник роботи: Коваленко Олена Олексіївна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 року № 66

2. Строк подання студентом роботи 13 травня 2022 р.

3. Вихідні дані до роботи: середовище моделювання: Lusid Chart; середовище розробки Visual Studio Code, мови розробки JavaScript, мова розмітки HTML, мова стилю CSS, операційна система – Windows 10.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану питання та постановка задачі дослідження; розробка структури та алгоритмів програмного продукту; розробка та аналіз моделей комунікацій; реалізація програми комунікації учасників освітнього процесу; тестування програмного додатку, висновки; список використаних джерел, додатки, графічна частина.

5. Перелік графічного матеріалу: демонстрація аналогів, інтерфейс додатку, блок-схеми функціонування програмного продукту, алгоритми роботи додатку та тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Коваленко О. О., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки та постановка задачі	26.03.2022-16.04.2022	Вик.
2	Розробка моделей та алгоритмів програми	17.04.2022 – 01.05.2022	Вик.
3	Розробка графічного інтерфейсу програмного додатку	02.05.2022 – 15.05.2022	Вик.
4	Розробка програмного продукту	16.05.2022-25.05.2022	Вик.
5	Тестування роботи системи	26.05.2022 – 02.06.2022	Вик.
6	Оформлення матеріалів до захисту БДР	03.06.2022 – 10.06.2022	Вик.

Студент _____ **Козлюк Я. В.**

(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Коваленко О. О.**

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 52 сторінок формату А4, на яких є 42 рисунки, 3 таблиці, список використаних джерел містить 21 найменування.

У бакалаврській дипломній роботі проведено детальний аналіз моделей комунікацій учасників освітнього процесу та їх програмних реалізацій.

Запропоновано використовувати циклічну або циркулярну модель комунікації з використанням модуля обміну повідомленнями.

Розроблено модуль для обміну повідомленнями між користувачами з використанням мови програмування JavaScript, бібліотеки Socket.IO та середовища розробки Visual Studio Code.

Розроблено програмний додаток, що забезпечує комунікацію учасників освітнього процесу. При розробці використано мови програмування JavaScript, HTML, CSS, технології Express, Angular та середовище розробки Visual Studio Code.

Отримані в бакалаврській дипломній роботі результати можна використати для побудови систем комунікації користувачів.

Ключові слова: моделі комунікації, циклічна модель, веб-застосунок, JavaScript, студентський чат.

ABSTRACT

The bachelor's thesis consists of 52 A4 pages, which have 42 figures, 3 tables, the list of sources used contains 21 items.

In the bachelor's thesis, a detailed analysis of communication models of participants in education and their program implementations was carried out.

Usage of a cyclic or circular communication’s model with the messaging module is proposed.

The module for messaging between users was developed using JavaScript programming language, Socket.IO library and Visual Studio Code development environment.

A software that provides communication between participants in the educational process has been developed. JavaScript, HTML, CSS programming languages, Express, Angular technology and Visual Studio Code development environment were used in the development.

Results obtained in the bachelor's thesis can be used to build user communication systems.

Keywords: communication models, cyclic model, web application, JavaScript, student chat.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Актуальність дослідження моделей комунікацій збільшується з кожним днем не тільки тому що до глобальної мережі інтернету приєднуються все нові користувачі, а і тому що Інтернет стає середовищем здійснення різноманітних процесів – від навчання до створення нових продуктів та послуг.

Комунікація між користувачами є дуже важливим і необхідним процесом у функціонуванні різних інтернет ресурсів. Наприклад, для уточнення певної інформації про товар, користувач надсилає повідомлення консультанту в чаті інтернет-магазину, невдоволений глядач відеоролика залишає негативний коментар, а спільнота програмістів обмінюється порадами і досвідом на тематичних форумах.

Різноманітні моделі комунікацій також потрібні для здійснення продуктивного управління в корпоративних і локальних мережах [1]. Так, наприклад, менеджер відділу може розсилати завдання для своїх підлеглих в робочому чаті організації.

Наявність різних комунікаційних моделей є надзвичайно важливою і в освітньому процесі, адже його учасникам необхідно ефективно обмінюватись інформацією для максимальної продуктивності навчання.

Особливо актуальною дана проблема стала під час пандемії коронавірусу та в умовах воєнного часу. Більшість освітніх установ були вимушені повністю перейти на дистанційне навчання, що значно збільшило попит на платформи для дистанційної освіти. Тому аналіз моделей комунікацій та розробка власної програмної реалізації застосунку для покращення взаємодії учасників освітнього процесу є актуальною.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення ефективності та зручності освітнього процесу за рахунок впровадження власного програмного модуля для комунікації освітян, який базується на використанні обміну повідомлень в режимі реального часу.

Задачі дослідження

1. провести аналіз моделей комунікацій та визначити найбільш ефективний підхід до їхньої програмної реалізації;
2. розробити та провести аналіз варіативних моделей комунікацій освітнього процесу;
3. розробити загальну модель системи комунікацій між учасниками освітнього процесу;
4. виконати аналіз моделей та визначити модуль для програмної реалізації;
5. розробити алгоритм для обміну повідомлень між користувачами в режимі реального часу;
6. розробити зручний та інтуїтивний графічний інтерфейс користувача для взаємодії із логікою додатку;
7. розробити веб застосунок для комунікації учасників освітнього процесу;
8. провести тестування розробленого додатку.

Об’єкт дослідження – процес комунікації учасників освітнього процесу.

Предмет дослідження – методи та засоби моделювання та програмної реалізації моделей комунікації освітян.

Методи дослідження. У процесі досліджень використовувались наступні методи:

- теорія інформації та методи моделювання;
- теорія алгоритмів і структур даних для розробки програмного забезпечення;
- методи проектування структури, сутностей та зв’язків в базах даних;
- методи побудови та взаємодії компонентів серверних веб-систем.

Наукова новизна отриманих результатів.

- подальшого розвитку отримав метод проектування програмних систем для реалізації моделей комунікацій між користувачами, у якому, на відміну від існуючих реалізацій використано циркулярну або циклічну модель комунікації, що призвело до швидшої та ефективнішої взаємодії учасників освітнього процесу.
- Запропоновано модель реалізації системи комунікації, що, на відміну від існуючих, містить блоки інтеграції для масштабування та запровадження в різних освітніх платформах.

Практичне значення отриманих результатів. Практичне значення роботи полягає у запровадженні розробленого програмного застосунку для комунікації між учнями чи студентами та викладачами як окремого модуля та можливостей його інтеграції до платформи дистанційного та змішаного навчання.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. В опублікованій науковій роботі [2], автору належать такі результати: розробка веб-застосунку для обміну повідомленнями між користувачами в режимі реального часу, що базується на роботі технології веб-сокетів; загальний аналіз моделей комунікацій між користувачами та методів проектування їх програмної реалізації.

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця 2022).

Публікації. Основні результати дослідження опубліковані в науковій роботі – в тезах доповіді на науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) [2].

1 АНАЛІЗ МОДЕЛЕЙ КОМУНІКАЦІЙ УЧАСНИКІВ ОСВІТНЬОГО ПРОЦЕСУ ТА ЇХ ПРОГРАМНИХ РЕАЛІЗАЦІЙ

1.1 Моделі комунікації

Комунікація між людьми як соціальне явище привела суспільство та людство в цілому до того рівня прогресу, на якому воно зараз знаходиться. Саме тому важко переоцінити значення різних комунікаційних моделей у всіх сферах людської діяльності.

Потреба в моделях комунікацій між користувачами та їх програмних реалізаціях в сфері цифрових технологій обумовлена активним розвитком глобальної мережі інтернет. Такі моделі широко розповсюджені як в приватному спілкуванні, що реалізується у вигляді різних веб-чатів та месенджерів, так і на спеціалізованих освітніх платформах.

В суспільстві сформувалося декілька основних видів комунікації (рис. 1.1). Вони функціонують у всіх сферах людського життя та визначаються базовими комунікативними моделями.

Рисунок 1.1 – Види комунікацій [3]

В соціології розглядають декілька моделей комунікацій [3], [4]. Серед них найвідомішими є:

- модель Лассвела, яка стала класичною в теорії масової комунікації;
- модель біхевіоризму, яка поставила в основу комунікації не мову як конструкцію чи систему, а самі мовні сигнали;
- модель Шеннона – Вівера, що розглядає надмірність комунікації в математичних та технічних аспектах;
- циклічна або циркулярна модель, яка є найближчою до звичайного двонаправленого процесу комунікації між людьми.

В рамках даного дипломного дослідження найбільше розглядалась саме циклічна модель. Вона була представлена У. Шраммом і Ч. Осгудом та полягає в двоканальній циклічній взаємодії її учасників [3], [4].

Цю взаємодію можна вважати саме циклічною, адже в ході комунікації джерело і одержувач періодично міняються ролями. Відповідно, в даній моделі комунікація вважається двостороннім процесом зв'язку між її учасниками, які в однаковій мірі взаємодіють між собою, обмінюючись різними комунікаційними сигналами або ж повідомленнями, в результаті чого комунікація перетворюється на потужний механізм зворотного зв'язку.

Таку модель можна вважати повноцінним діалогом. Тому її програмна реалізація виглядає найбільш доцільною для ефективної комунікації учасників освітнього процесу.

Існує багато можливих варіантів програмної реалізації різних комунікаційних моделей. Серед них можна виділити системи обміну повідомленнями, системи аудіо- та відео- зв’язку та файлообмінники. Оскільки найбільш розповсюдженим типом інформації в мережі є саме текстова інформація, а її обробка не займає багато часу та ресурсів комп’ютера, було вирішено реалізувати систему обміну текстовими повідомленнями у вигляді веб-застосунку студентського чату.

Системи обміну повідомлень – це найбільш доступні і затребувані засоби комунікації в інтернеті, а також в корпоративних та локальних мережах. Вони діляться на дві великі групи:

– Служби обміну повідомленнями в режимі офлайн (поштові системи e-mail);
– Служби обміну повідомленнями в режимі онлайн (Instant Messaging Service, IMS) [5].

Оскільки програмна реалізація має забезпечувати швидку, ефективну та безперебійну комунікацію учасників освітнього процесу, було вирішено розробити веб-застосунок у формі чату, що може підтримувати обмін повідомленнями в режимі реального часу (IMS).

1.2 Порівняльний аналіз аналогів

Представлені на ринку програмні реалізації моделей комунікацій зазвичай не розробляються конкретно під потреби учасників освітнього процесу, проте є й достатньо спеціалізованих систем. Для порівняльного аналізу були відібрані наступні програмні системи:

– Google classroom;
– JetIQ;
– Telegram;
– Google Meet.

Google Classroom (рис. 1.2) – це веб-сервіс, створений компанією Google з метою покращення ефективності навчання студентів та школярів.

Сервіс широко використовується навчальними закладами на повністю безкоштовній основі та допомагає значно прискорити процес створення, редагування і поширення файлів безпаперовим шляхом між учасниками освітнього процесу.

До переваг веб-сервісу можна віднести дуже зручний обмін файлами і можливість перевірки та оцінювання робіт студентів. Проте Google Classroom не має ефективної моделі обміну повідомленнями в режимі реального часу. Його система комунікації використовує коментарі до завдань, і за принципом роботи більше схожа на поштову систему.

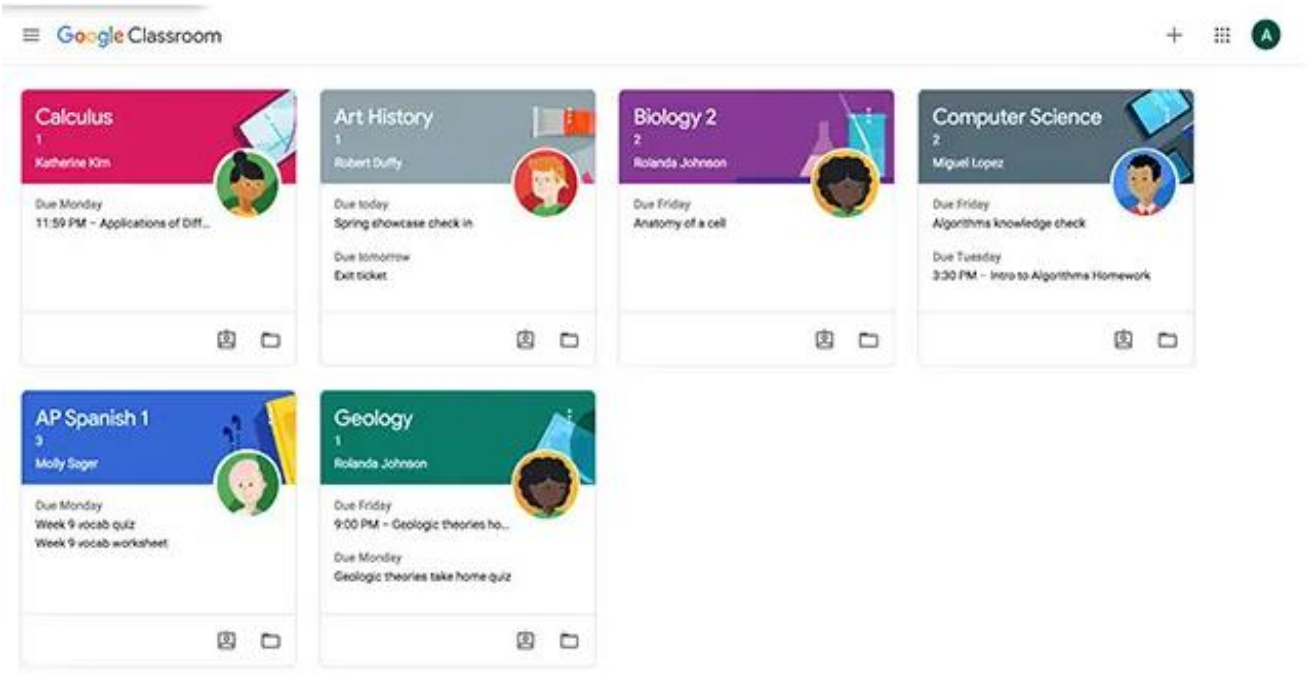

Рисунок 1.2 – Інтерфейс сервісу Google Classroom

JetIQ (рис. 1.3) – це сервіс, розроблений для покращення організації освітнього процесу у ВНТУ. Він має широкий функціонал, до якого входить обмін файлами, ведення журналів, тестування студентів та обмін повідомленнями.

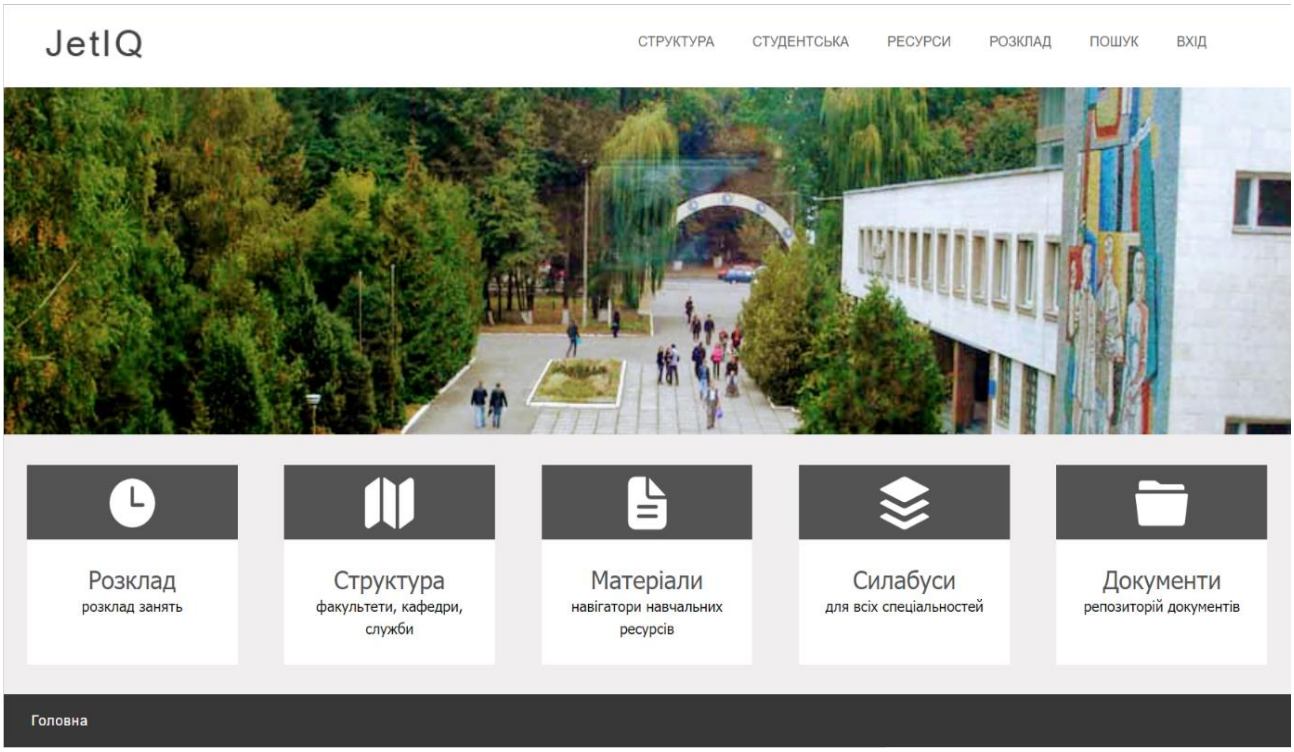

Рисунок 1.3 – Головна сторінка сайту JetIQ

До переваг сервісу можна віднести високу ступінь інтеграції в навчальний процес університету, що значно збільшує його ефективність.

Проте модуль обміну повідомленнями в даній програмній системі є дещо застарілим. Тут відсутній груповий або приватний чат в режимі реального часу, а користувачі при відправці повідомлення не можуть бачити інших учасників онлайн.

Telegram (рис. 1.4) – це хмарна служба обміну повідомленнями в режимі реального часу. Дана програмна система не розроблялася конкретно для комунікації учасників освітнього процесу, проте часто використовується в таких цілях.

До переваг служби належать наскрізне шифрування, обмін файлами різних типів, групові та особисті чати. Головним недоліком у використанні Telegram для освітнього процесу є його автономність, тобто для створення ефективного навчального комплексу потрібно використовувати багато різних сторонніх програм.

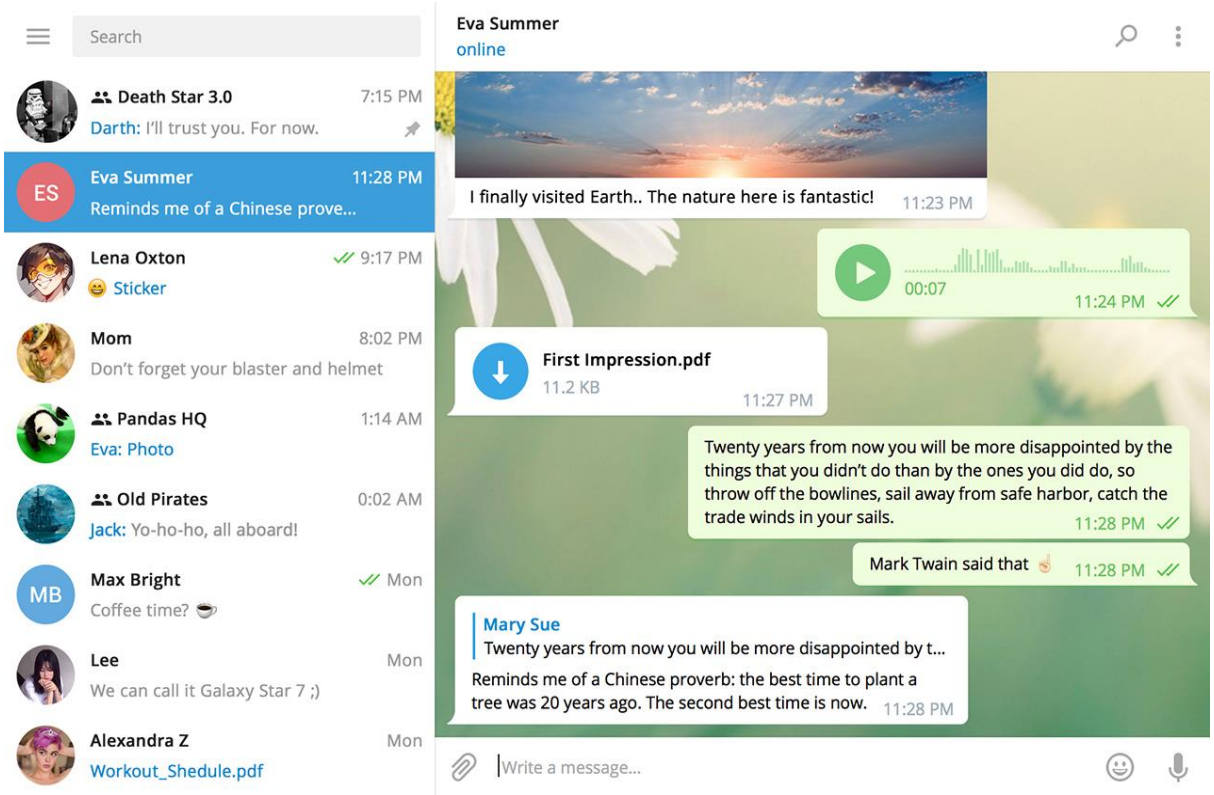

Рисунок 1.4 – Інтерфейс додатку Telegram

Google Meet (рис. 1.5) – це сервіс відеотелефонного зв'язку, розроблений компанією Google. Хоча він не є спеціалізованим для освітнього процесу, його часто використовують університети та школи для проведення дистанційних уроків.

Попри високу якість відеодзвінків та велику кількість можливих учасників, сервіс не зовсім підходить для організації комплексного освітнього процесу, адже має ті ж недоліки, що й Telegram та не є спеціалізованим.

Рисунок 1.5 – Інтерфейс сервісу Google Meet

Результати порівняння власного рішення з аналогами наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	Google classroom	JetIQ	Telegram	Google meet	Власний додаток
1	2	3	4	5	6
Обмін повідомленнями в режимі реального часу	-	-	+	+	+
Підтримка групової	+	+	+	+	+

комунікації					
Приватне листування за згодою користувача	-	-	-	-	+

Продовження таблиці 1.1

1	2	3	4	5	6
Збереження отриманих та відправлених повідомлень при наступній сесії програми	+	+	+	-	+
Наявність різних форм комунікацій	-	-	+	+	-
Можливість прямої інтеграції в освітні платформи	+	+	-	-	+
Загальна оцінка	50%	50%	66%	50%	83%

Таблиця порівняльних характеристик показала, що розробка програмного продукту є доцільною. Одержаний продукт зможе цілком покрити недоліки існуючих рішень та забезпечує кращу ефективність комунікації учасників освітнього процесу.

1.3 Методи реалізації програмних модулів комунікацій учасників освітнього процесу

Метод реалізації програмного модулю комунікації включає в себе:

1. обгрунтований вибір моделі комунікацій;
2. обгрунтований метод технологій реалізації;
3. визначені вимоги до програмного модулю.

Аналіз моделей комунікацій свідчить про те, що доцільно використати циркулярну модель, яка включає зворотній зв’язок і можливість формування послідовності дій кожним учасником. Але така модель повинна бути удосконалена для реалізації не тільки відносин один до одного, а і один до

багатьох, багато до одного. Для освітнього процесу пропонується схема комунікацій повідомлення – отримання – відповідь/участь у форумі.

Для розробки програмної реалізації моделей комунікацій учасників освітнього процесу може бути використано декілька методів. Серед них найбільш актуальними є методи використання V-подібної, інкрементної або водоспадної моделей розробки ПЗ [6].

V-подібна модель розробки передбачає покрокове розв'язання поставленої задачі з проведенням тестування на кожному етапі. Інкрементна методологія використовується в разі необхідності реалізації декількох варіантів готового програмного продукту, для подальшого вибору найбільш ефективного із них. Методологія використання водоспадної або каскадної моделі розробки ПЗ полягає в поетапній реалізації всіх модулів програмного продукту згідно з розробленим наперед детальним планом.

Оскільки власне програмне рішення не є надто масивним, був розроблений детальний план його реалізації, і, відповідно, була обрана каскадна модель розробки для основного функціоналу. Для подальших розробок доцільно використовувати гнучкі методології на основі Agile.

Після вибору методології розробки було розглянуто основні методи реалізації архітектури програмного рішення. Серед них найбільш актуальними є розробка власної бібліотеки для обміну повідомленнями, розробка окремого сервісу для інтеграції системи комунікації та розробка повноцінного веб-застосунку. Власна бібліотека могла б стати оптимальним рішенням, але вона має ряд незручностей в сфері імплементації. Розробка окремого сервісу з моделлю розповсюдження на API є ефективним рішенням для використання в існуючих платформах, проте вона не дасть повної гнучкості та масштабованості у проектуванні власної освітньої платформи.

Тому, врахувавши переваги та недоліки всіх методів, було вирішено використати розробку повноцінного веб-додатка, який міститиме всі необхідні модулі комунікації учасників освітнього процесу. Такий підхід дозволить

реалізувати весь запланований функціонал програмної системи та зробить можливим її подальше масштабування.

1.4 Постановка задач для програмної реалізації моделей комунікацій учасників освітнього процесу

Проаналізувавши питання розробки програмних реалізацій моделей комунікацій учасників освітнього процесу, було визначено завдання, які необхідно виконати для розробки програмного додатку:

- визначити найбільш ефективний підхід до програмної реалізації моделей комунікацій освітян;
- на основі визначеного методу розробити алгоритми для обміну повідомлень між користувачами в режимі реального часу;
- розробити зручний та інтуїтивний графічний інтерфейс користувача для взаємодії із логікою додатку;
- розробити веб застосунок для комунікації учасників освітнього процесу;
- провести тестування розробленого застосунку.

1.5 Висновки

У першому розділі було розглянуто стан питання моделей комунікацій та їх програмних реалізацій на сьогоднішній день.

Було проаналізовано існуючі аналоги та проведено їх порівняльний аналіз із власним програмним додатком. У результаті порівняння було доведено доцільність розробки власного програмного продукту.

Було проведено аналіз існуючих методів вирішення поставленої задачі. У результаті аналізу було обрано водоспадну методологію розробки, а також розробку архітектури у вигляді повноцінної веб-системи.

Сформульовано основні завдання, які необхідно виконати для розробки програмного додатку.

2 МОДЕЛІ КОМУНІКАЦІЙ ОСВІТНЬОГО ПРОЦЕСУ

2.1 Аналіз даних та варіативні моделі освітнього процесу

Основою роботи будь-якого програмного продукту є дані. Між різними модулями програми відбувається постійний обмін інформацією. Інколи дані генеруються алгоритмами самої програмної системи, проте зазвичай вони надходять від користувачів або спеціалізованих баз даних.

Під час взаємодії з графічним інтерфейсом програми, до її алгоритмів поступає вся надана користувачем інформація. Далі відбувається її обробка відповідно до визначених логікою програми алгоритмів. Механізми обробки даних залежать від специфіки задачі, для якої вона проводиться. Після всіх маніпуляцій дані повертаються вже в обробленому форматі до користувача і відтворюються з допомогою графічного інтерфейсу.

В розробленій програмній реалізації моделей комунікацій головними даними є повідомлення користувачів, тобто учасників освітнього процесу. Кожне відправлене повідомлення прив’язується до конкретного користувача і відображається від його імені в одержаних повідомленнях інших учасників. Всі повідомлення зберігаються в підключеній до програми нереляційній базі даних. В ній містяться декілька основних сутностей, таких як користувачі, повідомлення та підключення. Діаграма бази даних представлена на рисунку 2.1.

Кожна сутність користувача зберігає ім’я, пароль, групу, телефон, емейл та список друзів. Майже всі дані вказуються при реєстрації. З їх допомогою відбувається автентифікація та ідентифікація користувачів у веб-застосунку. Додаткової обробки зазнає пароль, адже він шифрується на стороні сервера в цілях безпеки. Атрибут списку друзів зберігає масив всіх користувачів, яких було добавлено в друзі. Сутність повідомлень містить сам текст повідомлення, ім’я відправника, ім’я одержувача та дату й час відправки. Сутність підключень

містить ім’я та унікальний ідентифікатор підключення і служить для відображення користувачів онлайн.

Рисунок 2.1 – Діаграма бази даних

При подальшому масштабуванні веб-застосунку можлива реалізація модуля обміну файлами. Такий модуль потребує інакшої обробки даних. З допомогою вбудованих бібліотек браузера, програма перетворює файл будь-якого типу на BLOB-об’єкт (рис. 2.2) та додає його в базу даних. Тип BLOB – це набір бінарних даних, що розглядаються як єдиний, неперервний об'єкт [7]. Обрана для власного додатку нереляційна база MongoDB забезпечує ефективне та незатратне зберігання BLOB-об’єктів. Після всіх маніпуляцій з даними, програма генерує URL посилання для завантаження кожного файлу.

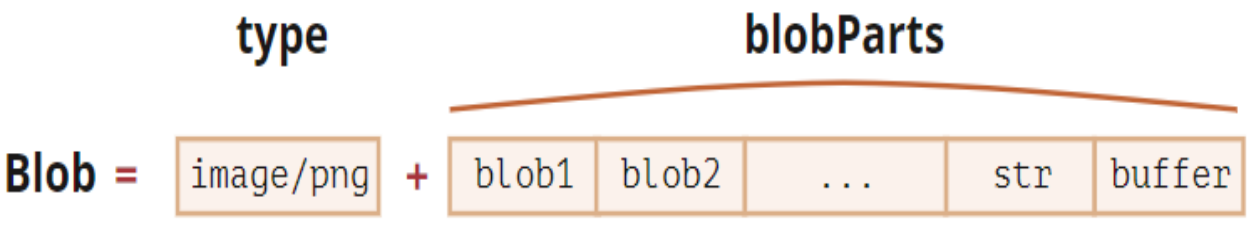

Рисунок 2.2 – Представлення BLOB-об’єкта

Можлива також й реалізація комунікації учасників освітнього процесу в форматі відео- та аудіо- зв’язку. Для цього методу програма повинна отримувати багато відзнятих веб-камерою кадрів та перетворювати їх на стабільний відеопотік, який транслюватиметься іншим користувачам. Відповідно, знадобляться методи обробки графічних даних.

Таким чином розроблений веб-застосунок забезпечує постійну циркуляцію даних, які необхідні для реалізації моделей комунікацій між учасниками освітнього процесу.

Для розробки програмної реалізації необхідно спочатку створити та детально проаналізувати самі моделі комунікацій учасників освітнього процесу [8].

В ході виконання даної бакалаврської дипломної роботи були створені циклічні моделі комунікацій, які реалізуються у вигляді модулів групового та приватного чатів у веб-застосунку.

Проте неможливо розглядати моделі комунікацій окремо від загальної моделі освітнього процесу. Саме тому було прийняте рішення розробити схему моделі організації освітнього процесу (рис. 2.3), яка містила б в собі й комунікаційні моделі. Серед складових даної схеми є ОМН, тобто організаційна модель навчання, яка слугує для вирішення питань ролей користувачів, облікових записів, доступу, управління, тощо. Ця модель відповідає за організацію навчального процесу. Вона також містить в собі МОС, тобто модель освітньої спільноти, яка відповідає за взаємодію через Інтернет. Модель викладання (МВП) передбачає створення певного навчального середовища (курсів, уроків, наукових робіт, тощо). Під час навчання та проходження тестування (МКД) програма визначає рівень знань студента (МС). Викладач (МВ) впливає на організацію процесів та створення ресурсів у моделі викладання. Модель комунікації (МК) забезпечує ефективну комунікацію між моделями студентів та моделями викладачів.

Розроблена модель освітнього процесу передбачає великий простір для масштабування та розвитку програмної реалізації. Її подальша розробка можлива в наступних наукових роботах, наприклад, в магістерській дипломній роботі.

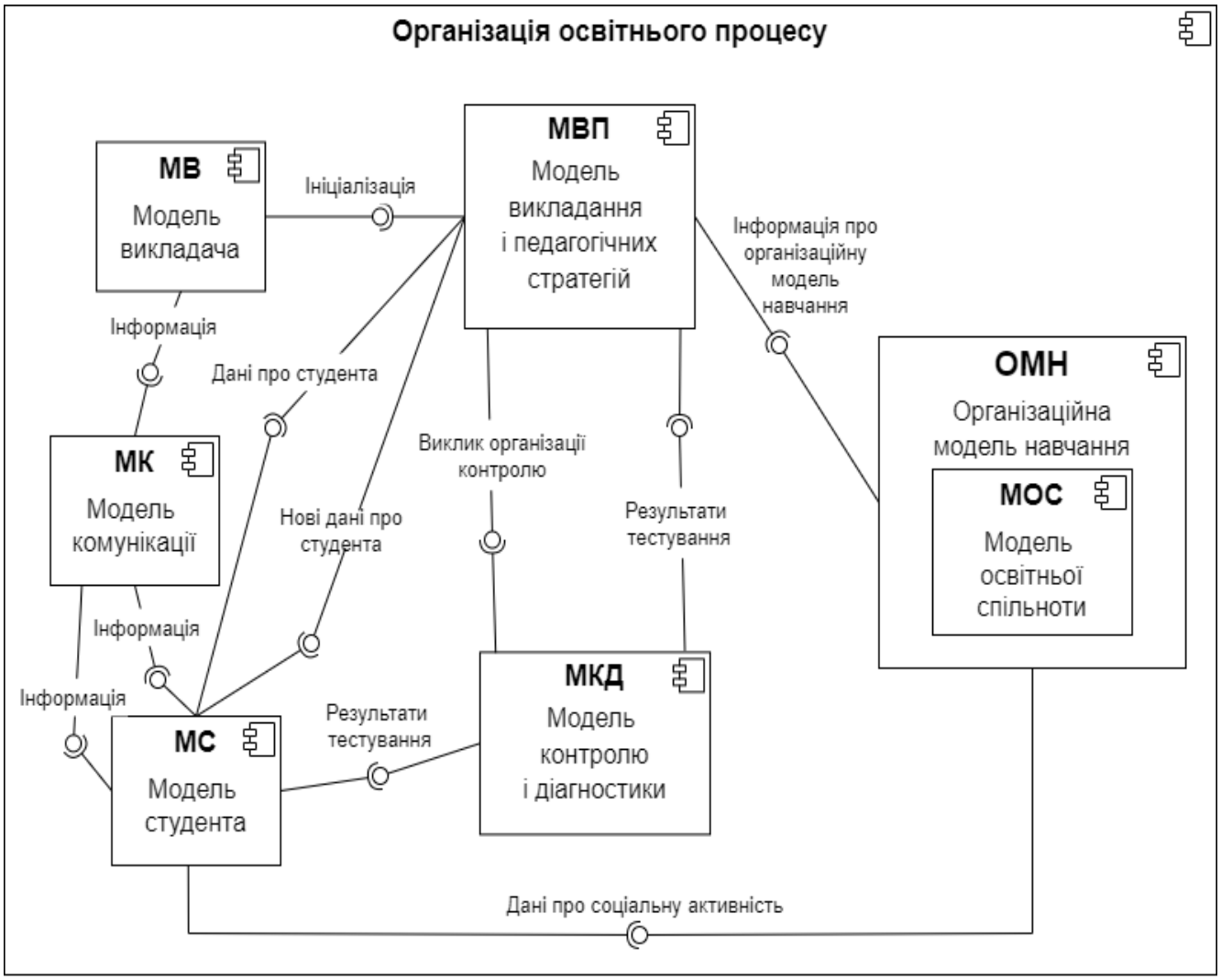

Рисунок 2.3 – Модель організації освітнього процесу

Після створення загальної схеми було розглянуто деякі можливі моделі комунікацій учасників освітнього процесу. Серед основних комунікаційних моделей було виділено модель Лассвела, модель Шеннона – Вівера та циклічну модель.

Модель Лассвела [9] описує загальну схему комунікації, яка також включає аспекти масової комунікації, за що дана модель стала популярною. Вона визначає відправника, повідомлення, канал, отримувача, зворотній зв’язок та взаємодію між цими компонентами. Отримувачем повідомлення може бути група людей, що

й задає масовість комунікації. Модель також передбачає різні формати повідомлення, каналів передачі та зворотного зв’язку. Схема моделі Лассвела представлена на рисунку 2.4.

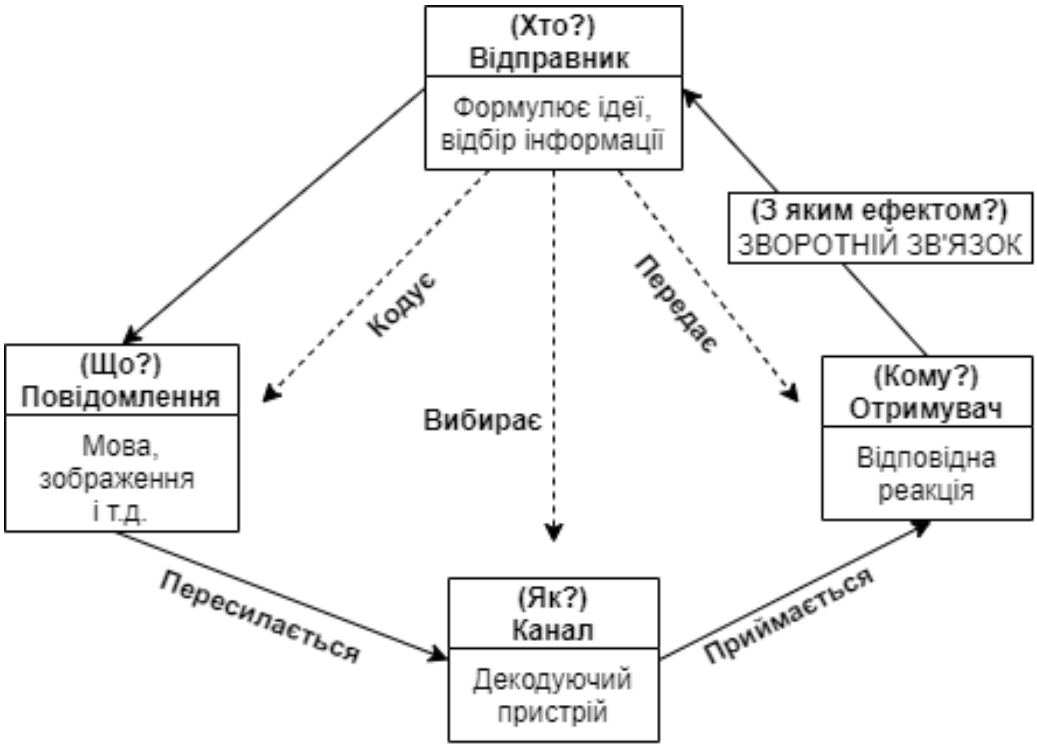

Рисунок 2.4 – Загальна схема моделі Лассвела

Модель Шеннона – Вівера [10] базується на математичній теорії комунікації. Вона також виділяє аспекти шумів та перешкод для комунікації і розглядає надмірність інформації, тобто багаторазове повторення сигналу для подолання цих перешкод. Модель передбачає використання різних каналів зв’язку, що робить її багатоканальною. Схема моделі Шеннона – Вівера представлена на рисунку 2.5.

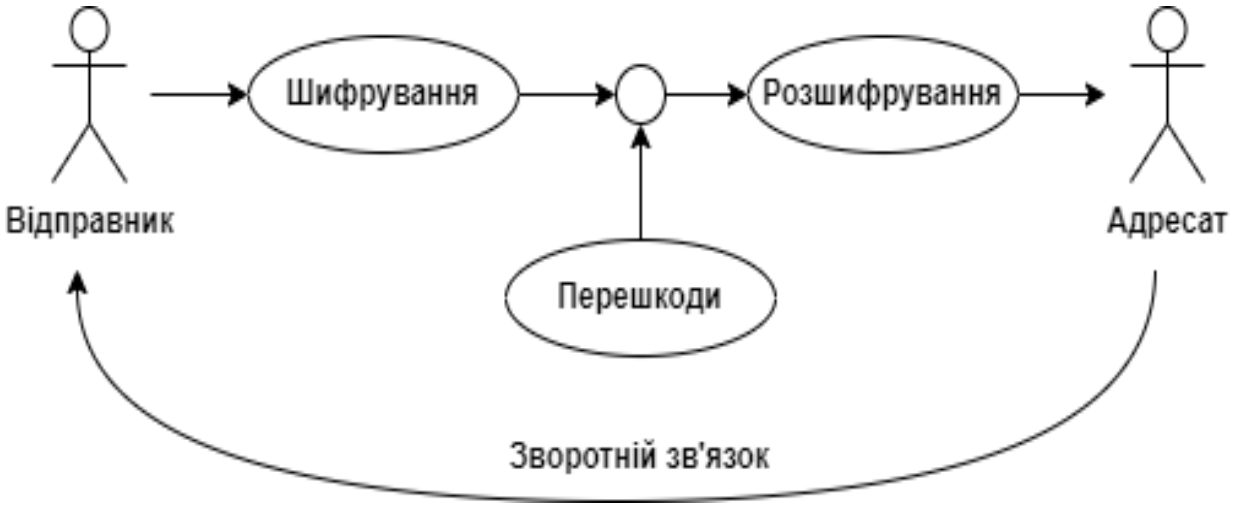

Рисунок 2.5 – Схема моделі Шеннона – Вівера

Циклічна модель У. Шрамма [11] є найбільш схожою на звичайний двонаправлений процес комунікації між людьми. Згідно цієї моделі, учасники можуть періодично мінятися ролями, що й визначає її циклічність.

Такий підхід забезпечує постійний та ефективний обмін комунікативними сигналами між учасниками комунікаційного процесу. Модель передбачає використання різних каналів передачі та форматів повідомлень, а також може використовуватись в груповій комунікації.

Зважаючи на всі плюси циклічної моделі, було прийнято рішення розробити її програмну реалізацію для комунікації учасників освітнього процесу. Загальна схема циклічної моделі представлена на рисунку 2.6.

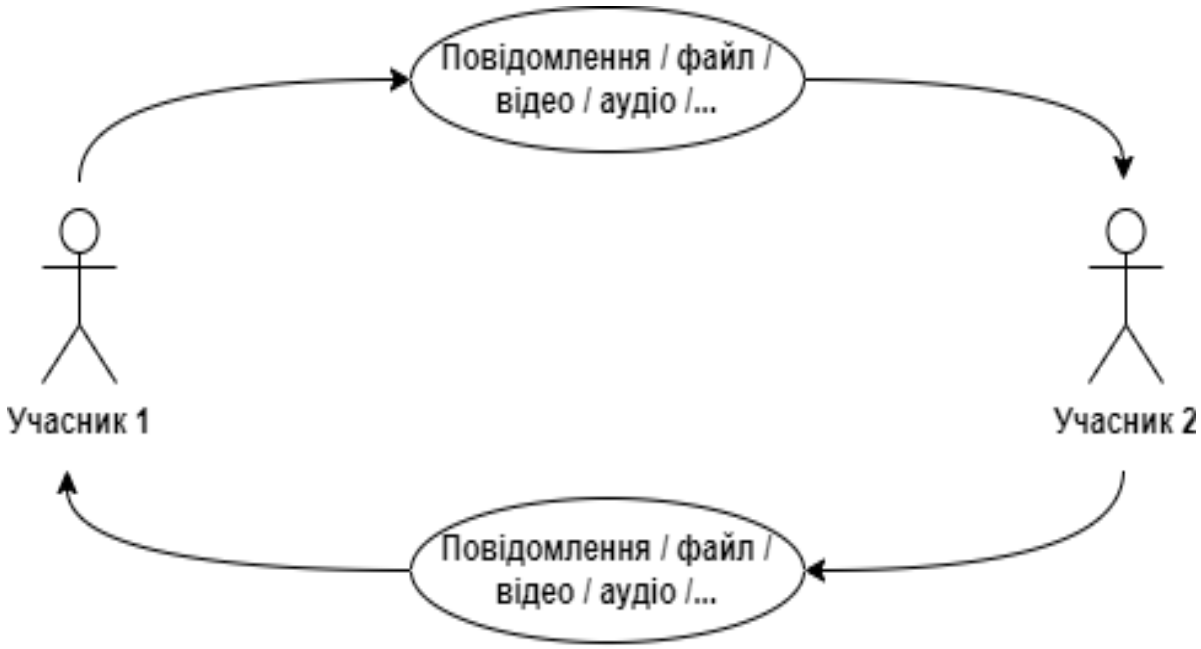

Рисунок 2.6 – Загальна схема циклічної моделі

Даний варіант моделі реалізовується в модулі приватного чату розробленого веб-застосунку. Комунікаторами є два користувачі, тобто учасники освітнього процесу, а каналом зв’язку виступає розроблений модуль обміну повідомленнями.

На даному етапі розробки система підтримує тільки обмін текстовими повідомленнями, проте в подальшому можуть бути реалізовані модулі обміну файлами та модулі відео і аудіо зв’язку.

В модулі групового чату веб-застосунку також використовується варіант групової циклічної моделі. Її схема представлена на рисунку 2.7.

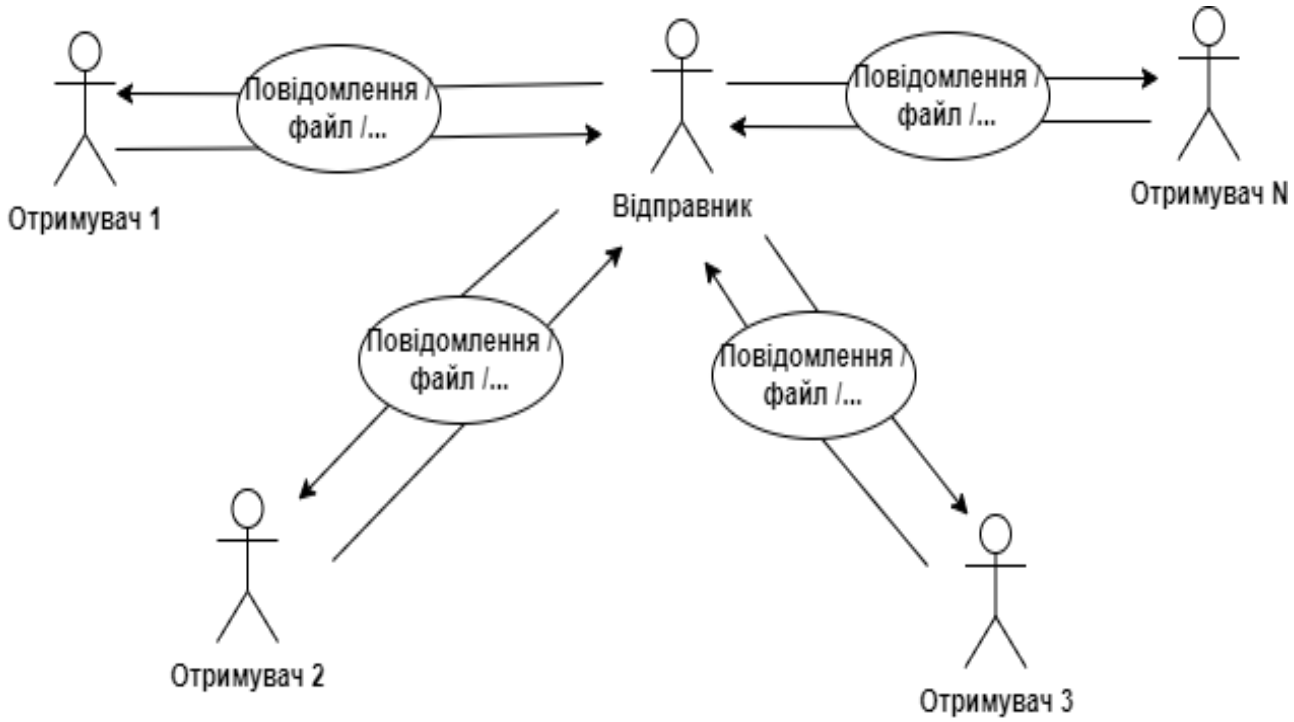

Рисунок 2.7 – Схема циклічної моделі для групової комунікації

Таким чином було розроблено та детально проаналізовано загальну модель організації освітнього процесу. Було передбачено широкий потенціал для масштабування в подальших наукових роботах. Також було розроблено та проаналізовано різні моделі комунікацій учасників освітнього процесу та обрано циклічну модель комунікації.

2.2 Загальна модель та структура інтерфейсу веб-застосунку

Графічний інтерфейс є дуже важливим елементом в процесі взаємодії користувача із логікою програми. На сьогодні існує декілька основних видів інтерфейсів [12]:

- Інтерфейс командного рядка;
- Інтерфейс на основі меню;
- Графічний інтерфейс користувача.

При проектуванні власного програмного рішення було обрано реалізацію саме графічного інтерфейсу користувача.

Інтерфейс основного вікна програми умовно поділений на три частини. Зліва знаходиться область групового чату (рис. 2.8), де всі студенти можуть обмінюватись повідомленнями.

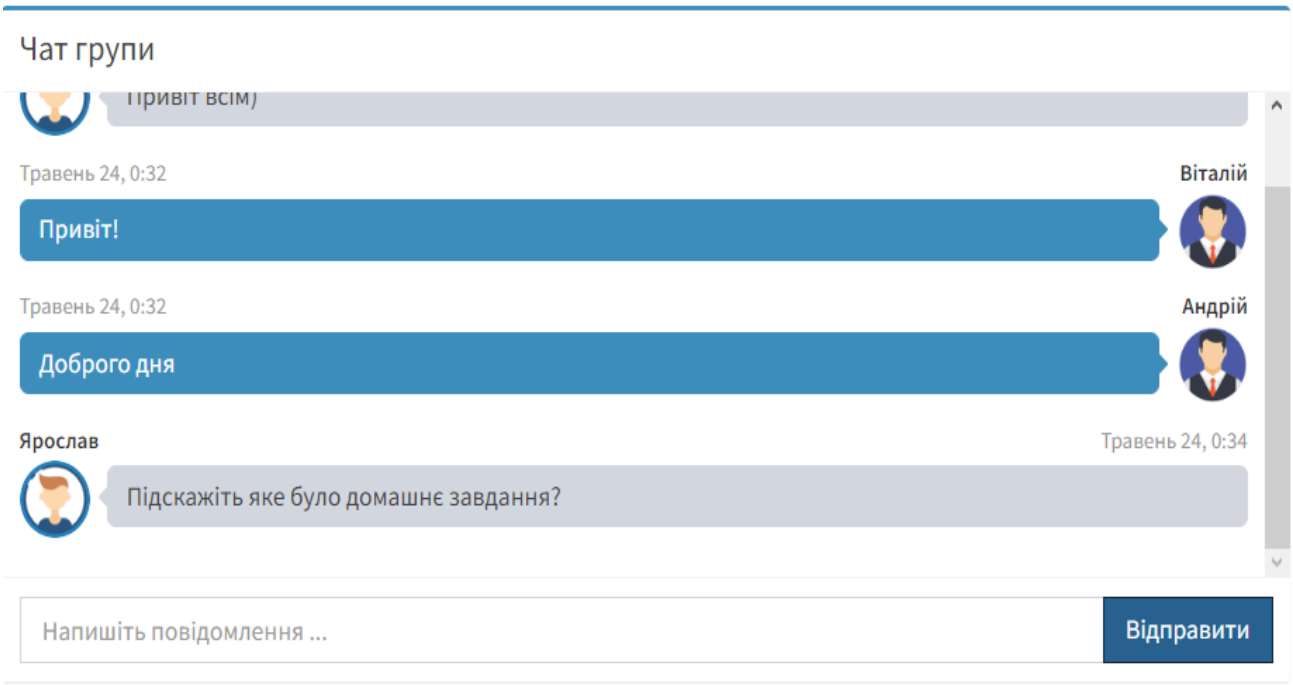

Рисунок 2.8 – Область групового чату

Запропонований чат дозволить реалізувати модель один до багатьох, а також дасть можливість написати повідомлення за моделлю один до одного.

Такі чати доцільно створювати до дисципліни (аналогічно як в системі JetIQ), а також для визначених цільових груп – академічних груп, спільнот для підготовки до змагань, команд, що працюють над одним проектом тощо.

В нижньому правому куті розміщується область відправки приватних повідомлень (рис. 2.9). Вона відображається тільки після того, як користувач надіслав успішний запит на листування іншому користувачу. Цей чат реалізовує модель один до одного і використовується для конфіденційного листування.

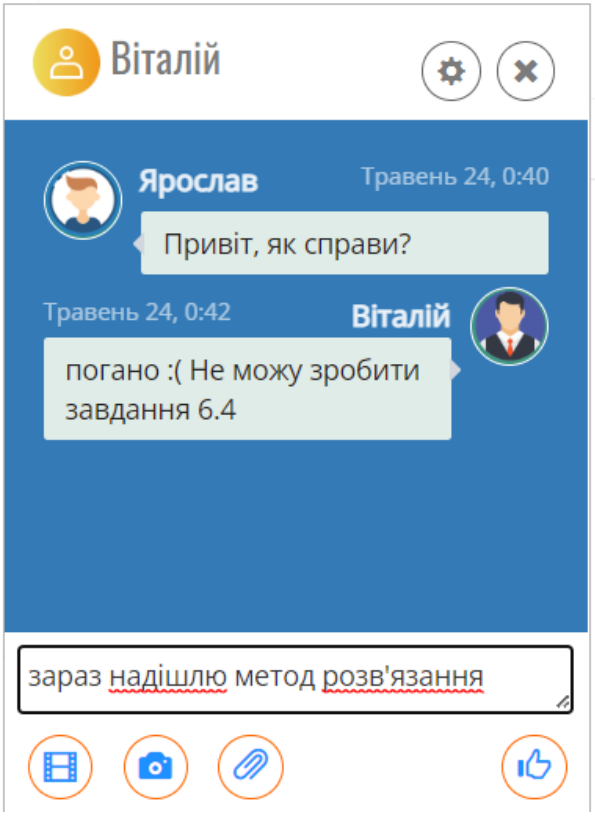

Рисунок 2.9 – Область приватних повідомлень

Праворуч розміщується область відображення друзів та студентів онлайн (рис. 2.10). Кожен підключений студент автоматично добавляється в розділ студенти онлайн. В розділі друзів відображаються користувачі, які прийняли запит на листування.

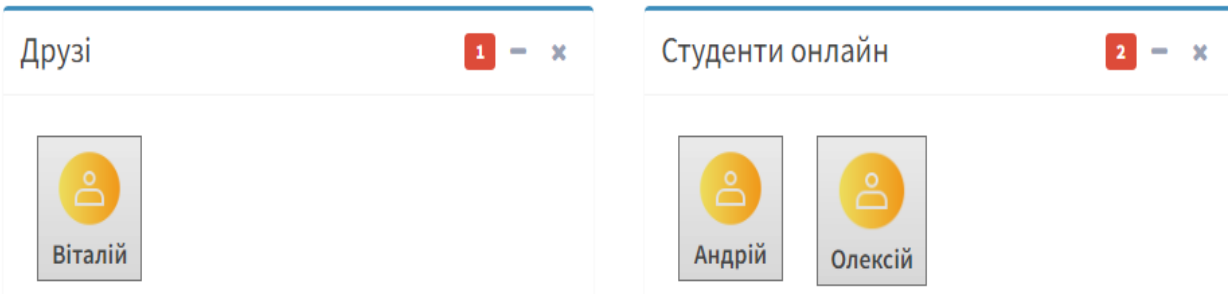

Рисунок 2.10 – Область відображення друзів та студентів онлайн

Загальний вигляд головного вікна розробленого веб-застосунку представлений на рисунку 2.11.

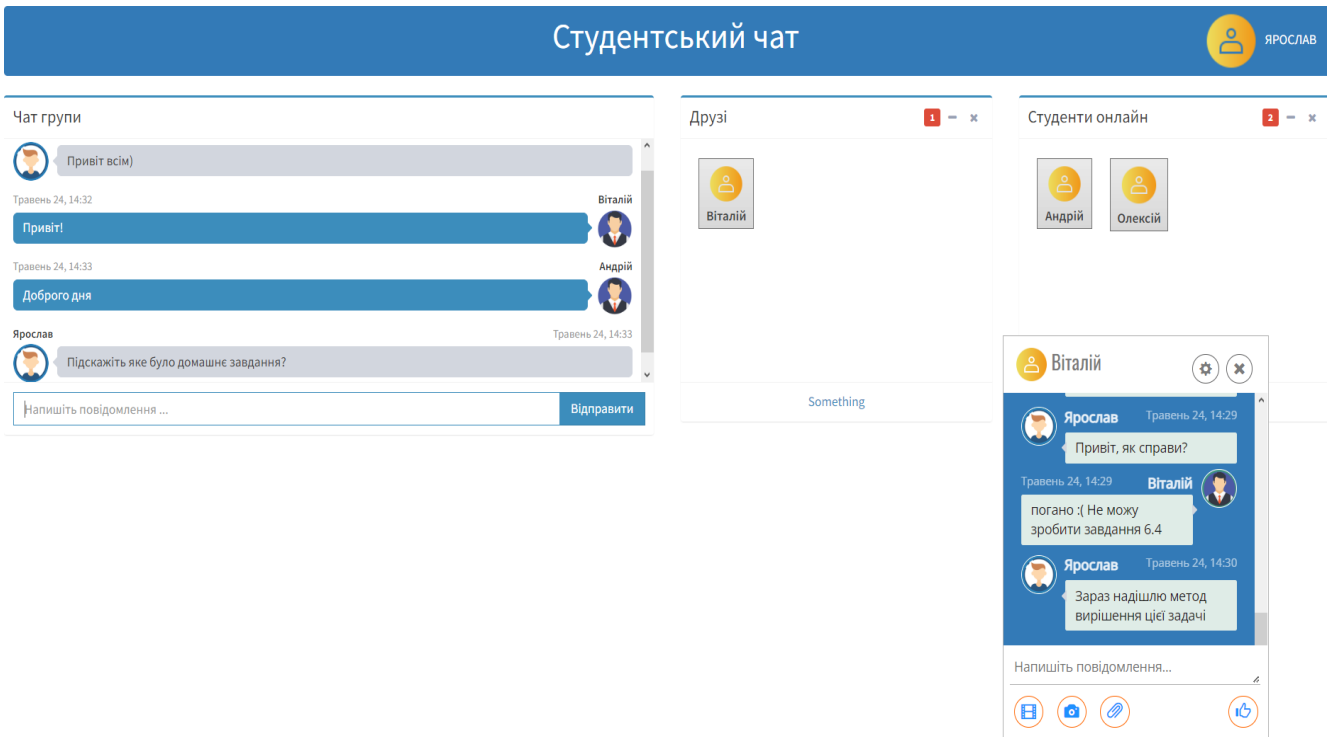

Рисунок 2.11 – Головне вікно веб-застосунку

Також розроблений застосунок містить декілька службових вікон та два вікна з формами реєстрації і авторизації користувачів в системі. Дані форми представлені на рисунку 2.12.

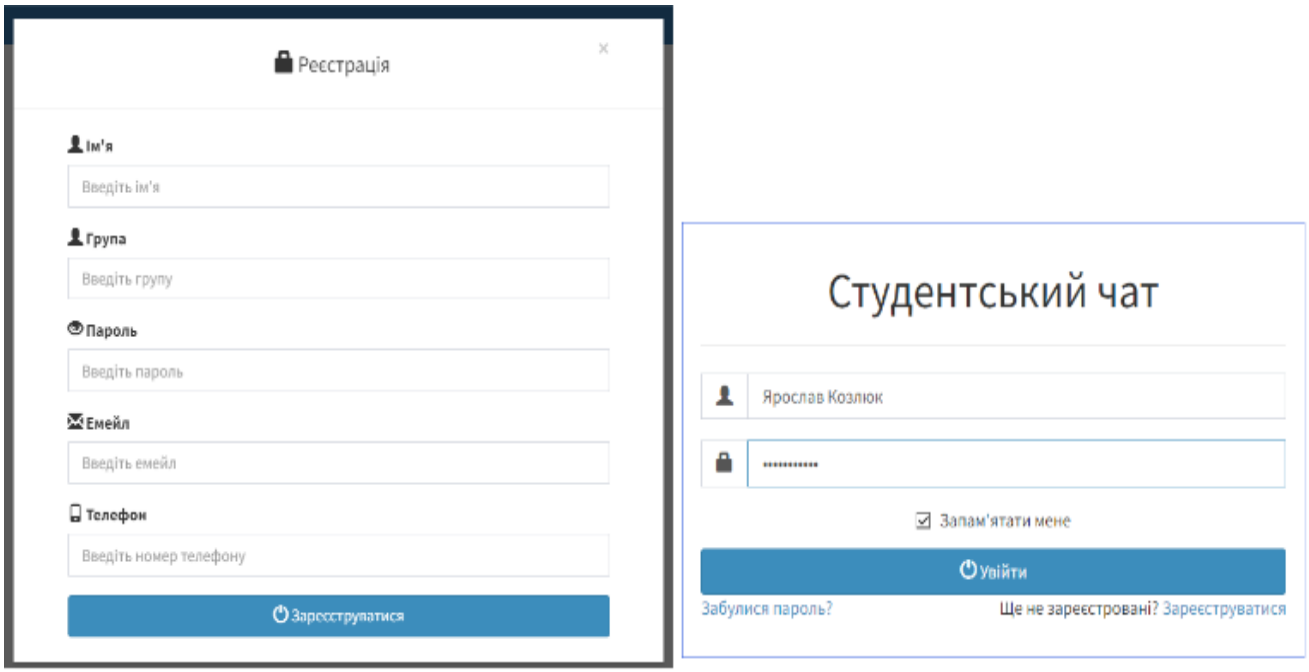

Рисунок 2.12 – Форми реєстрації та авторизації

Графічний інтерфейс програми був створений в середовищі розробки Visual Studio Code з використанням методології AJAX [13].

AJAX – це не технологія сама по собі, а термін, який описує новий підхід до використання існуючих технологій. В розробці програмної реалізації AJAX включає: мову розмітки HTML, мову стилізації CSS, скриптову мову JavaScript, об’єктну модель документа та об’єкт XMLHttpRequest. Об’єднавши дані технології разом з допомогою методології AJAX, було досягнуто здійснення швидкого оновлення інтерфейсу без жодної необхідності повного перезавантаження веб-сторінки.

2.3 Розробка алгоритмів роботи застосунку

Для розробки програмної реалізації спочатку потрібно розглянути загальний алгоритм [14] роботи додатку, який відображає основні процеси функціонування програми.

Алгоритм починається із встановлення з’єднання з сервером. Залежно від успішності попередньої операції, виконання програми ділиться на дві гілки. В разі невдачі, якщо час очікування не вийшов, програма знову спробує встановити з’єднання, інакше виведе помилку і виконання закінчиться. В разі успіху, користувачу потрібно буде ввійти в систему. Якщо він не зареєстрований, відкриється вікно реєстрації.

Після успішної автентифікації, відкривається головне вікно програми, де користувач може обрати груповий або приватні чати. В разі вибору групового чату, користувач вводить повідомлення і надсилає його.

За умови вибору приватного чату, необхідна наявність підтверджених запитів на листування. Якщо їх не має, користувач надсилає запит іншому користувачу і чекає відповіді від сервера.

Якщо ж підтверджені запити існують, то користувач обирає один із них, вводить повідомлення та надсилає його. Також програма постійно чекає

повідомлень від сервера та відображає їх, в разі наявності. Якщо користувач виходить з програми, з’єднання із веб-сервером переривається і програма завершує свою роботу.

Блок-схема алгоритму зображена на рисунку 2.13.

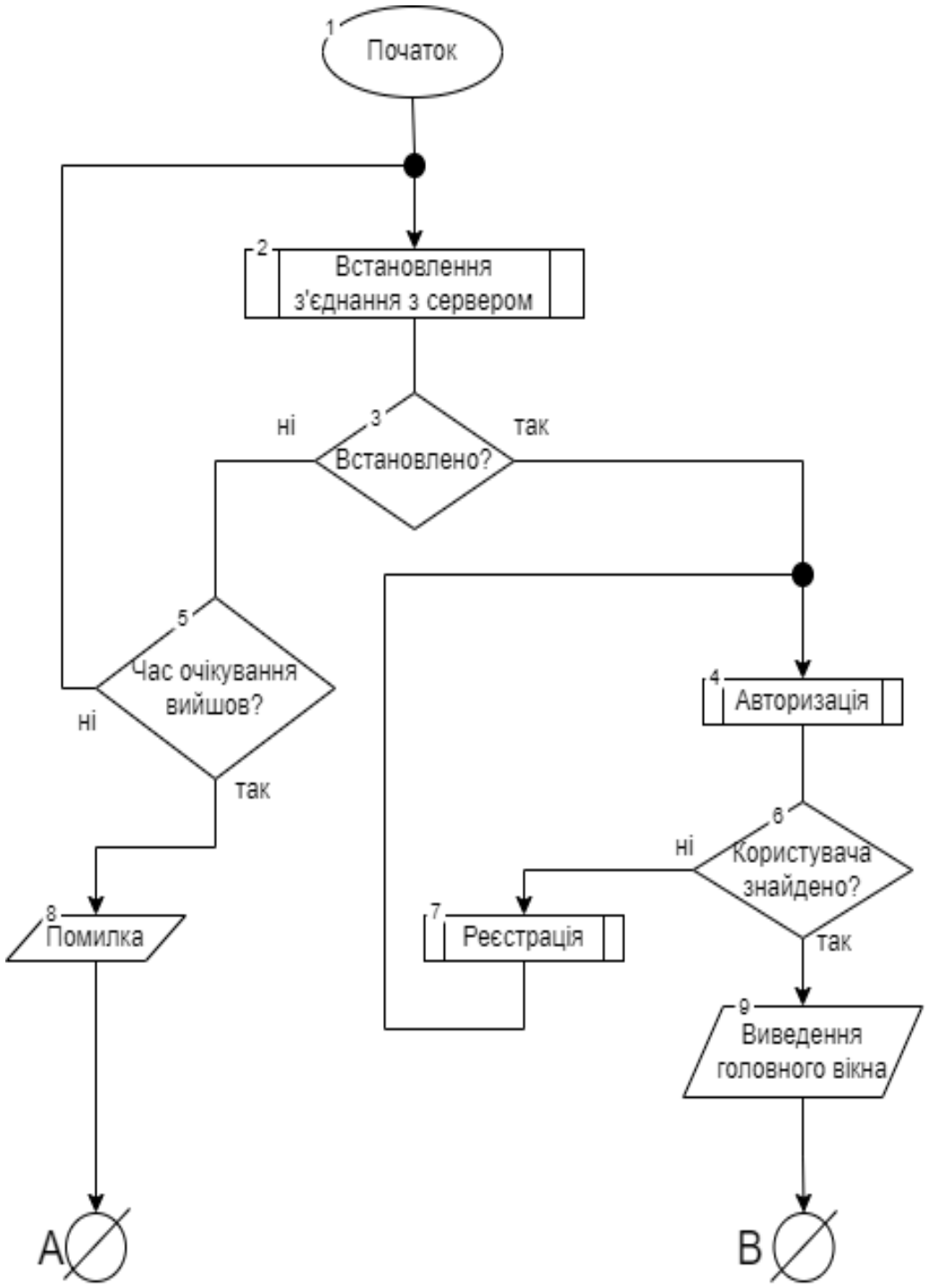

Рисунок 2.13 – Блок-схема загального алгоритму роботи додатку

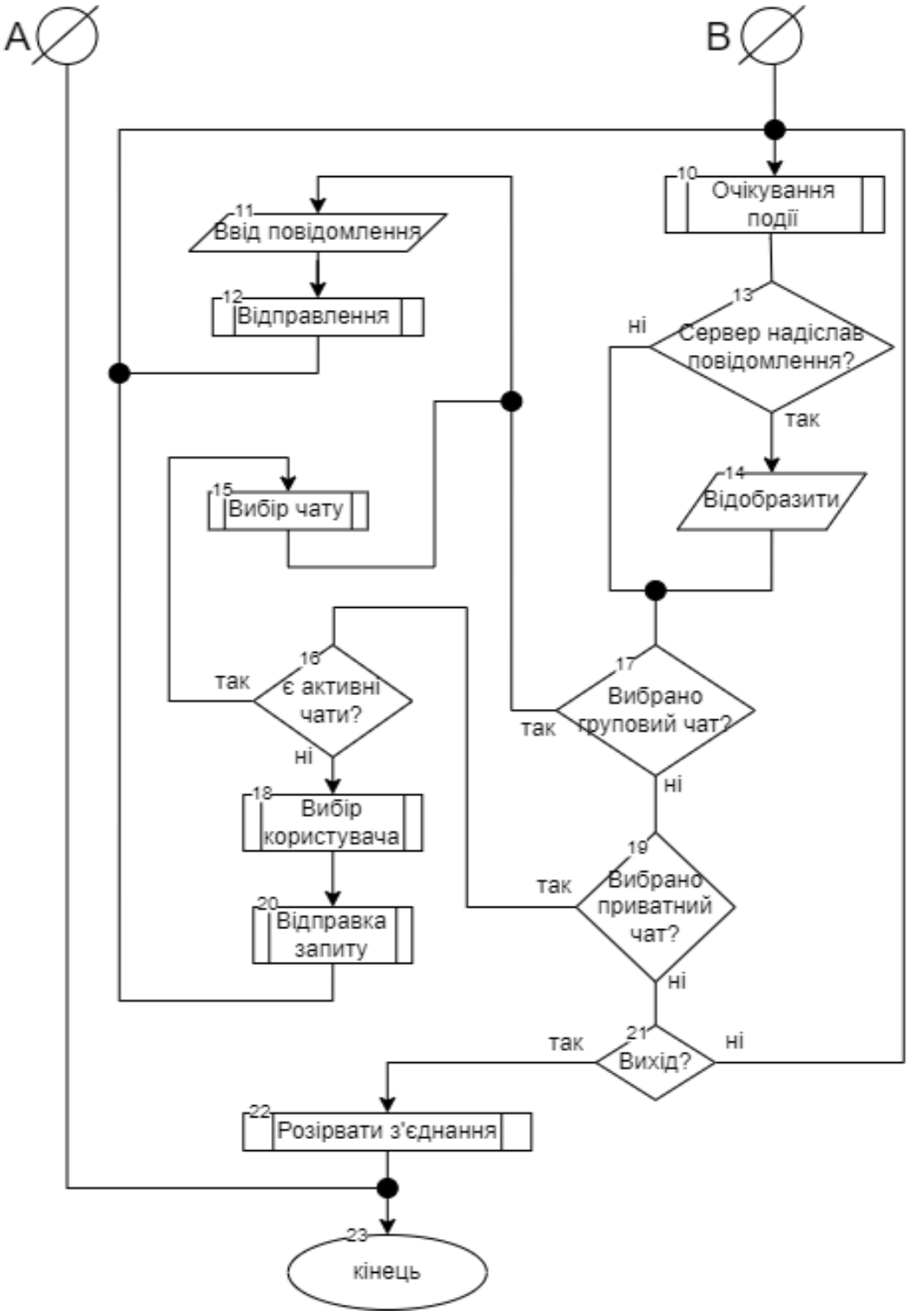

Рисунок 2.14 – Продовження блок-схеми

Наступним кроком є розробка та детальний опис алгоритму реєстрації і авторизації. Даний алгоритм відіграє важливу роль в процесі створення нових користувачів та надання доступу до веб-застосунку існуючим користувачам. Блок-схема даного алгоритму наведена на рисунку 2.15.

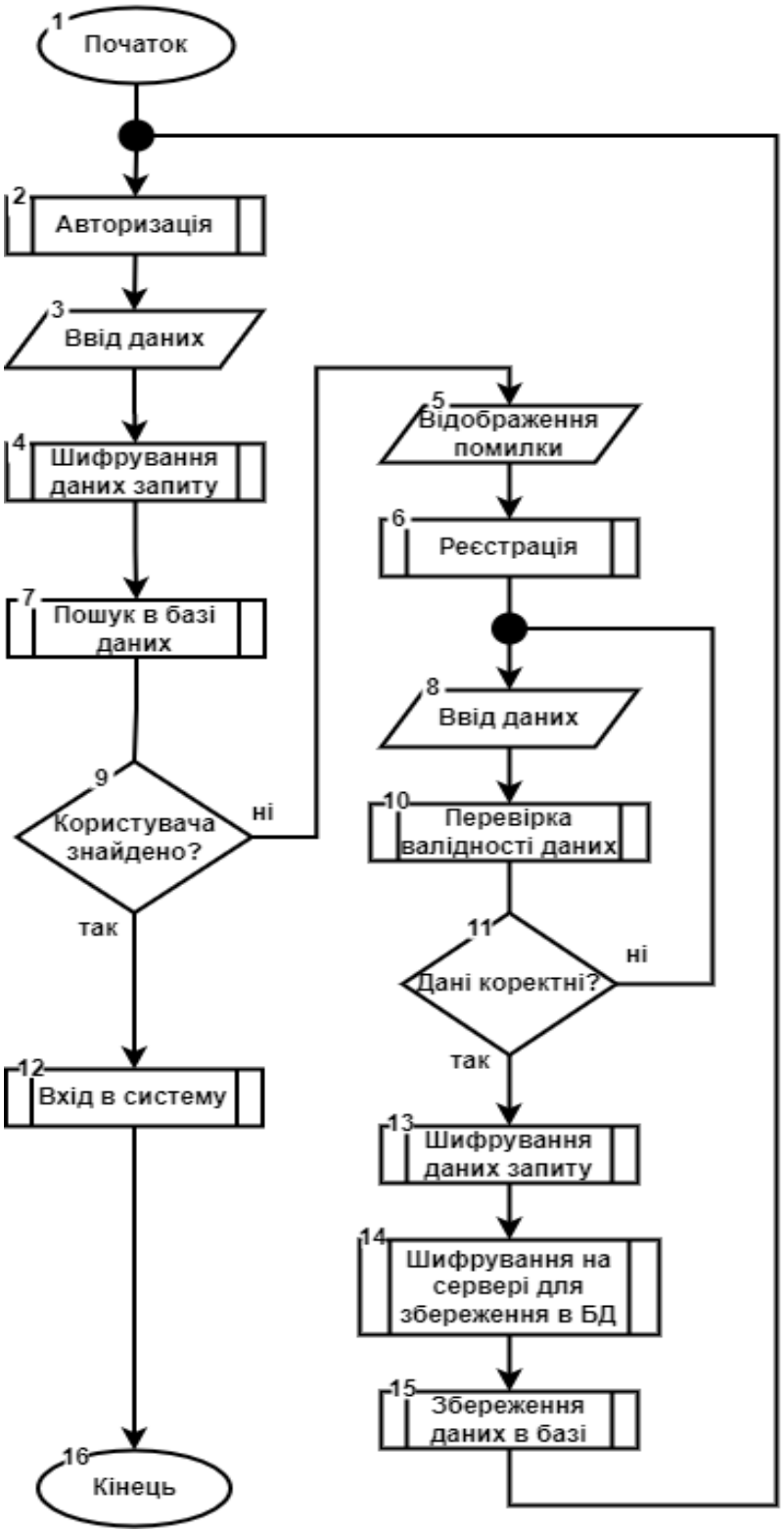

Рисунок 2.15 – Блок-схема алгоритму реєстрації та авторизації

Алгоритм розпочинається з процесу авторизації. Спочатку користувач вводить дані для входу (ім’я та пароль). Потім дані шифруються на стороні клієнта в цілях безпеки: так зловмисники не зможуть перехопити пароль із запиту.

Далі програма здійснює пошук користувачів в базі відповідно до введених даних. Якщо користувача було знайдено, він входить в систему і алгоритм завершується.

Якщо ж користувача не знайдено, відображається помилка та пропонується зареєструватися. Користувач вводить всі необхідні дані, після чого здійснюється їх перевірка на валідність. Якщо дані не коректні, користувач вводить інші. Якщо ж вони правильні, то відбувається їх шифрування на стороні клієнта та відправка на сервер. Потім дані ще раз шифруються за іншим алгоритмом та зберігаються до бази даних. Після цього хід виконання алгоритму знову переходить до процесу авторизації.

Таким чином, був розроблений загальний алгоритм роботи веб-застосунку, що забезпечує комунікацію між учасниками освітнього процесу та алгоритм авторизації і реєстрації. Алгоритми детально показують основні етапи виконання програми.

2.4 Аналіз алгоритму варіантів використання програмного модуля

Наступним важливим етапом розробки програмного додатку є аналіз та створення алгоритму роботи програмного модуля для зчитування та відображення повідомлень. Блок-схема даного алгоритму зображена на рисунку 2.16.

Даний алгоритм функціонує для відображення повідомлень користувачів. Спочатку алгоритм завантажує вже надіслані повідомлення із бази. Потім функція io.emit() бібліотеки socket.IO провокує подію відправки повідомлення, та приймає параметром саме повідомлення із всіма супутніми даними.

Майже на всіх етапах алгоритму використовується функція data.split(), яка виконує розбиття отриманих даних на необхідні елементи, такі як ім’я відправника, текс повідомлення, дата і час відправки та ідентифікатор підключення.

Далі створюється блок, в якому буде розміщуватись повідомлення, ім’я відправника та дата з часом відправки. Наступним етапом є наповнення створеного блоку отриманими даними із аргументу функції emit.

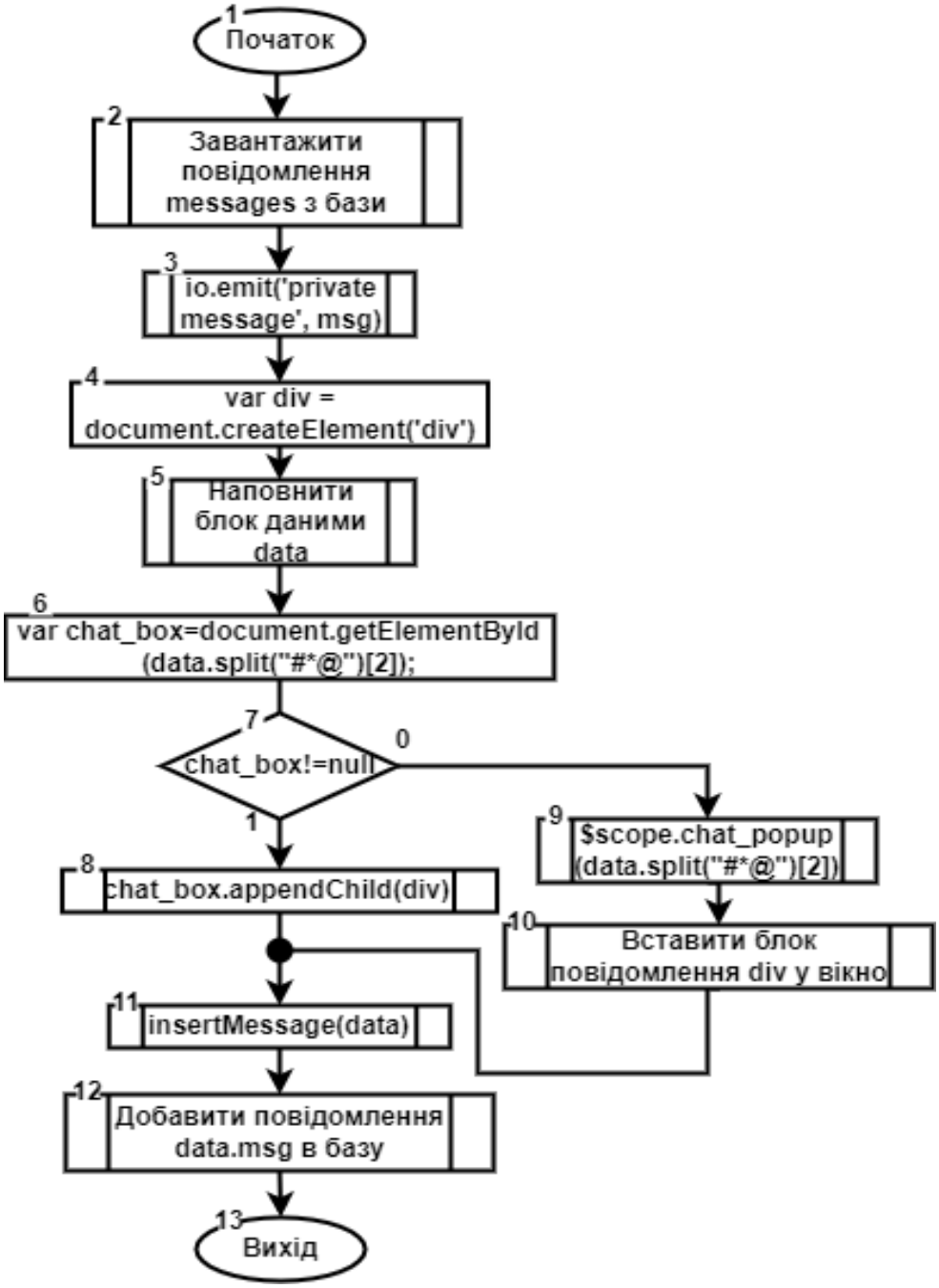

Рисунок 2.16 – Блок-схема алгоритму програмного модуля

В залежності від отриманих даних про відправника, повідомлення розміщуються необхідним чином. Якщо відправник це поточний користувач,

повідомлення та ім'я розміщуються зліва, а якщо відправник це інший користувач, то – зправа. Потім алгоритм отримує елемент вікна приватних повідомлень.

Якщо вікно не знайдене, то викликається функція, яка його створює відповідно до всіх параметрів. Далі блок з повідомленням додається до створеного вікна. Якщо ж воно вже існує, то повідомлення одразу розміщуються всередині нього.

Останнім етапом є додання повідомлення із всією необхідною інформацією до підключеної бази даних. Після цього етапу виконання даного алгоритму можна вважати завершеним.

Таким чином, був розроблений алгоритм варіантів використання програмного модуля, що забезпечує зчитування та відображення повідомлень. Алгоритм детально показує основні етапи відображення повідомлень.

2.5 Висновки

У другому розділі було проведено аналіз всіх даних у розробленому додатку та розглянуто варіанти їх циркуляції в системі. Було розглянуто структуру спроектованої бази даних та можливі варіанти масштабування структури даних веб-застосунку.

Було проаналізовано та розроблено загальну модель організації освітнього процесу, а також різні моделі комунікацій його учасників. Було обрано циклічну модель для програмної реалізації.

Було проведено розробку зручного та інтуїтивного графічного інтерфейсу користувача. Для розробки було обрано використання методології AJAX, що призвело до швидкого оновлення інтерфейсу.

Розроблено та проаналізовано загальний алгоритми роботи програмного додатку, який демонструє основні етапи виконання програми.

Також було проведено аналіз та розроблено алгоритм варіантів використання програмного модуля, який відповідає за зчитування та відображення повідомлень користувачів.

3 РОЗРОБКА ПРОГРАМИ КОМУНІКАЦІЙ УЧАСНИКІВ ОСВІТНЬОГО ПРОЦЕСУ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу

Для будь-якої програмної реалізації існує досить великий асортимент технологій та засобів. Перед початком процесу розробки дуже важливо обрати необхідні технології, адже ефективність розробки та продуктивність фінального програмного продукту напряму залежать від цього вибору. Для проведення варіантного аналізу засобів розробки, було обрано три мови програмування:

- C#;
- PHP;
- JavaScript.

C# [15] - це об’єктно-орієнтована мова програмування, розроблена компанією Microsoft. На ній пишуть різні програми, такі як невеликі десктопні додатки і навіть цілі веб-портали, які обслуговують тисячі користувачів.

C# є мовою із C-подібним синтаксисом, тому, в цьому відношенні, він схожий на C++ та Java. Ця програмна мова має сувору статичну типізацію. Серед її інших можливостей також можна виділити перевантаження операторів, підтримку поліморфізму, подій, атрибутів та наявність вказівників на функції-члени класів.

Мова C# була створена спеціально для роботи з фреймворком .NET, проте саме поняття .NET дещо ширше. Основою платформи .NET є спільне середовище виконання Common Language Runtime (CLR), завдяки чому фреймворк підтримує декілька мов. В .NET код компілюється у збірку загальною мовою CIL (Common Intermediate Language) - свого роду асемблер. Даний фреймворк також є кросплатформеним, хоча й з деякими обмеженнями. Для побудови графічних

програм з насиченим інтерфейсом в .NET існують технології WPF і WinUI, для створення більш простих графічних програм - Windows Forms.

Всі ці особливості роблять C# та .NET непоганим вибором для розроби серверної частини, проте він не зовсім підходить для роботи з клієнтською частиною веб-застосунку, також C# легко дизасемблюється, що робить код не зовсім захищеним.

PHP [16] – це скриптова мова загального призначення, яка є інтерпретованою, що дозволяє створювати програми в процедурному і об'єктно-орієнтованому стилі. PHP є найбільш поширеною мовою програмування у сфері веб-розробки. На сьогодні PHP лідирує серед серверних мов програмування, які застосовуються для створення динамічних веб-сайтів та веб-додатків.

Більшість backend частин написані саме на PHP, мова підтримується багатьма хостинг-провайдерами. Дана технологія набула високої актуальності завдяки своїй швидкості, мультипарадигмі, простоті та кросплатформеності. Ще одним плюсом PHP є відкриті вихідні коди. Мова має простий синтаксис, частково схожий на Java та C++. PHP перетворюється в HTML-код з допомогою серверу і передається на сторону клієнта. Оскільки браузер отримує готовий html-код, користувач не бачить PHP-коду. Це значно відрізняє PHP від JavaScript. З точки зору безпеки, це є перевагою проте значно погіршує інтерактивність веб-сторінок.

Таким чином, PHP є доцільним вибором для розробки веб-серверу, проте зовсім не підходить для програмування клієнтської частини. Також він не має стандартизованих фреймворків і бібліотек. Більшість фреймворків PHP є дуже розрізненими і не актуальними.

JavaScript [17] – це об'єктно-орієнтована мова програмування, розроблена компанією Netscape з метою створення інтерактивних HTML-документів. JavaScript дозволяє створювати додатки, що виконуються як на стороні клієнта, так і на стороні сервера. Синтаксис мови дуже подібний до синтаксису мови Java – тому його часто називають Java-подібним. Клієнтські програми JavaScript

виконуються браузером на пристрої користувача, а серверні програми – на сервері. JavaScript сам по собі досить компактний та гнучкий. Для нього існує велика кількість інструментів, таких як вбудовані в браузери програмні інтерфейси (API), сторонні API та фреймворки. Дані інструменти забезпечують різні функціональні можливості, наприклад, динамічне створення HTML, захоплення відеопотоку або ж роботу з веб-камерою.

JavaScript має велику кількість широко відомих фреймворків, таких як Node [18], React, Angular [19] та Vue. Вони містять в собі багато функцій і внутрішніх фреймворків, що забезпечують ефективну розробку веб-додатків. Node.js дозволяє розробляти серверні частини веб-застосунків, в той час, як інші фреймворки використовуються для програмування клієнтської частини. Важливим плюсом Node.js є підтримка роботи з нереляційними базами даних, наприклад, MongoDB [20]. JavaScript також дозволяє використовувати бібліотеку socket.IO, що є дуже необхідним при розробці моделей комунікацій.

JavaScript також передбачає застосування різних методологій розробки програмного забезпечення, таких як AJAX. Дана методологія забезпечує ефективну та швидку розробку графічного інтерфейсу веб-додатків.

Таким чином, розглянувши основні можливості різних мов програмування, було обрано використання мови JavaScript, адже вона має багато необхідних інструментів та технологій для ефективної розробки повноцінних веб-застосунків із серверною та клієнтською частинами.

3.2 Розробка модуля обміну повідомленнями

Модуль обміну повідомленнями є головним программним модулем веб-застосунку. Він реалізовує модель комунікації між учасниками освітнього процесу. В ході розробки модуля використовувалась технологія серверного програмування Node.js та фреймворк Angular для розробки клієнтської частини.

Для обміну інформацією між користувачами було використано бібліотеку socket.IO. Принцип її роботи базується на використанні сокетів (рис. 3.1). На сервері створюється сокет, який встановлює двосторонній зв’язок із сокетами клієнтських частин. Таким чином відбувається постійна взаємодія всіх клієнтів через сервер, який обробляє їхні запити.

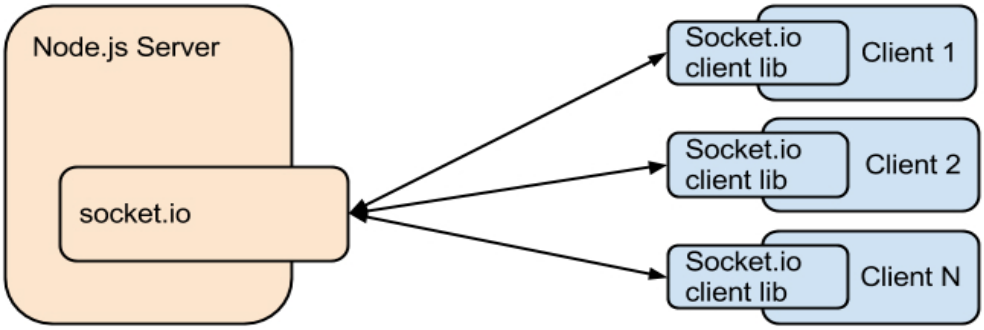

Рисунок 3.1 – Принцип роботи socket.io

Сокети можуть як викликати певну подію та передавати дані на сервер, так і обробляти необхідну подію відповідно до заданого алгоритму. Для кращого розуміння даного процесу на рисунку 3.2 представлена діаграма послідовностей socket.io.

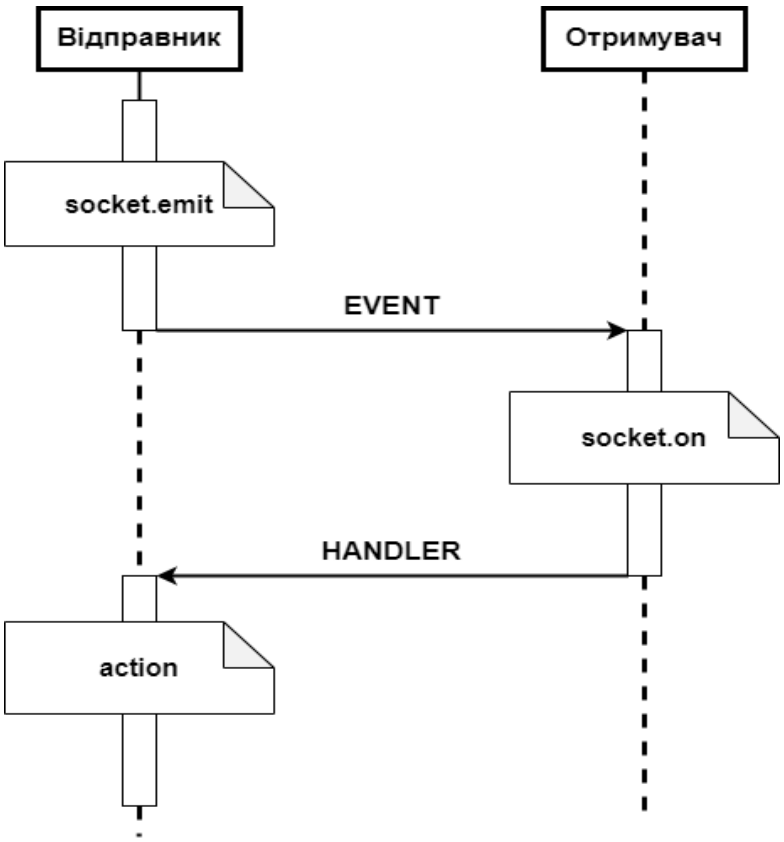

Рисунок 3.2 – Діаграма послідовностей socket.io

Сокет на стороні відправника провокує подію з допомогою функції socket.emit, а сокет на стороні отримувача обробляє її з допомогою функції socket.on і повертає запит. Даний механізм широко використовувався при розробці модуля обміну повідомленнями.

Перш ніж перейти до розробки самого модуля, необхідно розглянути та проаналізувати лістинг файлу model.js, який містить моделі даних, що використовуються в роботі модуля.

Спочатку підключаються необхідні бібліотеки та модулі. Потім встановлюється зв’язок з кластером бази даних за посиланням і вказуються необхідні параметри. Далі визначається код, що відповідає за помилку або успішне підключення до бази. Лістинг файлу наведено нижче.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;


mongoose.connect(

  'mongodb+srv://Yaroslav:admin@chat.7q9dy.mongodb.net/chat?retryWrites=true&w
=majority',

  {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  }
);


mongoose.connection.on('open', function (ref) {
  console.log('Connected to mongo server.');
});

mongoose.connection.on('error', function (err) {
  console.log('Could not connect to mongo server!');
  console.log(err);
});
```

Наступним кроком визначається схема даних моделі користувача. Детальний аналіз даних наведений в розділі 2.1.

```
module.exports.user = mongoose.model(
  'User',
  new Schema(
    {
      group: String,
      name: String,
      password: String,
      phone: String,
      email: String,
      friends: [],
    },

    { strict: false }
  )
);
```

Тим же способом визначаються схеми даних й інших моделей, таких як повідомлення та підключення.

```
module.exports.online = mongoose.model(
  'online',
  new Schema({
    name: String,
    connection_id: String,
  })
);


module.exports.messages = mongoose.model(
  'message',
  new Schema({
    message: String,
    sender: String,
    reciever: String,
    date: Date,
  })
);
```

Провівши аналіз лістингу моделей даних, можна перейти до розробки модуля обміну повідомленнями. Виконання коду розпочинається з виклику методу send_message, який створює блок повідомлення у відправника та викликає функцію socket.emit для провокування події приватного повідомлення. Лістинг методу наведено нижче.

```
$scope.send_message = function(chat,message){

        console.log(chat);

        div = document.createElement('div');
        div.innerHTML='<div class="direct-chat-msg"> \
                        <div class="direct-chat-info clearfix">\
                        <span class="direct-chat-name pull-
left">'+$scope.user+'</span>\

                        <span class="direct-chat-timestamp pull-
right">'+getDate()+'</span>\

                        </div>\
                        <img class="direct-chat-img" src=" " alt=" ">\

<div class="direct-chat-text">'
                        +message+
                        '</div>\
                        </div>';

        document.getElementById(chat).appendChild(div);

        document.getElementById(chat).scrollTop=document.getElementById(chat).
scrollHeight;

        socket.emit('private message',chat+"#*@"+message+"#*@"+$scope.user
+"#*@" + getDate());

        insertMessage($scope.user,chat,message);

        $scope.message=null;
    }
```

В даному методі використовується функція getDate для коректного відображення часу і дати відправки повідомлення. Лістинг функції наведено нижче.

```
var monthNames = ["Січень", "Лютий", "Березень", "Квітень", "Травень",
"Червень", "Липень", "Серпень", "Вересень", "Жовтень", "Листопад", "Грудень"];

 var getDate = function(){

        date = new Date();

        hour = date.getHours();
```

```
        form_date = monthNames[date.getMonth()]+" "+date.getDate()+",
"+hour+":"+date.getMinutes();

        return form_date;
    }
```

Потім функція socket.on обробляє подію приватного повідомлення на стороні сервера. В ході її виконання до бази даних додаються всі необхідні атрибути. Потім викликається інша функція io.to().emit() для отримувача повідомлення. Лістинг функції socket.on на стороні сервера наведено нижче.

```
socket.on('private message',function(msg){

            console.log('message  :'+msg.split("#*@")[0]);

            models.messages.create(

           {
                "message":msg.split("#*@")[1],
                "sender" :msg.split("#*@")[2],
                "reciever":msg.split("#*@")[0],
                "date" : new Date()
           }
           );

            io.to(users[msg.split("#*@")[0]]).emit('private message', msg);
      }
     );
```

На стороні отримувача повідомлення виконується обробка функцією socket.on. В ході її виконання створюється блок повідомлення в отримувача. Також блок повідомлення додається у вікно приватних повідомлень. Якщо таке вікно не знайдено, то програма його створює Останнім етапом викликається функція insertMessage. Лістинг функції socket.on на стороні клієнта наведено нижче.

```
socket.on('private message', function(data) {

        var div = document.createElement('div');

        div.innerHTML='<div class="direct-chat-msg right">\
                        <div class="direct-chat-info clearfix">\
                        <span class="direct-chat-name pull-right"

style="position: absolute !important; right: 50px
!important;">'+data.split("#*@")[2]+'</span>\
                        <span class="direct-chat-timestamp pull-
left">'+getDate()+'</span>\

                        </div>\
                        <img class="direct-chat-img" src=" " alt=" ">\
                        <div class="direct-chat-text">'
                        +data.split("#*@")[1]+
                        '</div>\

                        </div>';
        var chat_box = document.getElementById(data.split("#*@")[2]);

        console.log(chat_box);

        if(chat_box!=null){
            chat_box.appendChild(div);
        }
        else{
            $scope.chat_popup(data.split("#*@")[2]);

            document.getElementById(data.split("#*@")[2]).appendChild(div);
        }

        insertMessage(data.split("#*@")[2],data.split("#*@")[2],data.split("#*
@")[1]);

        document.getElementById(data.split("#*@")[2]).scrollTop=document.getEl
ementById(data.split("#*@")[2]).scrollHeight;
    }
);
```

Функція insertMessage виконує обробку даних повідомлень в локальному сховищі та додає нове повідомлення. Лістинг функції наведено нижче.

```
var insertMessage = function(from,to,msg){

        console.log(from + " " + to);

        if (to in $scope.messages){
            if ($scope.messages[to].length>25){
                $scope.messages[to].splice(0,1);
            }
        }
   else{
        $scope.messages[to]=[];
    }
    $scope.messages[to].push(
        {
         "sender":from,
         "msg" : msg,
         "date" : getDate()
        }
 );
    localStorage.setItem(to,JSON.stringify($scope.messages[to]));
    localStorage.setItem(from,JSON.stringify($scope.messages[from]));
    console.log(localStorage.getItem(to));
}
```

Таким чином, було розроблено програмний модуль для обміну повідомленнями між користувачами. Модуль забезпечує швидку та ефективну комунікацію учасників освітнього процесу.

3.3 Висновки

В третьому розділі було проведено аналіз мов програмування та обґрунтовано вибір засобів розробки і технологій, що будуть використані при розробці програмного додатку.

В результаті аналізу було обрано мову програмування JavaScript та технології Node.js, Angular та socket.IO для розробки серверної і клієнтської частин веб-застосунку.

Було проведено розробку та загальний аналіз коду програмного модуля для обміну повідомленнями між користувачами в режимі реального часу. Даний

модуль забезпечує швидку та ефективну модель комунікації учасників освітнього процесу.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ ОБМІНУ ПОВІДОМЛЕННЯМИ

4.1 Тестування програми

Тестування програмних систем [21] є критично важливим етапом їх розробки. Важливість тестування обумовлюється перевіркою відповідності між кінцевим програмним продуктом та його очікуваною версією. Воно дозволяє виявити наявні помилки та оцінити загальний рівень якості програмної системи. Існує багато методів та технологій для тестування, головною метою яких є верифікація та валідація програмного забезпечення.

Оскільки розроблений веб-застосунок повинен забезпечувати комунікацію учасників освітнього процесу, потрібно провести тестування модуля обміну повідомленнями, модулів авторизації і реєстрації та інших основних частин системи. Процес тестування розробленої веб-системи розпочинається з модуля авторизації та реєстрації. Спочатку оцінюється зручність і якість інтерфейсу та швидкість завантаження вікон. Вікно авторизації представлено на рисунку 4.1.

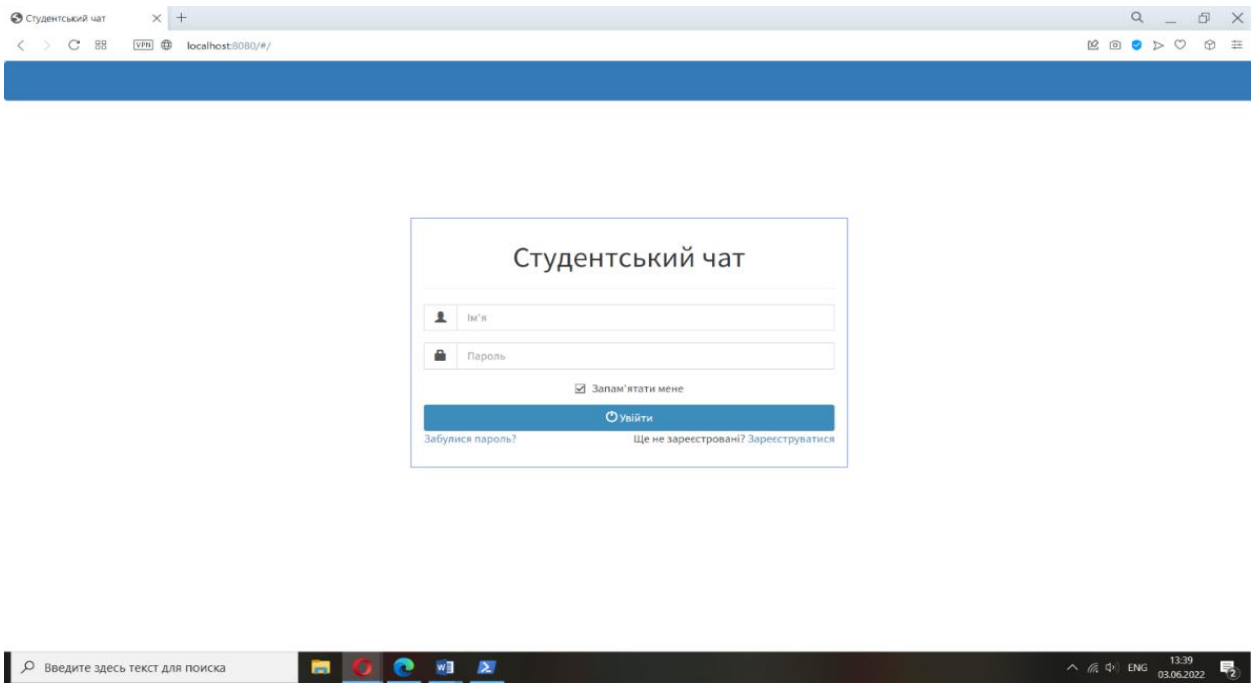

Рисунок 4.1 – Вікно авторизації

В ході перевірки були отримані задовільні результати швидкості завантаження та зручності інтерфейсу вікна. Тепер необхідно протестувати ввід даних. При вводі неправильних даних, з’являється повідомлення про некоректний ввід (рис. 4.2).

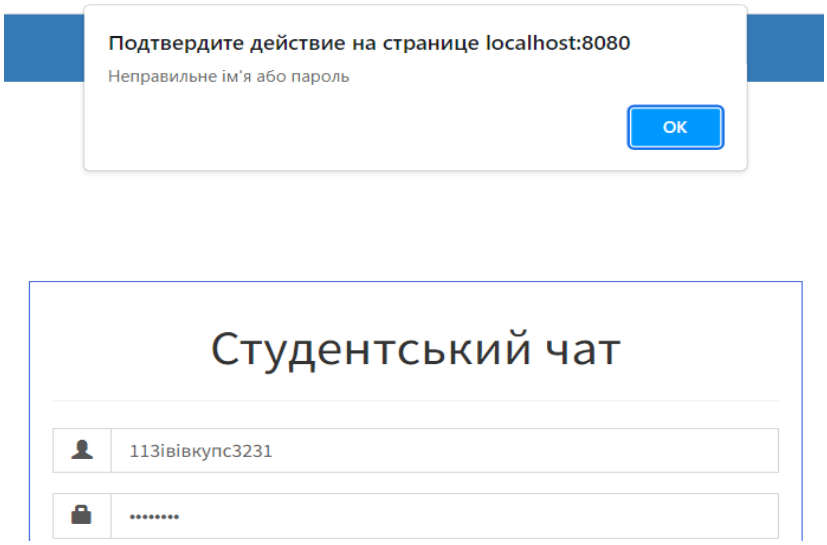

Рисунок 4.2 – Повідомлення про неправильні дані

Далі необхідно провести тестування форми реєстрації нового користувача (рис. 4.3). Для цього потрібно натиснути на напис «Зареєструватися».

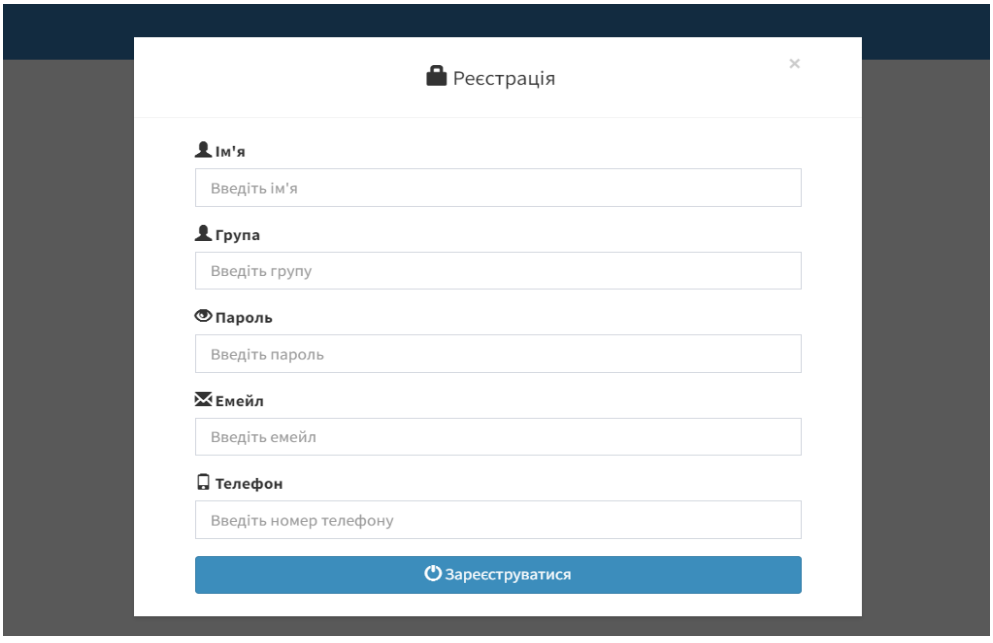

Рисунок 4.3 – Інтерфейс форми реєстрації

Форма швидко завантажується та правильно відображається. Протестуємо ввід даних. Для цього заповнимо усі поля окрім, наприклад, емейлу. В результаті, з’явиться повідомлення про порожнє поле (рис. 4.4).

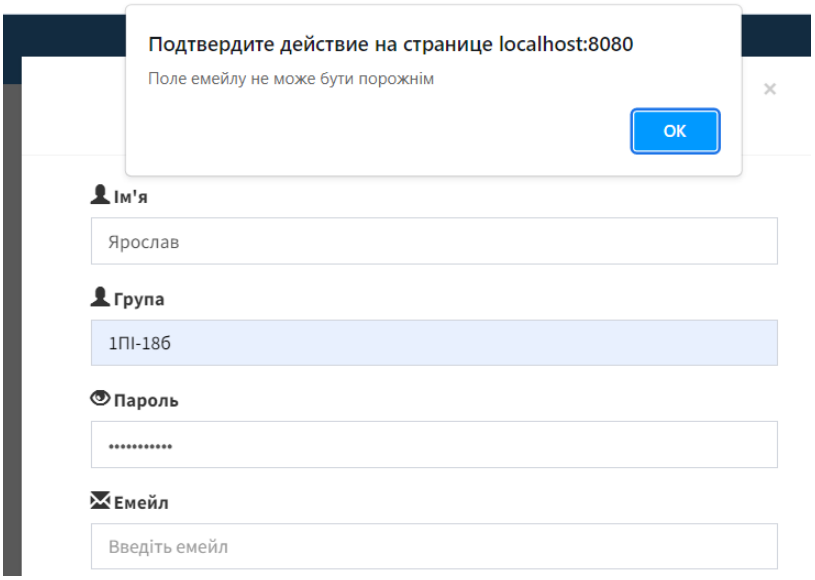

Рисунок 4.4 – Повідомлення про порожнє поле

Тепер протестуємо ввід некоректних даних. Для цього введемо в поле емейлу неправильне значення. В результаті, з’явиться повідомлення про некоректний формат емейлу (рис. 4.5).

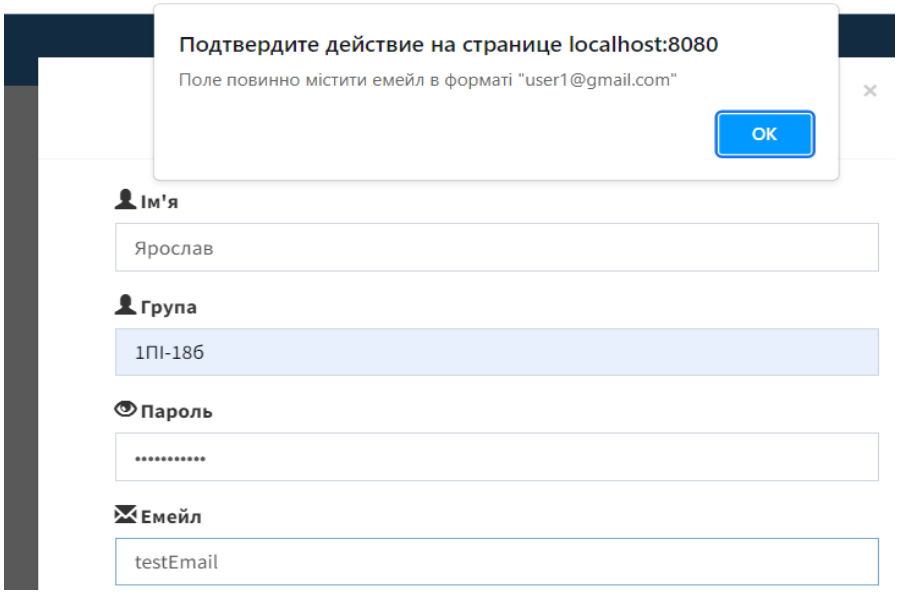

Рисунок 4.5 – Повідомлення про некоректний формат емейлу

Схожі повідомлення з’являються і при вводі неправильних даних в інші поля. Наприклад, якщо ввести в поле номера телефону літери, з’явиться повідомлення, що представлене на рисунку 4.6.

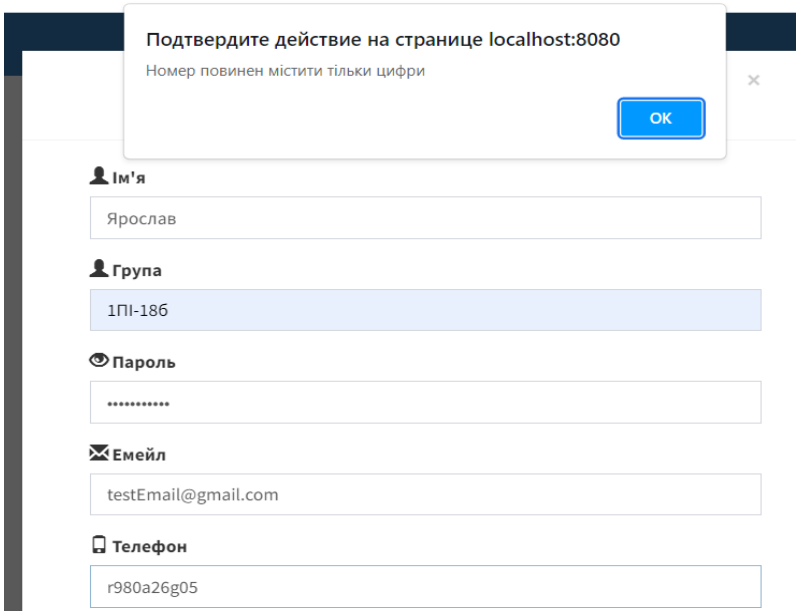

Рисунок 4.6 – Повідомлення при некоректному вводі номера

При вводі всіх правильних даних створюється новий користувач. Можна перейти до бази даних (рис. 4.7) і переконатися в цьому.

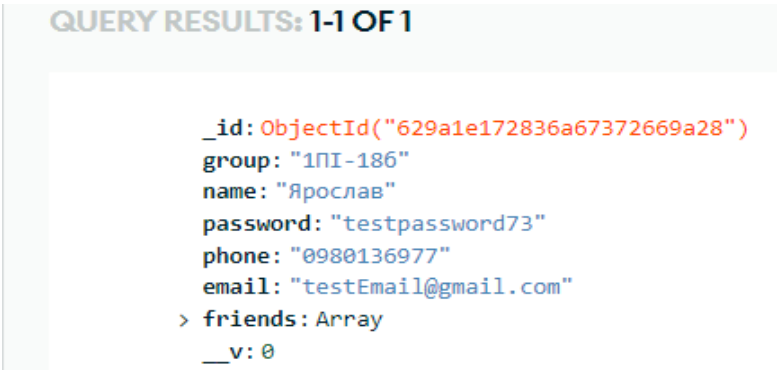

Рисунок 4.7 – Створений користувач в базі даних

Тепер спробуємо зареєструвати ще одного користувача, використовуючи ті ж дані. Наприклад, введемо вже зареєстрований емейл. В результаті, з’явиться повідомлення про існування користувача з такими даними (рис. 4.8).

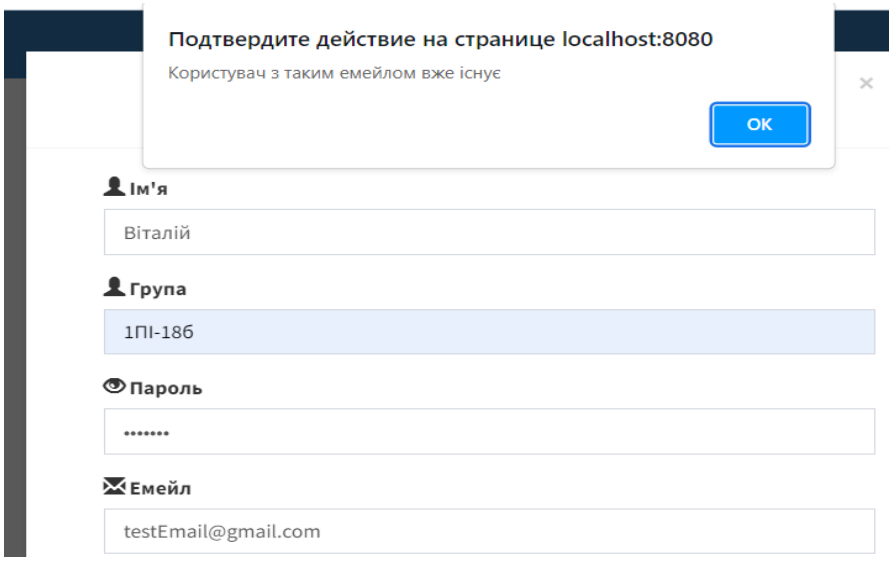

Рисунок 4.8 – Повідомлення про існування користувача

Після авторизації відкривається головне вікно веб-застосунку (рис. 4.9). Всі елементи вікна швидко та коректно відображаються, але поки що вони порожні.

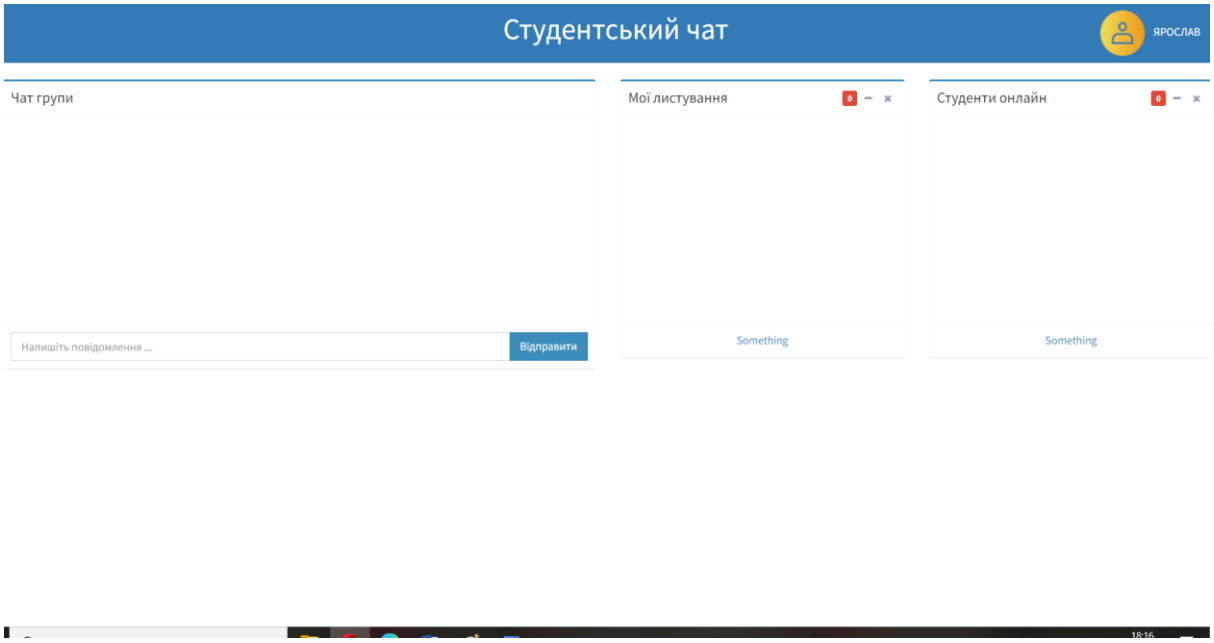

Рисунок 4.9 – Головне вікно веб-застосунку

Для тестування підключень потрібно відкрити ще декілька вкладок браузера з веб-чатом та авторизувати користувачів. В результаті, всі активні підключення коректно відображаються в розділі «Студенти онлайн» (рис. 4.10).

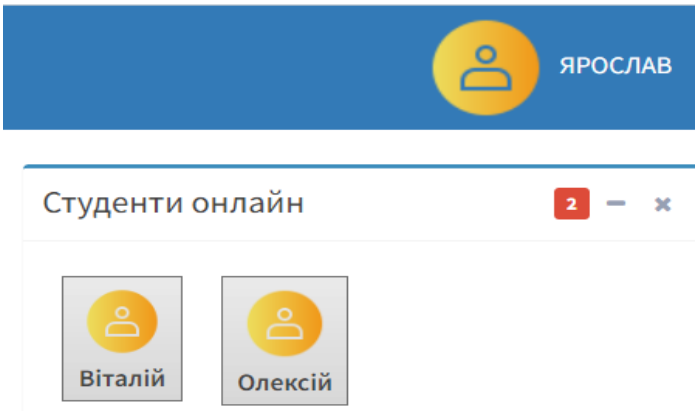

Рисунок 4.10 – Активні підключення

Далі протестуємо обмін повідомленнями в груповому чаті. Для цього потрібно написати будь-яке повідомлення в поле вводу та натиснути «Відправити». Потім потрібно перейти на інші вкладки з веб-чатом та відправити з них відповіді. В результаті, ім’я, час і дата відправки та самі повідомлення всіх користувачів коректно відображаються в груповому чаті (рис. 4.11).

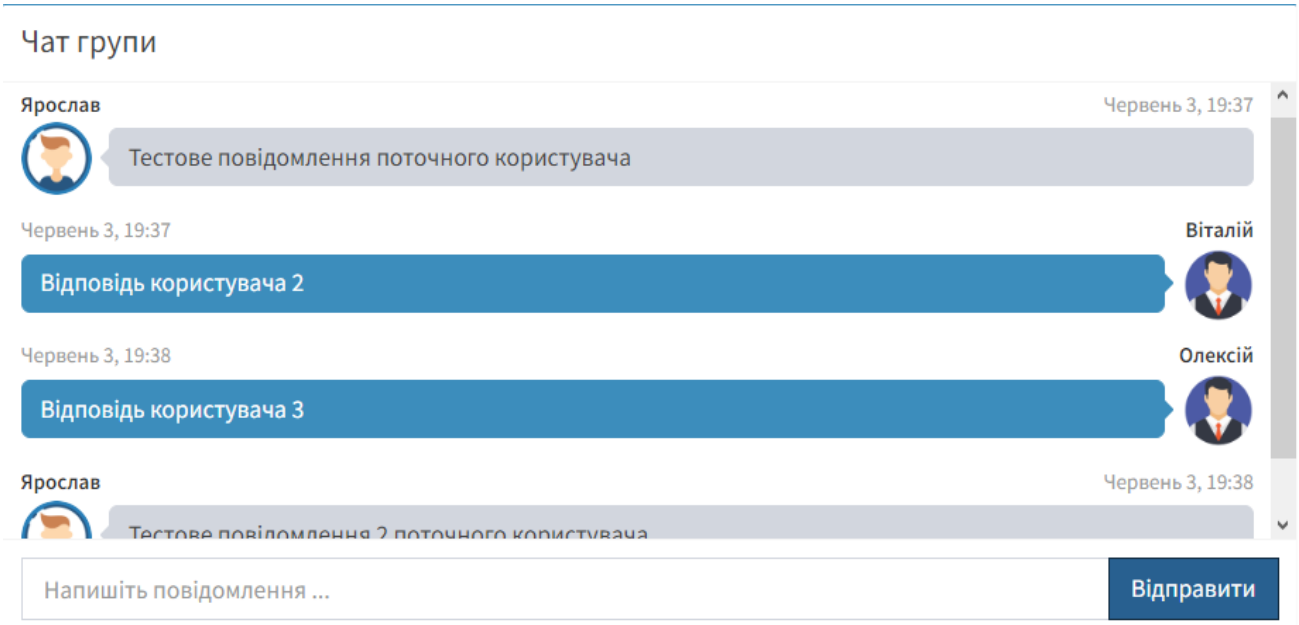

Рисунок 4.11 – Повідомлення в груповому чаті

Наступним кроком необхідно провести тестування модуля приватних повідомлень. Для цього спочатку потрібно відправити користувачу запит на листування (рис. 4.12).

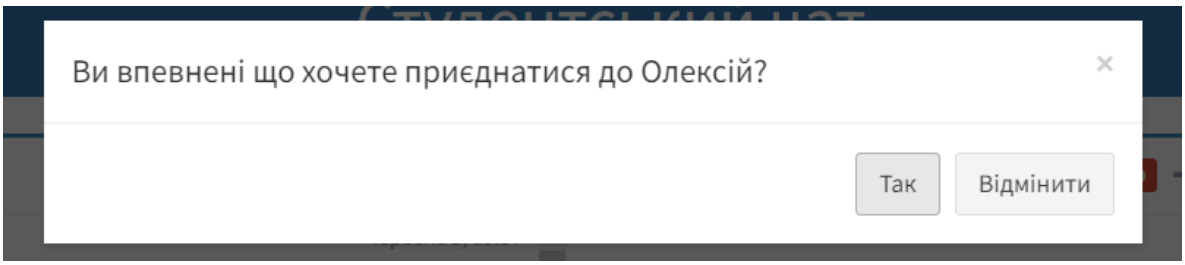

Рисунок 4.12 – Відправка запиту на листування

Далі потрібно перевірити надходження запиту. Відкривши вкладку веб-чату в обраного користувача, можна спостерігати наявність запиту на листування (рис. 4.13).

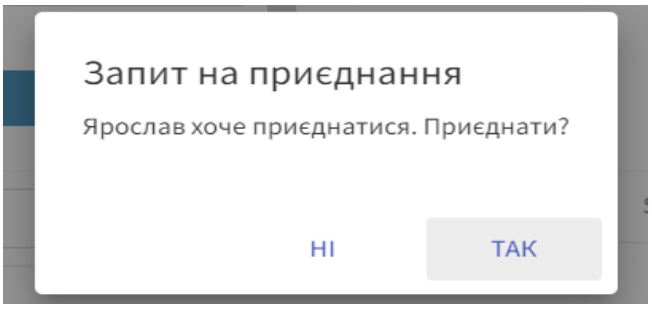

Рисунок 4.13 – Отримання запиту на листування

Перейшовши до розділу активних листувань, можна спостерігати коректне відображення обраного користувача (рис. 4.14).

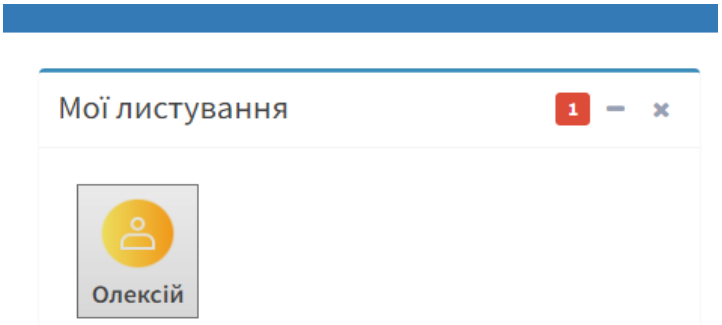

Рисунок 4.14 – Відображення активних листувань

Наступним кроком потрібно протестувати саму відправку приватних повідомлень. Для цього напишемо тестове повідомлення в поле вводу та

натиснемо «Відправити». Потім необхідно здійснити ті ж операції на стороні обраного користувача. В результаті, всі відправлені та отримані повідомлення коректно відображаються (рис. 4.15).

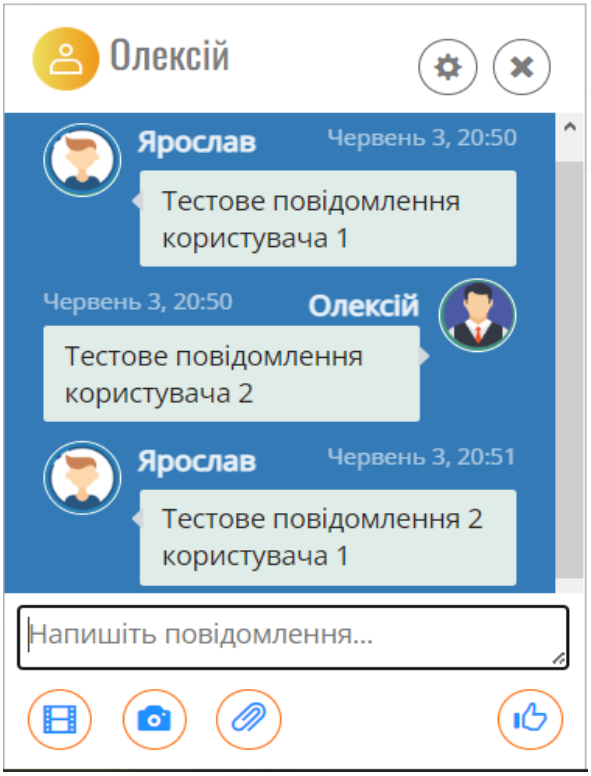

Рисунок 4.15 – Повідомлення в приватному чаті

Останнім етапом є перевірка збереження повідомлень. Перейшовши до бази даних, можна спостерігати всі надіслані та отримані повідомлення в сховищі (рис. 4.16).

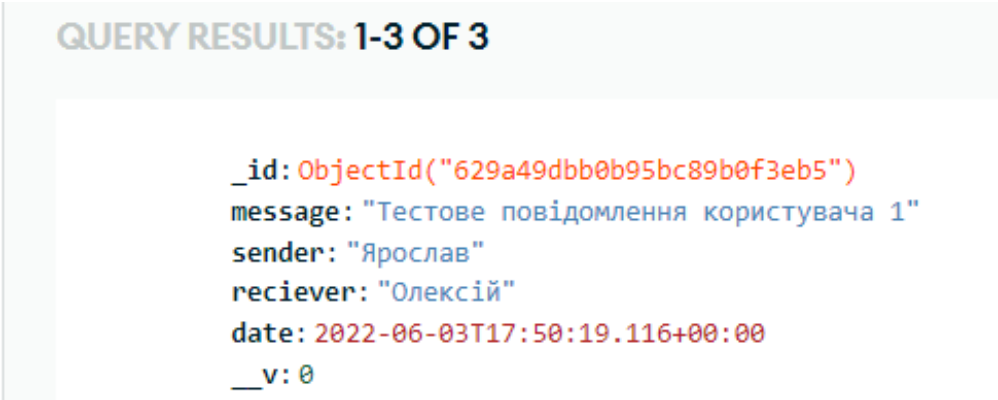

Рисунок 4.16 – Збережені повідомлення в базі даних

Таким чином, в ході процесу тестування було доведено що інтерфейс та функціонал розробленого веб-застосунку відповідає всім очікуванням та поставленому завданню.

4.2 Розробка інструкції користувача

Для проектування інструкції користувача спочатку необхідно чітко визначити технічні вимоги для розробленого програмного забезпечення. В таблиці 4.1 міститься загальна інформація стосовно мінімальної конфігурації персонального комп'ютера необхідного для виконання програми.

Таблиця 4.1 – Мінімальна конфігурація ПК

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 1 ГГц
Об'єм оперативної пам'яті	500 МБ для 32-розрядної системи і 1 ГБ для 64-разрядної системи
Вільне місце на жорсткому диску	500 МБ
Операційна система	Windows 7

В таблиці 4.2 міститься рекомендована конфігурація ПК.

Таблиця 4.2 – Рекомендована конфігурація ПК

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 2 ГГц
Об'єм оперативної пам'яті	2 ГБ для 32-розрядної системи і 4 ГБ для 64-разрядної системи
Вільне місце на жорсткому диску	1 ГБ

Операційна система	Windows 10

Розроблений веб-застосунок використовує мережеву платформу Node.js, яка є середовищем виконання JavaScript коду, тому для успішного запуску додатку спочатку необхідно встановити дану платформу. Для цього потрібно перейти на офіційну сторінку Node.js, завантажити актуальну версію (рис. 4.17), запустити інсталятор та прослідувати всім інструкціям.

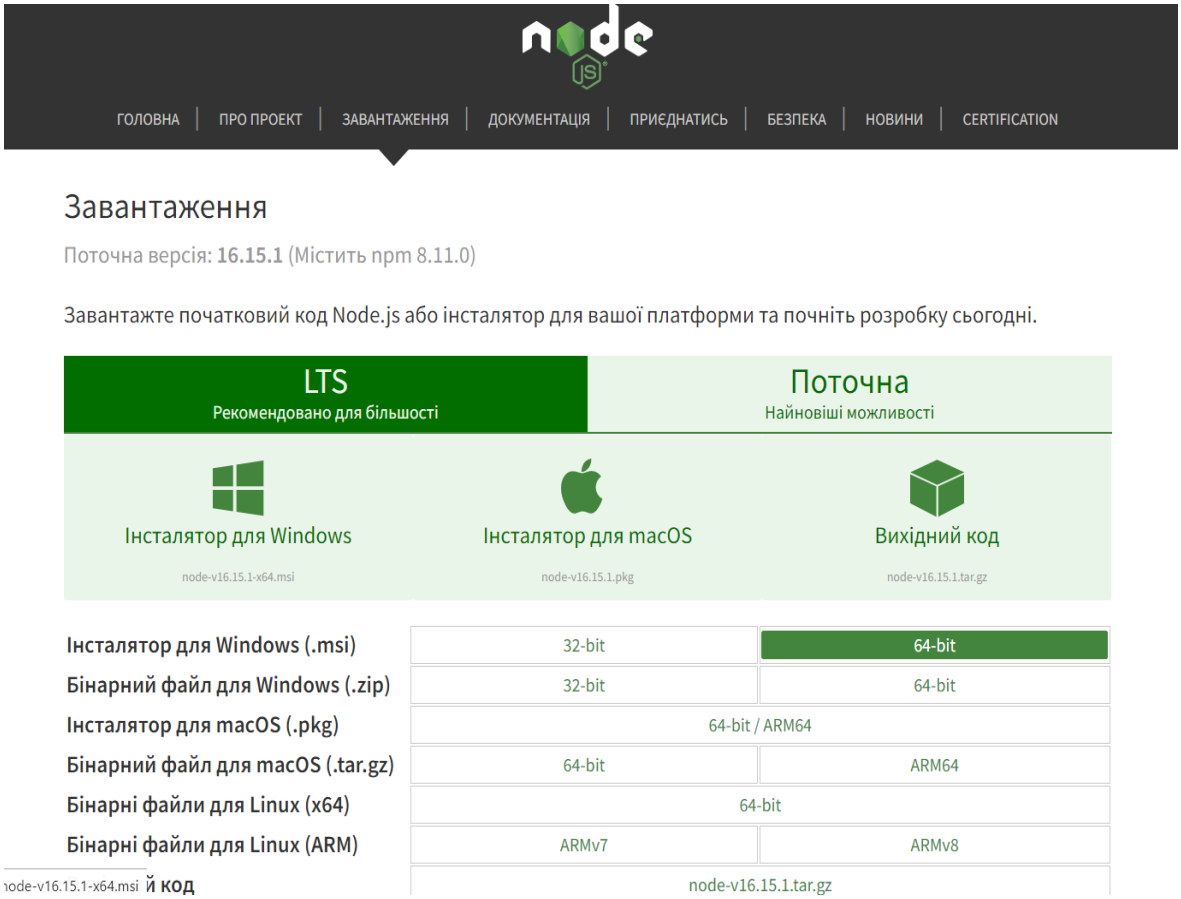

Рисунок 4.17 – Завантаження Node.js

Разом із середовищем виконання Node.js автоматично встановлюється й менеджер пакетів NPM (Node Package Manager). Для подальшого виконання програми необхідно запустити командний рядок Windows в директорії програмного продукту та ввести команду «npm start» (рис. 4.18). Таким чином запуститься збірка всіх необхідних для виконання програмного засобу пакетів та модулів Node.js.

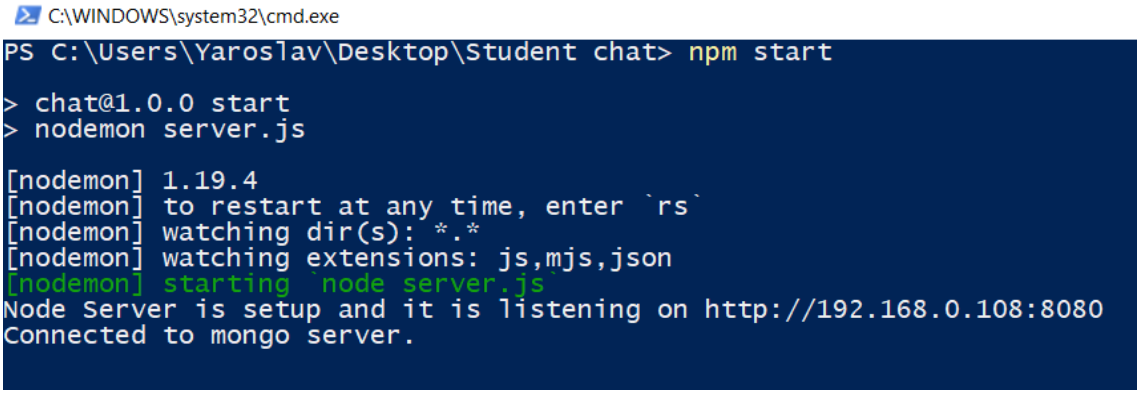

Рисунок 4.18 – Запуск npm

Останнім кроком є підключення до сервера. Для цього потрібно відкрити будь-який браузер, наприклад, Opera та підключитися до локального порту 8080, ввівши в адресний рядок «localhost:8080» (рис. 4.19).

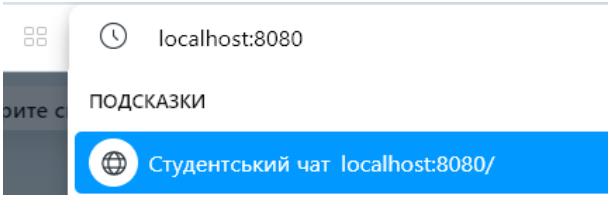

Рисунок 4.19 – Підключення до сервера

Після виконання всіх вказаних налаштувань користувач може повністю використовувати функціонал розробленого веб-застосунку. Для цього спочатку потрібно зареєструватись та авторизуватись в системі.

4.3 Висновки

У четвертому розділі було здійснено тестування розробленого веб-застосунку. Тестування показало що додаток відповідає всім очікуванням та технічному завданню. Вікна додатку швидко завантажуються, а інтерфейс коректно відображається. Весь функціонал працює як і очікувалось.

Було створено інструкцію користувача по встановленню та використанню програмного застосунку.

ВИСНОВКИ

В ході виконання бакалаврської дипломної роботи було розроблено програмну реалізацію моделей комунікацій учасників освітнього процесу. Для розробки було використано середовище програмування Visual Studio Code.

Було проведено аналіз стану моделей комунікацій та їх програмних реалізацій. Проаналізовано переваги та недоліки наявних аналогів і проведено їх порівняння з власним веб-додатком. Було визначено та сформовано головні завдання бакалаврської дипломної роботи.

В бакалаврській дипломній роботі було виконано наступні задачі:

- проведено аналіз моделей комунікацій та визначено найбільш ефективний підхід до їхньої програмної реалізації;
- розроблено та проведено аналіз варіативних моделей комунікацій освітнього процесу;
- розроблено загальну модель системи комунікацій між учасниками освітнього процесу;
- виконано аналіз моделей та визначено модуль для програмної реалізації;
- розроблено алгоритм для обміну повідомлень між користувачами в режимі реального часу;
- розроблено зручний та інтуїтивний графічний інтерфейс користувача для взаємодії із логікою додатку;
- розроблено веб додаток для комунікації учасників освітнього процесу;
- проведено тестування розробленого додатку.

Також було проведено детальний аналіз даних в розробленому веб-застосунку. Було розроблено загальну модель організації освітнього процесу, в рамках якої обрано та проаналізовано циклічну модель комунікації.

Було створено та описано блок-схеми алгоритмів роботи програмного застосунку. Розглянуто загальну схему роботи програми та схему реєстрації і

авторизації користувачів. Також було проаналізовано алгоритм варіантів використання програмного модуля для відображення повідомлень.

В ході проведеного аналізу технологій розробки програмного забезпечення було обрано мову програмування JavaScript та фреймворки Node.js, Angular, бібліотеку Socket.IO та нереляційну базу даних MongoDB. Розроблено та описано модуль для обміну повідомленнями між користувачами в режимі реального часу.

Було здійснено тестування розробленого веб-застосунку, яке підтвердило його ефективність та працездатність і встановило відповідність кінцевого програмного рішення всім очікуванням та поставленому технічному завданню. Також було розроблено та описано інструкцію користувача.

Спроектована модель організації освітнього процесу, модель обміну повідомленнями та метод реалізації програми передбачають широкий простір для масштабування і розвитку. Запропонований програмний модуль може бути реалізованим в декілька етапів розробки. На даному етапі створено модель комунікації учасників освітнього процесу та розроблено її програмну реалізацію у вигляді веб-застосунку студентського чату. Інші складові загальної моделі можуть бути реалізовані в наступних наукових роботах, наприклад, в магістерській дипломній роботі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Corporate Communication? *Simpplr*. : веб-сайт. URL: https://www.simpplr.com/blog/2020/what-is-corporate-communications/ (дата звернення: 28.05.2022).

2. Козлюк Я. В. Моделі комунікацій учасників освітнього процесу та їх програмна реалізація. Матеріали науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ). Вінниця, 2022. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15540.

3. Cobley P., Schulz P. J. Theories and Models of Communication : довідник. Берлін : Walter de Gruyter, 2013. 452 с.

4. Narula U. Communication Models : довідник. Нью-Делі : Atlantic Publishers & Dist, 2006. 121 с.

5. What is Instant Messaging? *Brosix* : веб-сайт. URL: https://www.brosix.com/blog/what-is-instant-messaging/ (дата звернення: 28.05.2022).

6. SDLC Models. *JavaTpoint* : веб-сайт. URL: https://www.javatpoint.com/software-engineering-sdlc-models (дата звернення: 28.05.2022).

7. Blob. *JavaScript.info* : веб-сайт. URL: https://javascript.info/blob (дата звернення: 30.05.2022).

8. Maarten C. A., De Vries M. J. Science and Technology Education and Communication. Нью-Йорк : Springer, 2016. 200 с.

9. What Is The Lasswell Communication Model? *FourWeekMBA* : веб-сайт. URL: https://fourweekmba.com/lasswell-communication-model/ (дата звернення: 30.05.2022).

10. Shannon Weaver Model Of Communication – 7 Key Concepts. *HelpfulProfessor* : веб-сайт. URL: https://helpfulprofessor.com/shannon-weaver-model/ (дата звернення: 30.05.2022).

11. Schramm’s Model of communication. *QS Study* : веб-сайт. URL: https://qsstudy.com/schramms- model-communication/ (дата звернення: 30.05.2022).

12. Exploring the UI Universe. *Altia* : веб-сайт. URL: https://www.altia.com/2014/09/22/different-types-of-ui/ (дата звернення: 2.06.2022).

13. An Introduction to AJAX for Front-End Designers. *Evantotutsplus* : веб-сайт. URL: https://webdesign.tutsplus.com/tutorials/an-introduction-to-ajax-for-front-end-designers--cms-25099 (дата звернення: 2.06.2022).

14. Вступ до алгоритмів / Кормен Т., Лейзерсон Ч., Рівест Р., Стайн К. Київ: К.І.С., 2019. 1288 с.

15. Overview of C# language. *WIDE SKILLS* : веб-сайт. URL: https://www.wideskills.com/csharp/overview-csharp (дата звернення: 2.06.2022).

16. Lockhart J. Modern PHP : навч. посіб. Ньютон : O'Reilly Media, Inc., 2015. 270 с.

17. Haverbeke M. Eloquent JavaScript : навч. посіб. Вид. 3-тє. Сан-Франциско : No Starch Press, 2018. 472 с.

18. Introduction to Node.js. *Nodejs.dev* : веб-сайт. URL: https://nodejs.dev/learn (дата звернення: 2.06.2022).

19. What is Angular? *Angular* : веб-сайт. URL: https://angular.io/guide/what-is-angular (дата звернення: 2.06.2022).

20. What is MongoDB? *Guru99* : веб-сайт. URL: https://www.guru99.com/what-is-mongodb.html (дата звернення: 2.06.2022).

21. What is Software Testing? *Guru99* : веб-сайт. URL: https://www.guru99.com/software-testing-introduction-importance.html (дата звернення: 4.06.2022).

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

" 25 " березня 2022 р.

Технічне завдання

на бакалаврську дипломну роботу «Моделі комунікацій учасників освітнього процесу та їх програмна реалізація»

за спеціальністю

121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

_____ к.т.н., доц. О.О. Коваленко

"____"_____2022 р.

Виконав:

_____ ст. гр. 1ПІ-18б Я.В. Козлюк

"____"_____2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Моделі комунікацій учасників освітнього процесу та їх програмна реалізація».

Галузь застосування – системи управління навчанням, платформи дистанційного навчання

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 13 від « 07 » лютого 2022 р.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності та зручності освітнього процесу за рахунок впровадження власного програмного модуля для комунікації освітян, який базується на використанні обміну повідомлень в режимі реального часу.

Призначення роботи – запровадженя розробленого програмного застосунку для комунікації між учнями чи студентами та викладачами як окремого модуля та можливостей його інтеграції до платформи дистанційного та змішаного навчання.

4. Вихідні дані для проведення НДР

1. Cobley P., Schulz P. J. Theories and Models of Communication : довідник. Берлін : Walter de Gruyter, 2013. 452 с.
2. Maarten C. A., De Vries M. J. Science and Technology Education and Communication. Нью-Йорк : Springer, 2016. 200 с.
3. Narula U. Communication Models : довідник. Нью-Делі : Atlantic Publishers & Dist, 2006. 121 с.
4. Schramm’s Model of communication. *QS Study* : веб-сайт. URL: https://qsstudy.com/schramms- model-communication/ (дата звернення: 30.05.2022).

5. Технічні вимоги

Вхідні дані – особисті дані користувача, текст повідомлень; вихідні дані – надіслані та отримані повідомлення користувачів у веб-чаті.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Моделі комунікацій учасників освітнього процесу та їх програмна реалізація

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. Коваленко О.О.

Оригінальність	96
Схожість	4

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

 Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

 Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Козлюк Я. В.

Керівник роботи _____ Коваленко О. О

Додаток В – Лістинг програми

serverController.js

```
var models = require('../model/model.js');
var path = require('path');
var bodyParser = require('body-parser');
const bcrypt = require('bcryptjs');


module.exports = function (app,io){
    app.use( bodyParser.json() );
    app.use(bodyParser.urlencoded({
        extended: true
    }));

    app.get('/',function(req,res){
        res.sendFile(path.resolve(__dirname+"/../views/index.html"));
    });

    app.post('/register', async function(req,res){
        res.setHeader('Access-Control-Allow-Origin', '*');
        res.setHeader("Access-Control-Allow-Method","'GET, POST, OPTIONS, PUT, PATCH,
DELETE'");

        console.log(req.body.password);

        var hashedPassword;

        await bcrypt.hash(req.body.password, 7, (error, hash) => {
            if (error) {
                console.log('Error: ', error);
            } else {
                console.log(`Your encrypted password is: ${hash}`)

                hashedPassword = hash;

            }
        });

        let regArray = [];

        if (await models.user.findOne({name: req.body.name})) {
            regArray.push({"isNameExist": true});
        }

        if (await models.user.findOne({email: req.body.email})) {
            regArray.push({"isEmailExist": true});
```

```
        }

        if (await models.user.findOne({phone: req.body.phone})) {
            regArray.push({"isPhoneExist": true});
        }

        res.json(regArray);

        if (regArray.length == 0){
            var user= await models.user.create({
                "name":req.body.name,
                "group":req.body.group,
                "password":hashedPassword,
                "phone":req.body.phone,
                "email":req.body.email,
            });

            console.log(user);
        }

        regArray = [];

    });


    var name = null;
    var private = null;
    var users = {};
    var keys = {};

    app.post('/login',function(req,res){

        res.setHeader('Access-Control-Allow-Origin', '*');
        res.setHeader("Access-Control-Allow-Method","'GET, POST, OPTIONS, PUT, PATCH,
DELETE'");

        console.log(req.body.name);

        name = req.body.name;

        const password = req.body.password;

        models.user.findOne({name})
            .then(user => {
                if (!user) {
                    res.send("false");
                } else {
                    bcrypt.compare(password, user.password)
                    .then(isMatch => {
```

```
                    if (isMatch) {
                        console.log("log Path"+__dirname);
                        res.send("success");
                    } else {
                        res.send("false");
                    }
                });
            }
        });
});

io.on('connection',function(socket){
    console.log("Connection :User is connected  "+ name);
    console.log("Connection : " +socket.id);
    io.to(socket.id).emit('name', name);
    users[name] = socket.id;
    keys[socket.id] = name;
    console.log("Users list : "+users);
    console.log("keys list : "+keys);
    models.user.find({"name" : name},{friends:1,_id:0},function(err,doc){
        if(err){res.json(err);}
        else{
            friends=[];
            pending=[];
            all_friends=[];
            console.log("friends list: "+doc);
            list=doc[0].friends.slice();
            console.log(list);

            for(var i in list){
                if(list[i].status=="Friend"){
                    friends.push(list[i].name);
                }
                else if (list[i].status=="Pending"){
                    pending.push(list[i].name);
                }
                else{
                    continue;
                }
            }
            console.log("pending list: "+pending);
            console.log("friends list: "+friends);
            io.to(socket.id).emit('friend_list', friends);
            io.to(socket.id).emit('pending_list', pending);
            io.emit('users',users);
        }
    });

    socket.on('group message',function(msg){
```

```
            console.log(msg);
            io.emit('group',msg);
        });

        socket.on('private message',function(msg){
            console.log('message  :'+msg.split("#*@")[0]);
            models.messages.create({
                "message":msg.split("#*@")[1],
                "sender" :msg.split("#*@")[2],
                "reciever":msg.split("#*@")[0],
                "date" : new Date()});
            io.to(users[msg.split("#*@")[0]]).emit('private message', msg);
        });

        socket.on('disconnect', function(){
            delete users[keys[socket.id]];
            delete keys[socket.id];
            io.emit('users',users);
            console.log(users);
        });
    });

    app.post('/friend_request',function(req,res){
        res.setHeader('Access-Control-Allow-Origin', '*');
        res.setHeader("Access-Control-Allow-Method","'GET, POST, OPTIONS, PUT, PATCH,
DELETE'");
        friend=true;
        models.user.find({"name" :
req.body.my_name,"friends.name":req.body.friend_name},function(err,doc){
            if(err){res.json(err);}
            else if(doc.length!=0){
                console.log("Friend request : "+doc.length);
                console.log("Friend request : friend request already sent "+doc);
                res.send("Friend request already sent ");
            }
            else{
                console.log("Friend request : "+doc.length);
                models.user.update({
                    name:req.body.my_name
                },{
                    $push:{
                        friends:{
                            name: req.body.friend_name,
                            status: "Pending"
                        }
                    }
                },{
                    upsert:true
                },function(err,doc){
```

```
                    if(err){res.json(err);}
                });
                io.to(users[req.body.friend_name]).emit('message', req.body);
            }
        });
    });

    app.post('/friend_request/confirmed',function(req,res){
        console.log("friend request confirmed : "+req.body);
        if(req.body.confirm=="Yes"){
            models.user.find({
                "name" : req.body.friend_name,
                "friends.name":req.body.my_name
            },function(err,doc){
                if(err){
                    res.json(err);
                }
                else if(doc.length!=0){
                    console.log("Friend request confirmed : "+doc.length);
                    console.log("Friend request confirmed : friend request already
sent "+doc);
                    res.send("Friend request already accepted");
                }
                else{
                    models.user.update({
                        "name":req.body.my_name,
                        "friends.name":req.body.friend_name
                    },{
                        '$set':{
                            "friends.$.status":"Friend"
                        }
                    },function(err,doc){
                        if(err){res.json(err);}
                        else{
                            console.log("friend request confirmed : Inside yes
confirmed");
                            io.to(users[req.body.friend_name]).emit('friend',
req.body.my_name);
                            io.to(users[req.body.my_name]).emit('friend',
req.body.friend_name);
                        }
                    });
                    models.user.update({
                        name:req.body.friend_name
                    },{
                        $push:{
                            friends:{
                                name: req.body.my_name,
                                status: "Friend"
```

```
                        }
                    }
                },{upsert:true},function(err,doc){
                    if(err){res.json(err);}
                });
            }
        });
    }
    else{

        console.log("friend request confirmed : Inside No confirmed");
        models.user.update({
            "name":req.body.my_name
        },{
            '$pull':{
                'friends':{
                    "name":req.body.friend_name,
                }
            }
        },function(err,doc){
        if(err){res.json(err);}
        else{
            console.log("No");
        }
    });
    }
});
}
```

server.js

```
var express=require('express');
var app=express();
var http=require('http').Server(app);
var io = require('socket.io')(http);
var ip = require('ip');
app.use(express.static('./'));

require("./controller/controller.js")(app,io);

http.listen(8080,function(){
    console.log("Node Server is setup and it is listening on
http://"+ip.address()+":8080");
})
```

clientController.js

```
var app = angular.module('myapp',['ngMaterial','ui.router','ngStorage']);
```

```
app.factory('socket', ['$rootScope', function($rootScope) {
    var socket = io.connect();

    return {
        on: function(eventName, callback){
            socket.on(eventName, callback);
        },
        emit: function(eventName, data) {
            socket.emit(eventName, data);
        }
    };
}]);

app.config(['$stateProvider','$urlRouterProvider',function($stateProvider,$urlRouterP
rovider){
    $urlRouterProvider.otherwise('/');
    $stateProvider
    .state('login',{
        url:'/',
        views:{
            'body':{
                templateUrl:'/views/login.html',
                controller:'registerController'
            }
        }
    })
    .state('loggedin',{
        url:'/login',
        views:{
            'body':{
                templateUrl:'/views/chat.html',
                controller:'myController'
            }
        }
    })
}]);


app.directive('myEnter', function () {
    return function (scope, element, attrs) {
        element.bind("keydown keypress", function (event) {
            if(event.which === 13) {
                scope.$apply(function (){
                    scope.$eval(attrs.myEnter);
                });

                event.preventDefault();
            }
        });
```

```
    };
});


app.controller('myController',['$scope','socket','$http','$mdDialog','$compile','$loc
ation','$state','$localStorage',
'$sessionStorage',function($scope,socket,$http,$mdDialog,$compile,$location,$state,$l
ocalStorage, $sessionStorage){
    url= location.host;
    $scope.users=[];
    $scope.online_friends=[];
    $scope.allfriends=[];
    $scope.messages={};
    var monthNames = ["Січень", "Лютий", "Березень", "Квітень", "Травень",
"Червень","Липень", "Серпень", "Вересень", "Жовтень","Листопад", "Грудень"];

    socket.on('name', function(data) {
        $scope.user = data;
        console.log("Get name : "+$scope.user);
    });


    socket.on('friend_list', function(data) {
        console.log("Friends list : "+data);
        $scope.$apply(function () {
            $scope.allfriends.push.apply($scope.allfriends,data);
        });
        console.log("Friends list : "+$scope.allfriends);
    });

    socket.on('pending_list', function(data) {

    });

    socket.on('users', function(data) {
        console.log("users list : "+data);
        $scope.$apply(function () {
            $scope.users=[];
            $scope.online_friends=[];
            for(var i in data){
                console.log("users list : "+i);
                if (i!=$scope.user){
                    console.log(i);
                    console.log("users list : "+$scope.allfriends);
                    if ( $scope.allfriends.includes(i) ){
                        $scope.online_friends.push(i);
                    }
                    else{
                        $scope.users.push(i);
                    }
```

```
            }
        }
        console.log("users list : "+$scope.allfriends);
        console.log("users list : "+$scope.users);
        console.log("users list : "+$scope.online_friends);
    });
});

$scope.confirm=function(){
    var data = {
        "friend_name":$scope.friend,
        "my_name":$scope.user
    };


    $http({method: 'POST',url:'http://'+url+'/friend_request',data})
        .success(function (data) {
        console.log(data)
    })
        .error(function (data) {
        console.log(data)
    });
};

$scope.showConfirm = function(data) {

    var confirm = $mdDialog.confirm()
    .title(" Запит на приєднання ")
    .textContent(data.my_name+' хоче приєднатися. Приєднати?')
    .ariaLabel('Lucky day')
    .ok('Так')
    .cancel('Ні');

    $mdDialog.show(confirm).then(function() {
        data['confirm']="Yes";
        $http({method: 'POST',url:'http://'+url+'/friend_request/confirmed', data
        })
    }, function() {
        data['confirm']="No";
        $http({method: 'POST',url:'http://'+url+'/friend_request/confirmed', data
        })
    });
};

socket.on('message', function(data) {
    $scope.showConfirm(data);
});

socket.on('friend', function(data) {
```

```
        console.log("Connection Established"+data);
        $scope.$apply(function () {
            if (!$scope.online_friends.includes(data)){
                console.log(data);
                $scope.online_friends.push(data);
                $scope.users.splice($scope.users.indexOf(data),1);
            }

        });
    });


    $scope.friend_request = function(user) {
        $scope.friend = user;
    };

    var getDate=function(){
        date = new Date();
        hour=date.getHours();

        form_date=monthNames[date.getMonth()]+" "+date.getDate()+",
"+hour+":"+date.getMinutes();
        return form_date;
    }


    socket.on('group', function(data) {
        var div = document.createElement('div');
        if(data.split("#*@")[1]!=$scope.user){
            div.innerHTML='<div class="direct-chat-msg right">\
                           <div class="direct-chat-info clearfix">\
                           <span class="direct-chat-name pull-
right">'+data.split("#*@")[1]+'</span>\
                           <span class="direct-chat-timestamp pull-
left">'+getDate()+'</span>\
                           </div>\
                           <img class="direct-chat-img" src="https://cdn-icons-
png.flaticon.com/128/6998/6998058.png" alt="message user image">\
                           <div class="direct-chat-text">'
                           +data.split("#*@")[0]+
                           '</div>\
                           </div>';
            document.getElementById("group").appendChild(div);
            document.getElementById("group").scrollTop=document.getElementById("group
").scrollHeight;
        }
    });

    $scope.group_message= function(message){
```

```
        div = document.createElement('div');
        div.innerHTML='<div class="direct-chat-msg"> \
                        <div class="direct-chat-info clearfix">\
                        <span class="direct-chat-name pull-
left">'+$scope.user+'</span>\
                        <span class="direct-chat-timestamp pull-
right">'+getDate()+'</span>\
                        </div>\
                        <img class="direct-chat-img" src="https://cdn-icons-
png.flaticon.com/128/2489/2489278.png" alt="message user image">\
                        <div class="direct-chat-text">'
                        +message+
                        '</div>\
                        </div>';
        document.getElementById("group").appendChild(div);
        document.getElementById("group").scrollTop=document.getElementById("group").s
crollHeight;
        socket.emit('group message',message+"#*@"+$scope.user);
        $scope.groupMessage=null;
    }

    var insertMessage = function(from,to,msg){
        console.log(from + " " + to);
        if (to in $scope.messages){
            if ($scope.messages[to].length>25){
                $scope.messages[to].splice(0,1);
            }
        }
        else{
            $scope.messages[to]=[];
        }
        $scope.messages[to].push({
            "sender":from,
            "msg" : msg,
            "date" : getDate()
        });
        localStorage.setItem(to,JSON.stringify($scope.messages[to]));
        localStorage.setItem(from,JSON.stringify($scope.messages[from]));
        console.log(localStorage.getItem(to));
    }

    socket.on('private message', function(data) {
        var div = document.createElement('div');
        div.innerHTML='<div class="direct-chat-msg right">\
                        <div class="direct-chat-info clearfix">\
                        <span class="direct-chat-name pull-right" style="position:
absolute !important; right: 50px !important;">'+data.split("#*@")[2]+'</span>\
                        <span class="direct-chat-timestamp pull-
left">'+getDate()+'</span>\
```

```
                        </div>\
                        <img class="direct-chat-img" src="https://cdn-icons-
png.flaticon.com/128/6998/6998058.png" alt="message user image">\
                        <div class="direct-chat-text">'
                        +data.split("#*@")[1]+
                        '</div>\
                        </div>';
        var chat_box=document.getElementById(data.split("#*@")[2]);
        console.log(chat_box);
        if(chat_box!=null){
            chat_box.appendChild(div);
        }
        else{
            $scope.chat_popup(data.split("#*@")[2]);
            document.getElementById(data.split("#*@")[2]).appendChild(div);
        }
        insertMessage(data.split("#*@")[2],data.split("#*@")[2],data.split("#*@")[1])
;
        document.getElementById(data.split("#*@")[2]).scrollTop=document.getElementBy
Id(data.split("#*@")[2]).scrollHeight;
    });

    $scope.send_message=function(chat,message){
        console.log(chat);
        div = document.createElement('div');
        div.innerHTML='<div class="direct-chat-msg"> \
                        <div class="direct-chat-info clearfix">\
                        <span class="direct-chat-name pull-
left">'+$scope.user+'</span>\
                        <span class="direct-chat-timestamp pull-
right">'+getDate()+'</span>\
                        </div>\
                        <img class="direct-chat-img" src="https://cdn-icons-
png.flaticon.com/128/2489/2489278.png" alt="message user image">\
                        <div class="direct-chat-text">'
                        +message+
                        '</div>\
                        </div>';
        document.getElementById(chat).appendChild(div);
        document.getElementById(chat).scrollTop=document.getElementById(chat).scrollH
eight;
        socket.emit('private
message',chat+"#*@"+message+"#*@"+$scope.user+"#*@"+getDate());
        insertMessage($scope.user,chat,message);
        $scope.message=null;
    }


    popups=[];
```

```
    $scope.chat_popup = function(chat_friend){
        console.log(chat_friend);
        console.log(popups);
        for(var iii = 0; iii < popups.length; iii++)
        {
            if($scope.chat_friend == popups[iii])
            {
                popups.splice(iii,1);

                popups.push(chatfriend);

                display_popups();
            }
        }

        console.log($scope.messages);
        console.log($scope.messages[chat_friend]);

        div = document.createElement('div');
        div.innerHTML='<div class="popup-box popup-box-on chat-popup"
id="'+chat_friend+'01">\
                        <div class="popup-head">\
                        <div class="popup-head-left pull-left"><img src="https://cdn-
icons-png.flaticon.com/128/6558/6558852.png" style="border: 0px;"
alt="pic">'+chat_friend+'</div>\
                        <div class="popup-head-right pull-right">\
                        <div class="btn-group">\
                        <button class="chat-header-button" data-toggle="dropdown"
type="button" aria-expanded="false">\
                        <i class="glyphicon glyphicon-cog"></i> </button>\
                        <ul role="menu" class="dropdown-menu pull-right">\
                        <li><a href="">Заблокувати</a></li>\
                        <li><a href="">Очистити чат</a></li>\
                        </ul>\
                        </div>\
                        <button  ng-click="close_chat(\''+chat_friend+'\')"
class="chat-header-button pull-right" type="button">  <i class="glyphicon glyphicon-
remove"></i></button>\
                        </div>\
                        </div>\
                        <div class="box-body popup-messages">\
                        <div class="direct-chat-messages" id="'+chat_friend+'" >\
                        </div>\
                        </div>\
                        <div class="popup-messages-footer">\
                        <textarea id="status_message" placeholder="Напишіть
повідомлення..." rows="10" cols="40" ng-model="message" my-
enter="send_message(\''+chat_friend+'\',\'{{message}}\')"></textarea>\
```

```
                        <div class="btn-footer">\
                        <button class="bg_none"><i class="glyphicon glyphicon-
film"></i> </button>\
                        <button class="bg_none"><i class="glyphicon glyphicon-
camera"></i> </button>\
                        <button class="bg_none"><i class="glyphicon glyphicon-
paperclip"></i> </button>\
                        <button class="bg_none pull-right" ng-
click="send_message('+chat_friend+',message)"><i class="glyphicon  glyphicon-thumbs-
up"></i> </button>\
                        </div>\
                        </div>\
                        </div>';
        $compile(div)($scope);


        if(popups.length>1){
            document.getElementById(chat_friend+"01").className=document.getElementBy
Id(popups[popups.length-2]+"01").className.replace(/(?:^|\s)popup-box-on(?!\S)/g ,
'');
        }
        var body=document.getElementsByTagName("body")[0];
        body.appendChild(div);
        if(localStorage.getItem(chat_friend)!==null){
            $scope.messages[chat_friend] =
JSON.parse(localStorage.getItem(chat_friend));
        }
        if($scope.messages[chat_friend] != undefined){
            for(var i=0; i<$scope.messages[chat_friend].length; i++){
                console.log($scope.messages[chat_friend][i].sender);
                if($scope.messages[chat_friend][i].sender==$scope.user){
                    div = document.createElement('div');
                    div.innerHTML='<div class="direct-chat-msg"> \
<div class="direct-chat-info clearfix">\
<span class="direct-chat-name pull-
left">'+$scope.messages[chat_friend][i].sender+'</span>\
<span class="direct-chat-timestamp pull-
right">'+$scope.messages[chat_friend][i].date+'</span>\
</div>\
<img class="direct-chat-img" src="https://cdn-icons-
png.flaticon.com/128/2489/2489278.png" alt="message user image">\
<div class="direct-chat-text">'
    +$scope.messages[chat_friend][i].msg+
    '</div>\
</div>';
                    document.getElementById(chat_friend).appendChild(div);
                    document.getElementById(chat_friend).scrollTop=document.getElemen
tById(chat_friend).scrollHeight;
                }
```

```
                else{
                    div = document.createElement('div');
                    div.innerHTML='<div class="direct-chat-msg right">\
<div class="direct-chat-info clearfix">\
<span class="direct-chat-name pull-right" style="position: absolute !important;
right: 50px !important;">'+$scope.messages[chat_friend][i].sender+'</span>\
<span class="direct-chat-timestamp pull-
left">'+$scope.messages[chat_friend][i].date+'</span>\
</div>\
<img class="direct-chat-img" src="https://cdn-icons-
png.flaticon.com/128/6998/6998058.png" alt="message user image">\
<div class="direct-chat-text">'
    +$scope.messages[chat_friend][i].msg+
    '</div>\
</div>';
                    document.getElementById(chat_friend).appendChild(div);
                    document.getElementById(chat_friend).scrollTop=document.getElemen
tById(chat_friend).scrollHeight;
                }
            }
        }
        console.log($scope.online_friends);
        // $compile(body)($scope);
        popups.push(chat_friend);

    }


    $scope.close_chat= function(chat_friend)
    {
        chat_box=null;
        console.log(chat_friend);
        console.log(popups);

        for(var iii = 0; iii < popups.length; iii++)
        {
            if(chat_friend == popups[iii])
            {
                console.log("sss");
                var chat_box=document.getElementById(popups[popups.length-1]+"01");
                chat_box.parentElement.removeChild(chat_box);
                popups.splice(iii,1);
            }
        }
    }
}]);


var encrypt = function(str){
    h = 7;
```

```
        letters = "abcdefghijklmnopqrstuvwxyz-_1234567890@!#$%&*.,"
        for (var i = 0; i < str.length; i++){
            h = (h * 37 + letters.indexOf(str[i]));
        }
        return h + "";
}


app.controller('registerController',['$scope', '$http','$state',function($scope,
$http,$state){
    url= location.host;

    $scope.user={
        'group':'',
        'name':'',
        'password':'',
        'email':'',
        'phone':''
    };

    $scope.login_data={
        'name':'',
        'password':''
    };

    $scope.Register = function(){

        console.log($scope.user.password);

        let tempObj = {
            group: $scope.user.group,
            name: $scope.user.name,
            password: encrypt($scope.user.password),
            email: $scope.user.email,
            phone: $scope.user.phone
        };

        var reg = document.getElementById("reg");

        let isPhoneCorrect = true,
            isNameCorrect = true,
            isEmailCorrect = true,
            isPasswordCorrect = true,
            isGroupCorrect = true;

        if($scope.user.name.length <= 25 && $scope.user.name.length >= 5){
            for(let char of $scope.user.name){
                if(!isNaN(char)){
                    alert("Ім'я не може містити цифри");
                    isNameCorrect = false;
```

```
                    break;
                }
            }
        } else {
            alert("Ім'я повинно містити не менше 5 та не більше 25 символів");
            isNameCorrect = false;
        }

        if($scope.user.group.length == 0){
            alert("Поле групи не може бути порожнім");
            isGroupCorrect = false;
        }

        if($scope.user.password.length == 0){
            alert("Пароль не може бути порожнім");
            isPasswordCorrect = false;
        }

        if($scope.user.email.length != 0){
            if($scope.user.email.indexOf("@") == -1 || $scope.user.email.indexOf(".")
== -1){
                alert('Поле повинно містити емейл в форматі "user1@gmail.com"');
                isEmailCorrect = false;
            }
        } else {
            alert("Поле емейлу не може бути порожнім");
            isEmailCorrect = false;
        }

        if($scope.user.phone.length == 10){
            for(let dig of $scope.user.phone){
                if(isNaN(dig)){
                    alert("Номер повинен містити тільки цифри");
                    isPhoneCorrect = false;
                    break;
                }
            }
        } else {
            alert('Номер повинен складатися із 10 цифр в форматі "0962035661"');
            isPhoneCorrect = false;
        }

        if(isPhoneCorrect && isNameCorrect && isEmailCorrect && isPasswordCorrect &&
isGroupCorrect){

            $http({method: 'POST',url:'http://'+url+'/register', data:tempObj})

            .success(function (data) {
            console.log(data)
```

```
        })

        .error(function (data) {
        console.log(data)
        })

        .then(res => {
            if(res.data.length > 0) {

                for(let obj of res.data) {

                    if(obj.isNameExist) {
                        alert("Користувач з таким іменем вже існує");
                    }

                    if(obj.isEmailExist) {
                        alert("Користувач з таким емейлом вже існує");
                    }

                    if (obj.isPhoneExist) {
                        alert("Користувач з таким номером вже існує");
                    }
                }
            } else {
                reg.setAttribute("data-dismiss", "modal");
                reg.click();
                reg.removeAttribute("data-dismiss");
            }

            console.log(res.data);
        });
    }
}

$scope.login = function(){

    console.log($scope.login_data);

    let tempObj = {
        name: $scope.login_data.name,
        password: encrypt($scope.login_data.password)
    };

    $http({ method: 'POST', url:'http://'+url+'/login', data:tempObj })

        .success(function (data) {
        if(data == "success"){
            console.log("Inside success login");
            $state.go('loggedin');
```

```
            }
            })

            .error(function (data) {
            console.log(data)
            })

            .then(res => {
                if(res.data == "false"){
                    alert("Неправильне ім'я або пароль");
                }
            });
    }
}]);
```

model.js

```
var mongoose = require('mongoose');

var Schema = mongoose.Schema;

mongoose.connect(
  'mongodb+srv://Yaroslav:admin@chat.7q9dy.mongodb.net/chat?retryWrites=true&w=majori
ty',
  {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  }
);

mongoose.connection.on('open', function (ref) {
  console.log('Connected to mongo server.');
});
mongoose.connection.on('error', function (err) {
  console.log('Could not connect to mongo server!');
  console.log(err);
});

module.exports.user = mongoose.model(
  'User',
  new Schema(
    {
      group: String,
      name: String,
      password: String,
      phone: String,
      email: String,
      friends: [],
    },
```

```
    { strict: false }
  )
);
module.exports.online = mongoose.model(
  'online',
  new Schema({
    name: String,
    connection_id: String,
  })
);
module.exports.messages = mongoose.model(
  'message',
  new Schema({
    message: String,
    sender: String,
    reciever: String,
    date: Date,
  })
);
```

Додаток Г – Графічна частина

ГРАФІЧНА ЧАСТИНА
МОДЕЛІ КОМУНІКАЦІЙ УЧАСНИКІВ ОСВІТНЬОГО ПРОЦЕСУ ТА ЇХ ПРОГРАМНА РЕАЛІЗАЦІЯ

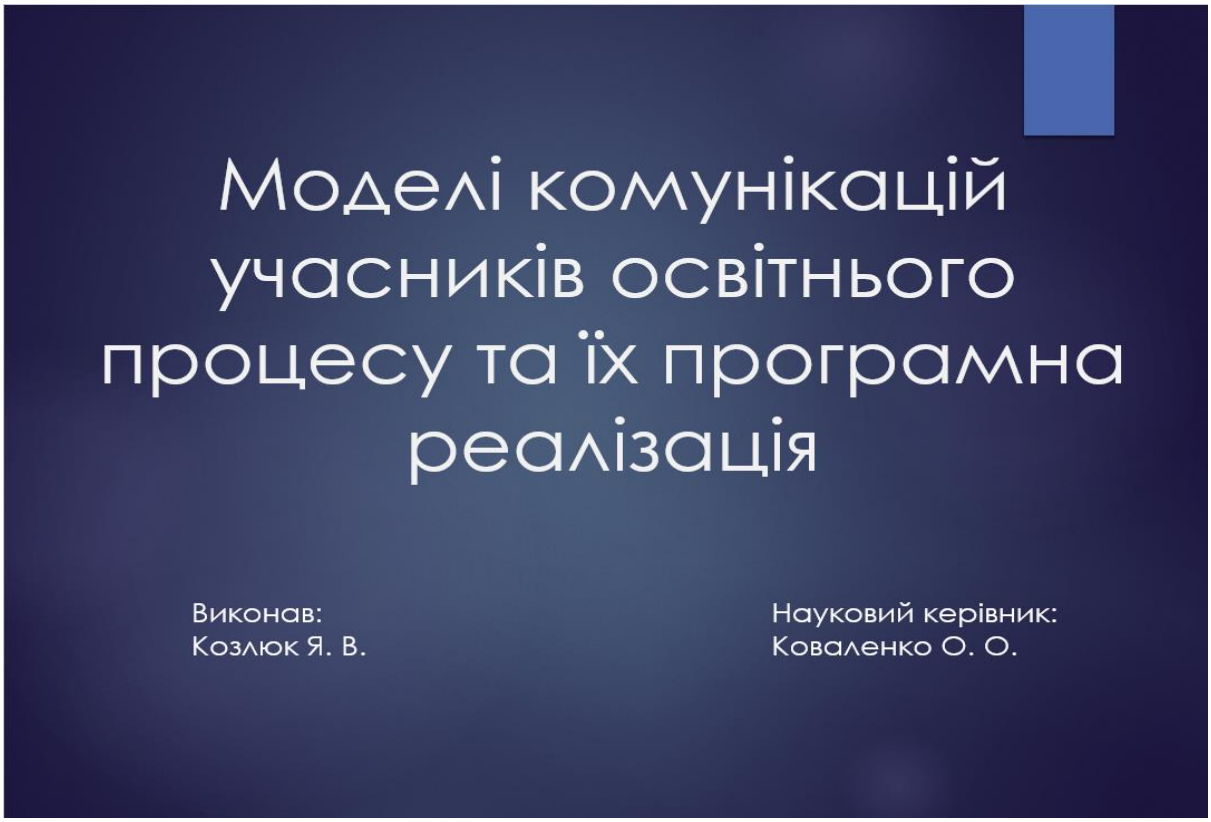

Рисунок Г.1 – Назва роботи

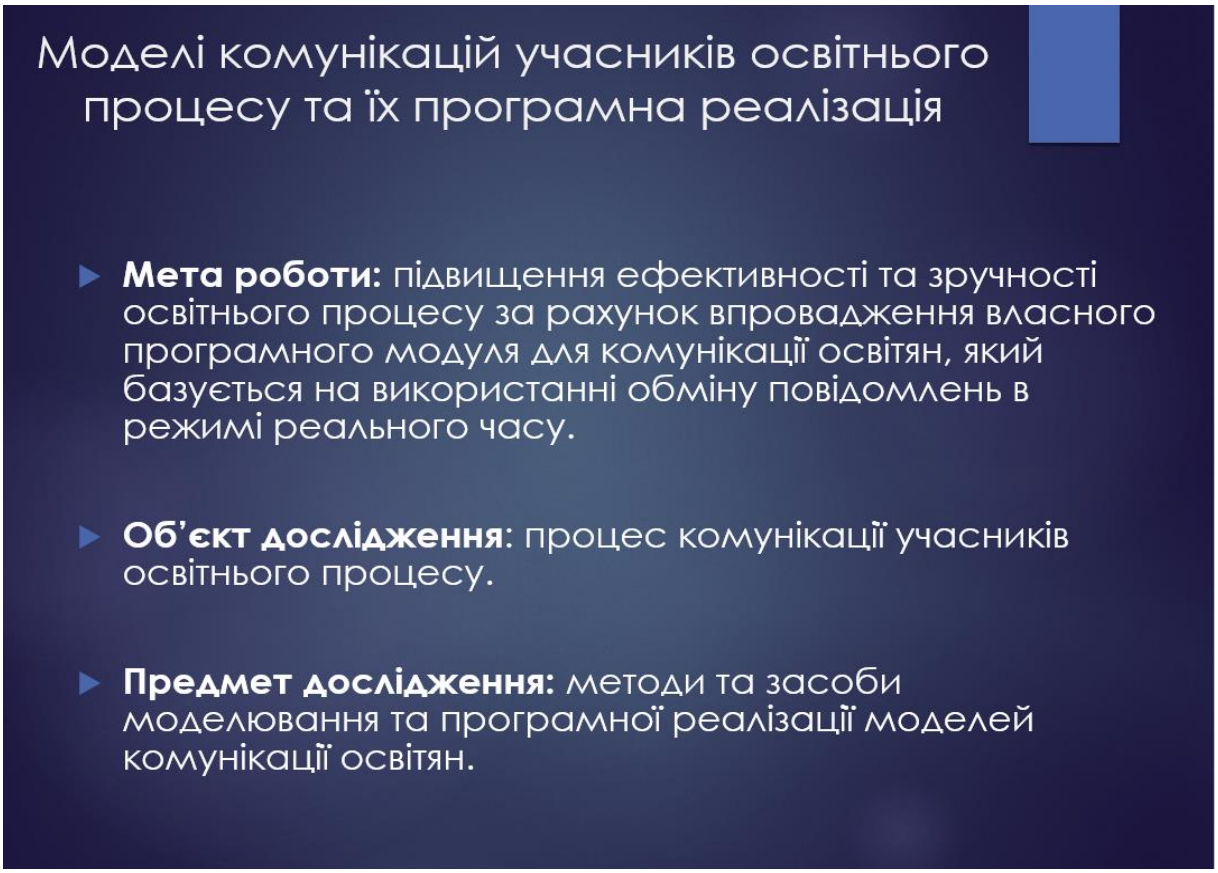

Рисунок Г.2 – Мета, об'єкт і предмет дослідження

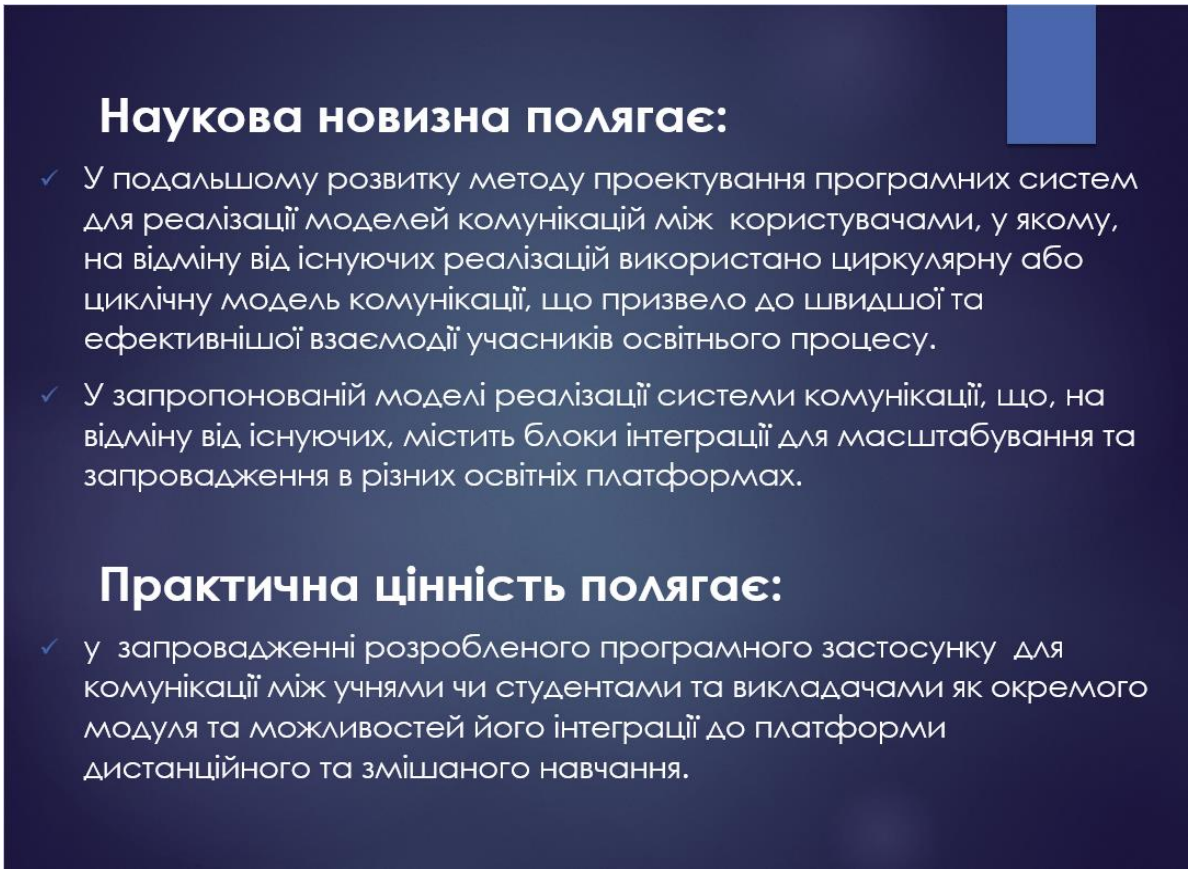

Рисунок Г.3 – Наукова новизна та практична цінність

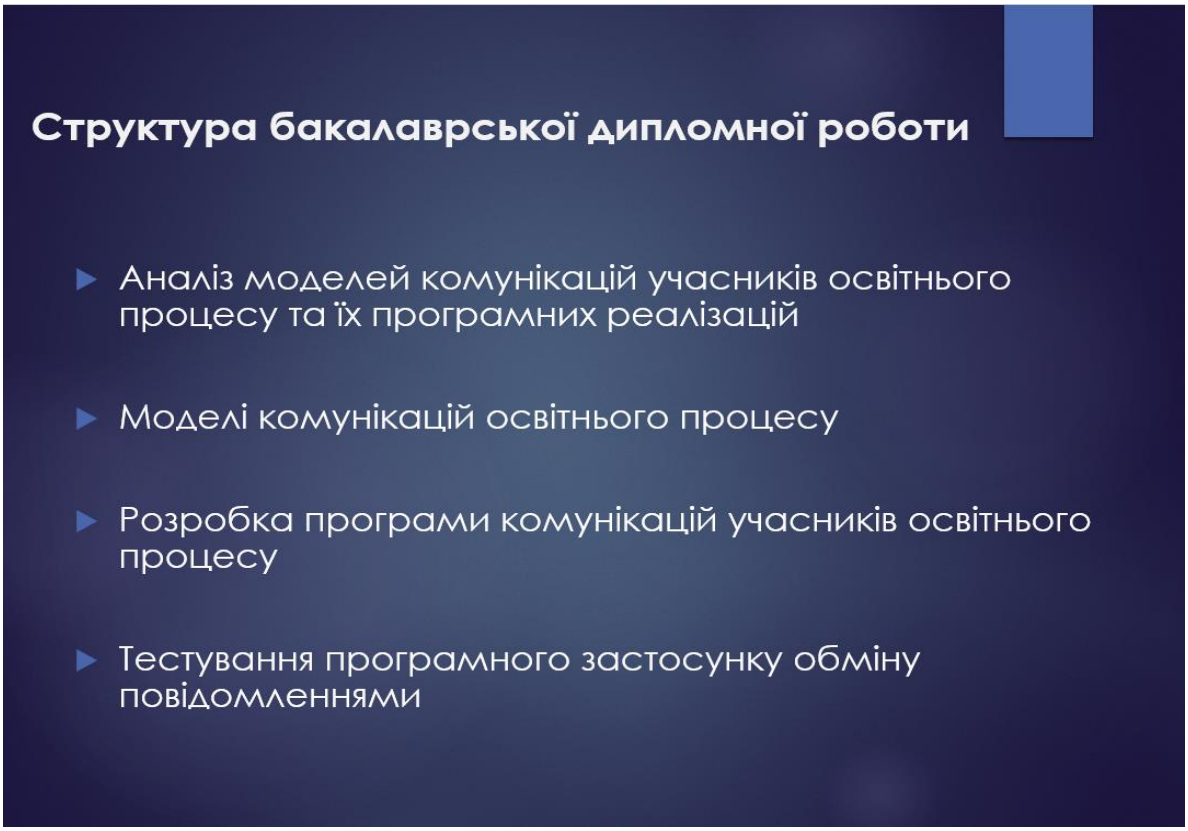

Рисунок Г.4 – Структура бакалаврської дипломної роботи

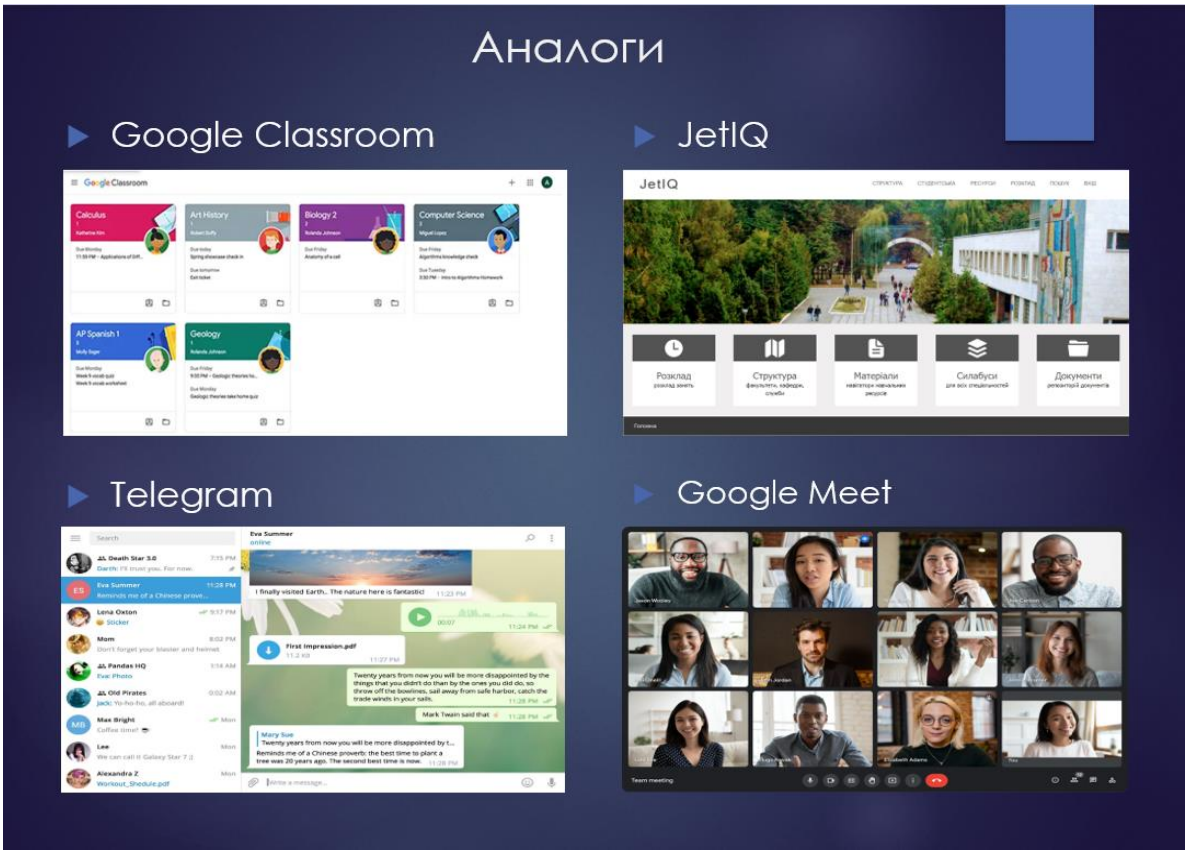

Рисунок Г.5 – Аналоги

Рисунок Г.6 – Модель організації освітнього процесу

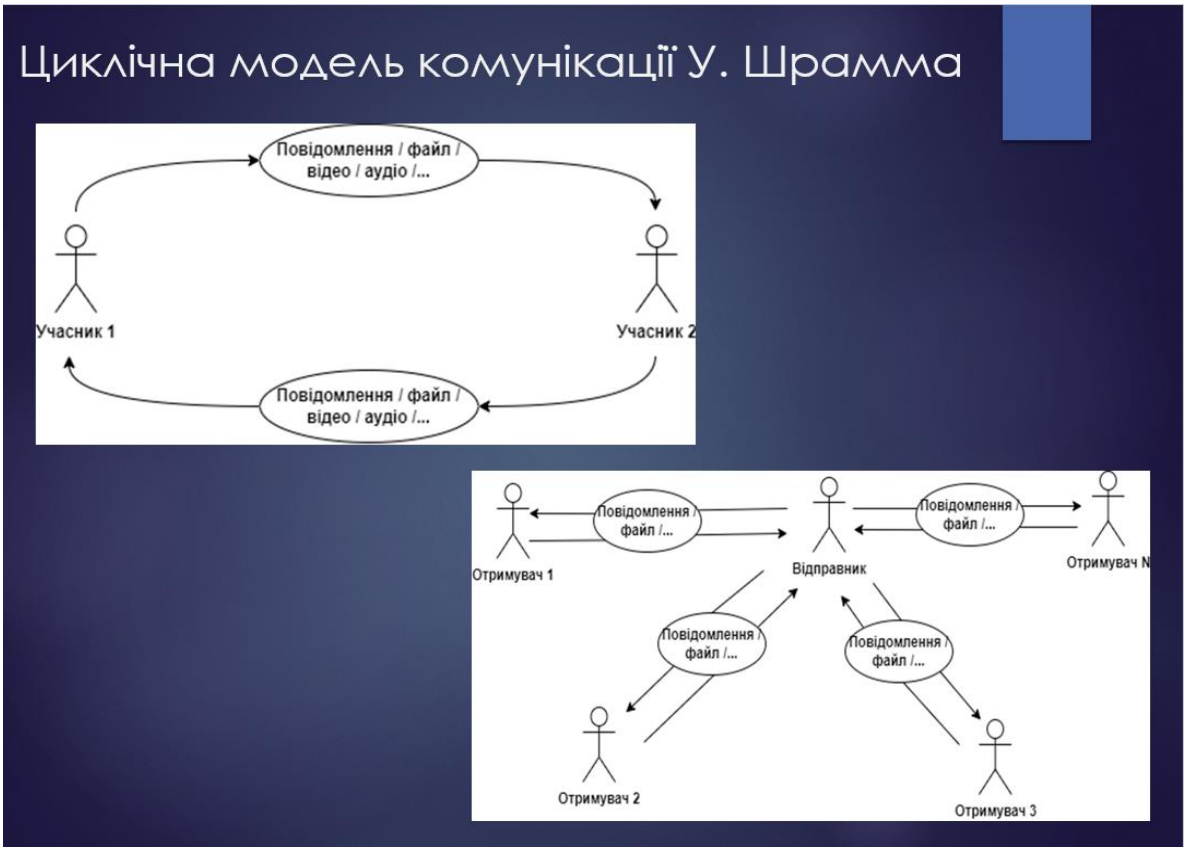

Рисунок Г.7 – Циклічна модель комунікації У. Шрамма

Рисунок Г.8 – Загальний алгоритм роботи застосунку

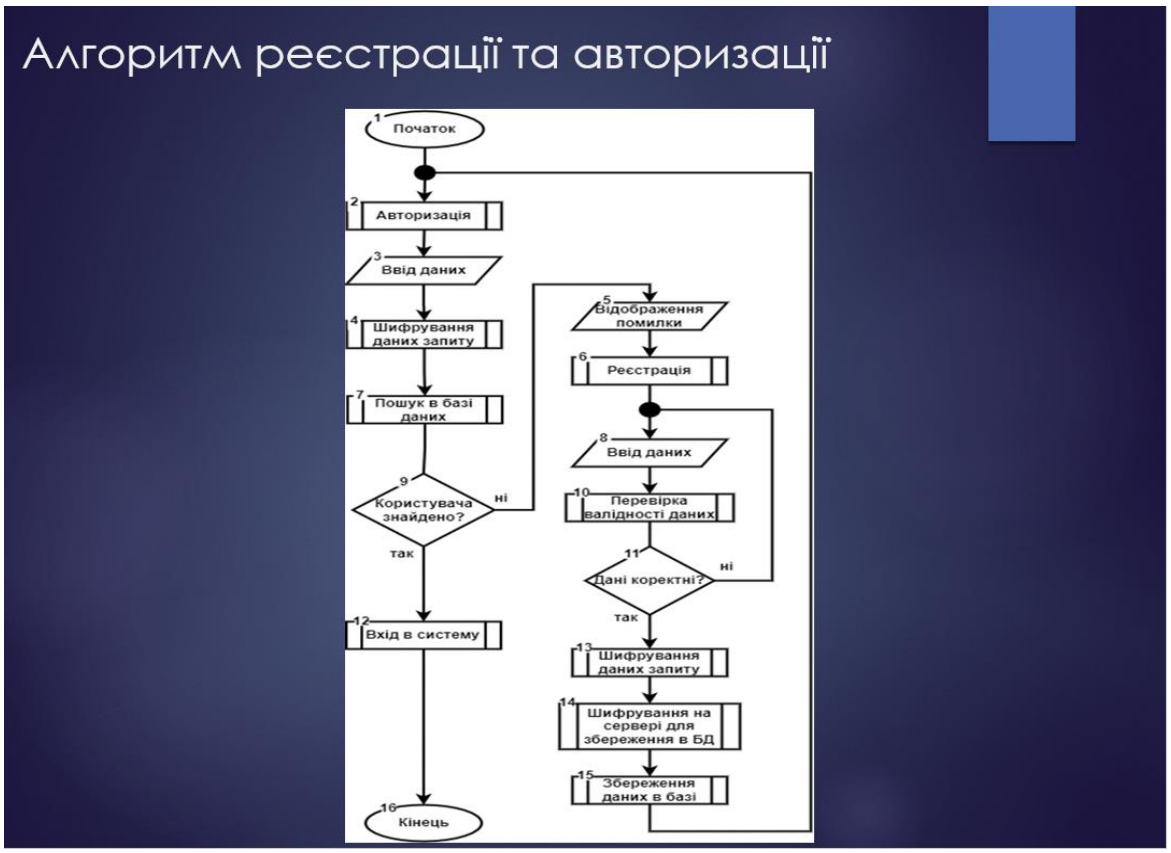

Рисунок Г.9 – Алгоритм реєстрації та авторизації

Рисунок Г.10 – Алгоритм програмного модуля

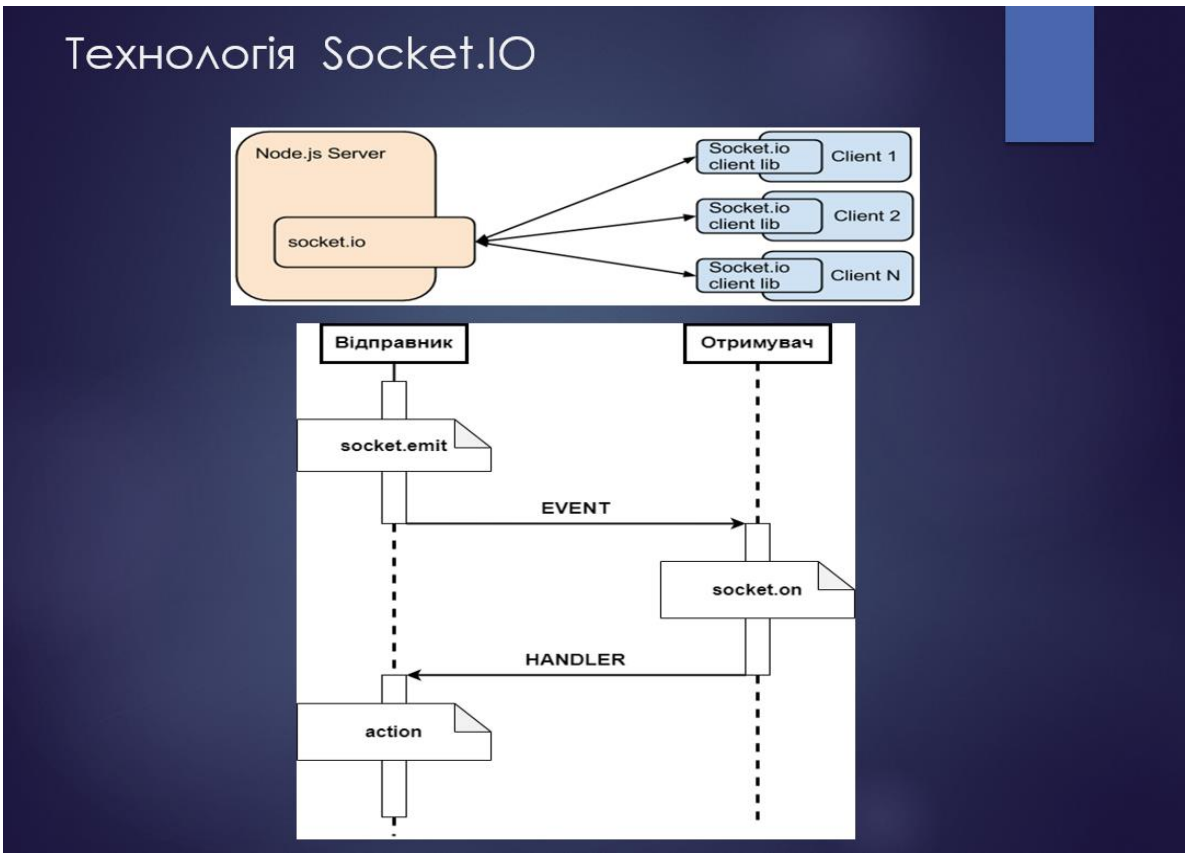

Рисунок Г.11 – Технологія Socket.IO

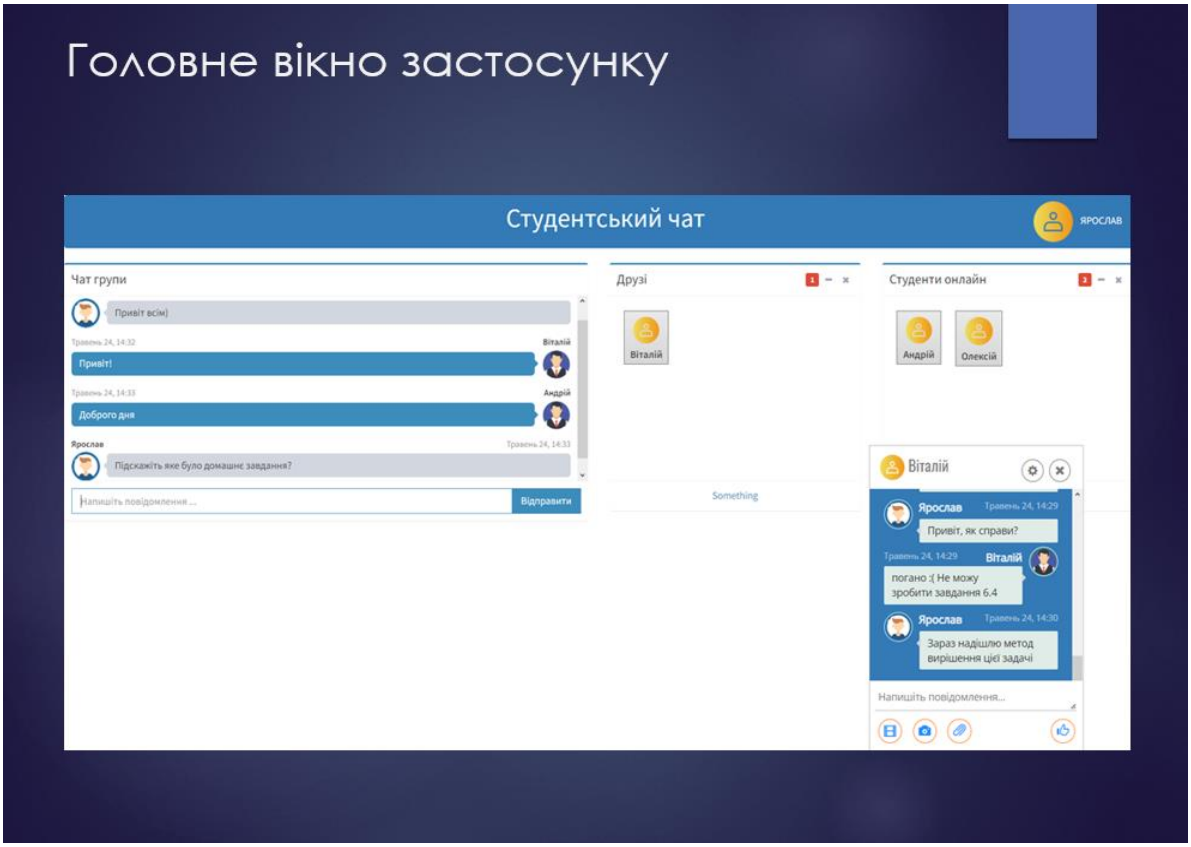

Рисунок Г.12 – Головне вікно застосунку

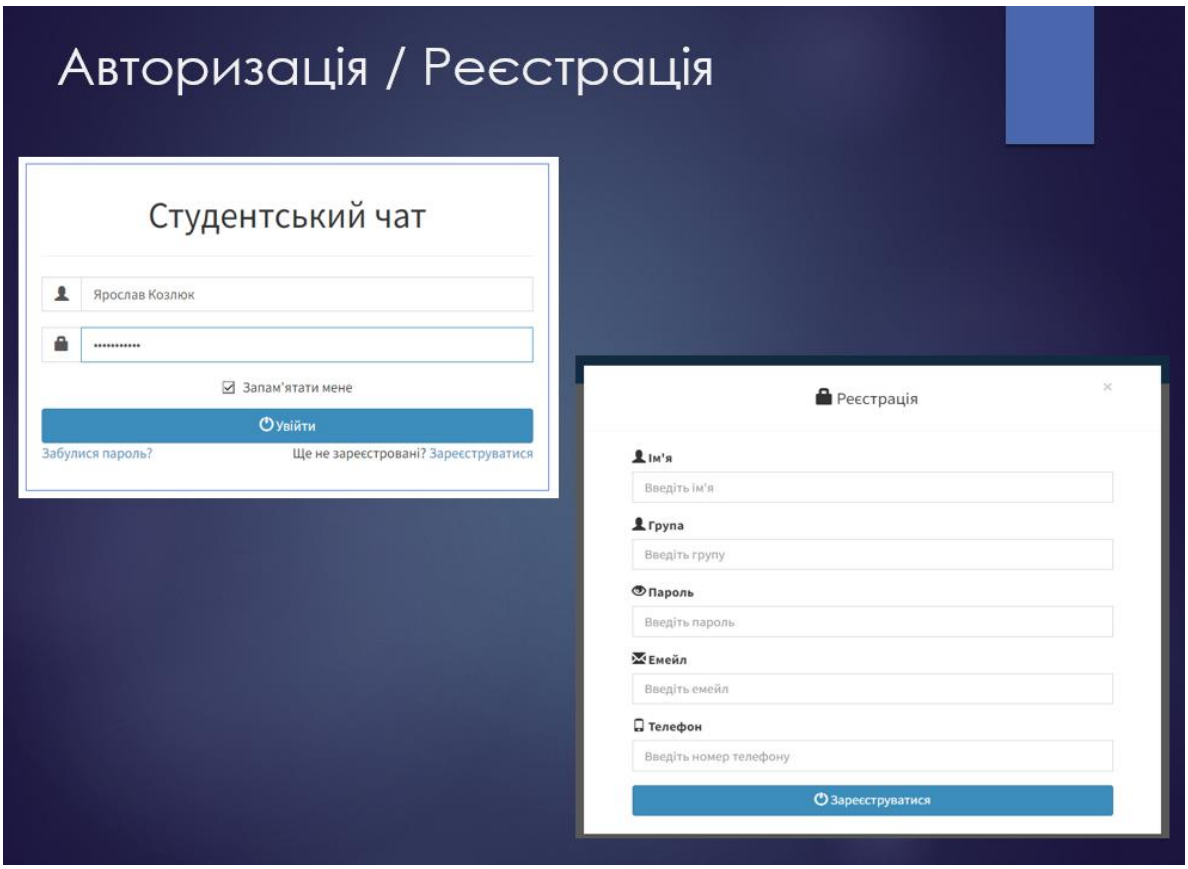

Рисунок Г.13 – Форми реєстрації і авторизації

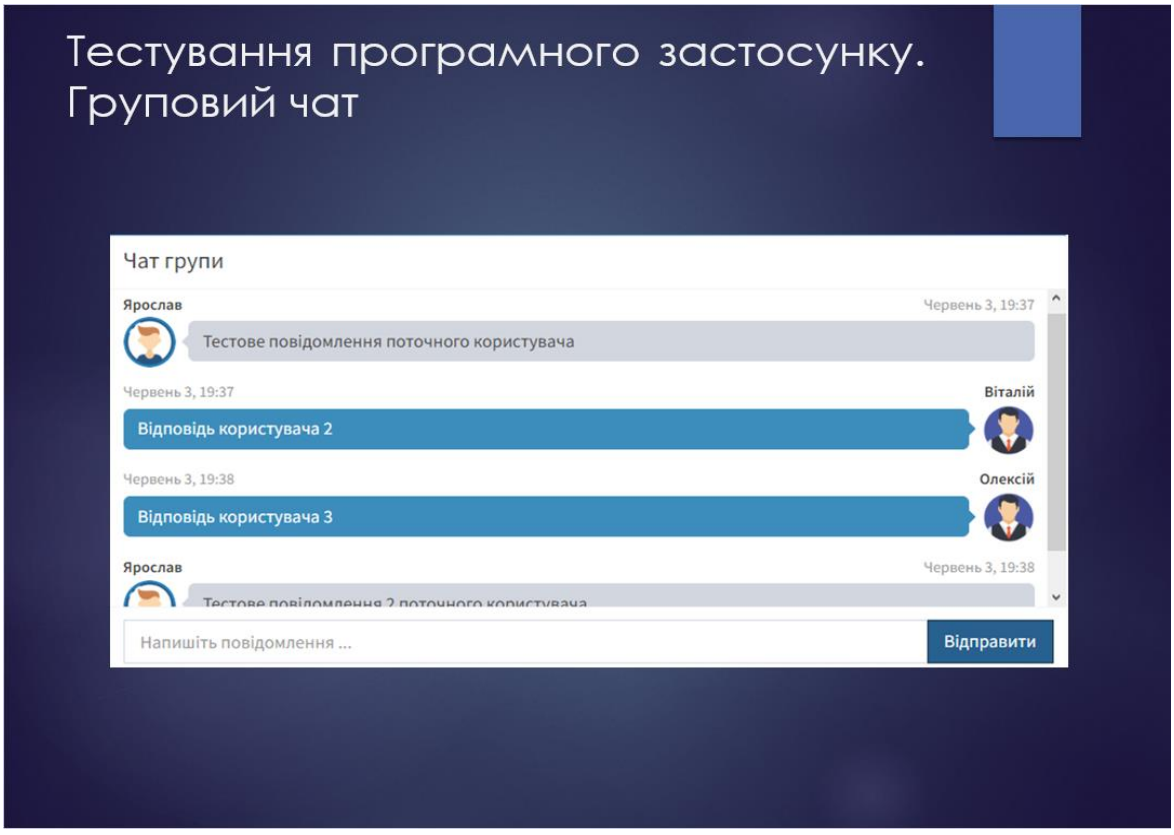

Рисунок Г.14 – Тестування групового чату

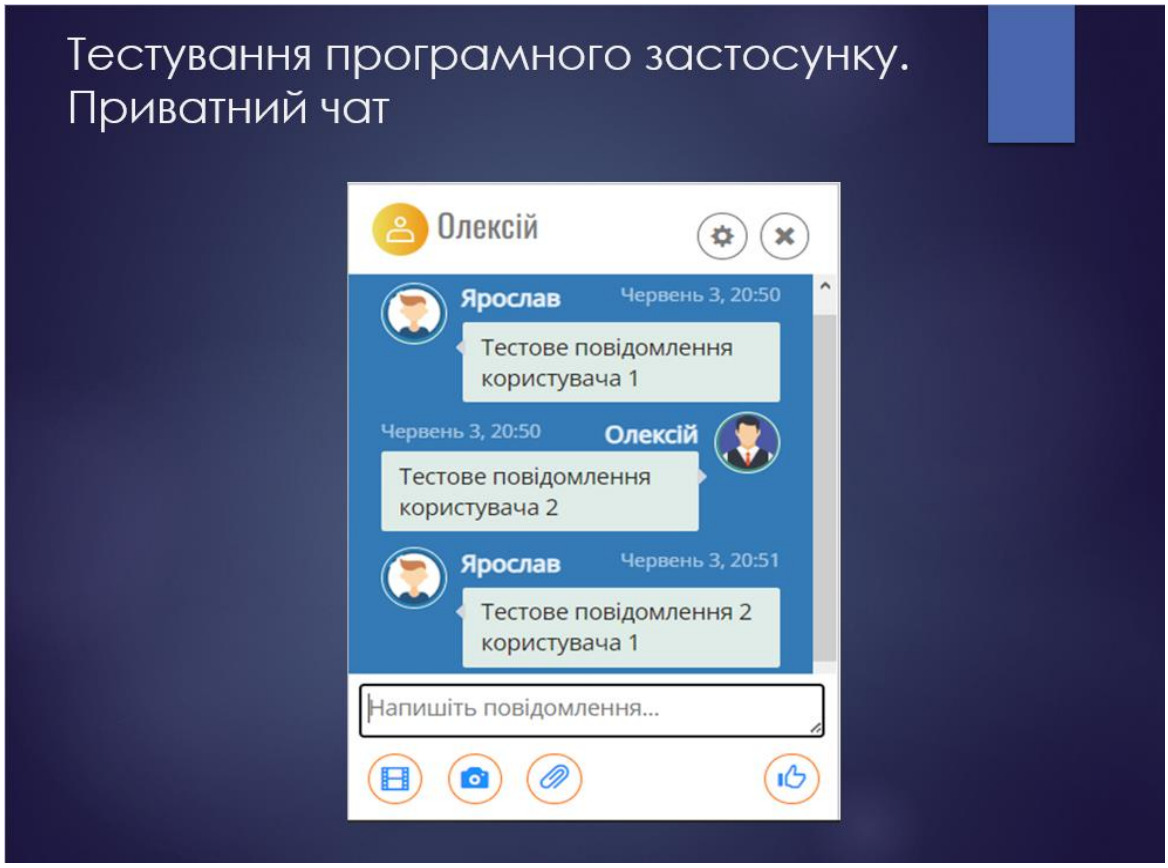

Рисунок Г.15 – Тестування приватного чату

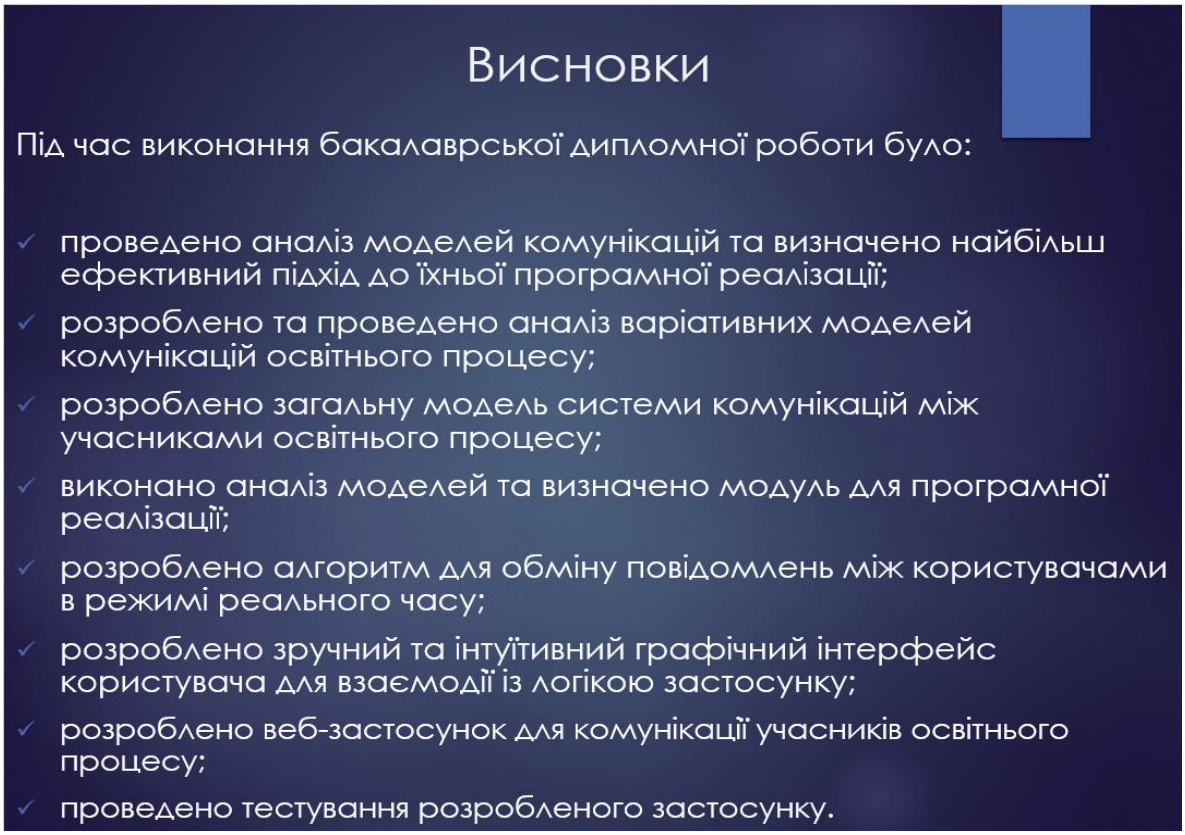

Рисунок Г.16 – Висновки

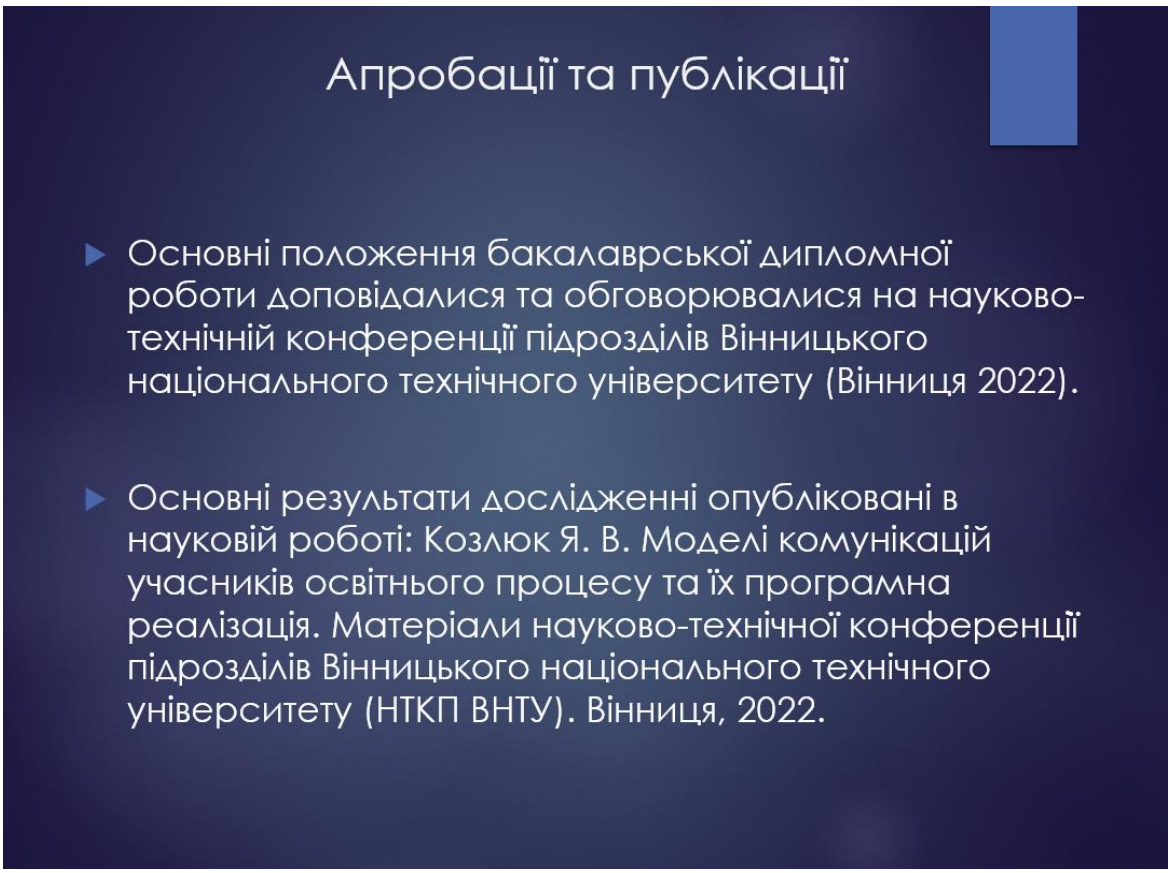

Рисунок Г.17 – Апробації та публікації