ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ДЕЦЕНТРАЛІЗОВАНОГО МЕСЕНЖЕРУ З МЕТОЮ ЗАБЕЗПЕЧЕННЯ ЕФЕКТИВНОГО УПРАВЛІННЯ ПРОЕКТАМИ НА МОВІ C#**»

Виконав: студент 5 курсу, групи ППЗ–51

Спеціальності:

121 Інженерія програмного забезпечення

Коденцев М.І.

Керівник: Довженко Т.П.

Рецензент_____

Київ-2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність - 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ О.В. НЕГОДЕНКО

«____» _____2021 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

КОДЕНЦЕВ МИКИТА ІГОРОВИЧ

1.Тема роботи: «Розробка децентралізованого месенжеру з метою забезпечення ефективного управління проектами на мові C#»

Керівник роботи: Довженко Т.П.

затверджені наказом вищого навчального закладу від «12» березня 2021 року №65.

2. Строк подання студентом роботи «1» червня 2021 року.

3. Вхідні дані до роботи: мова програмування C#, децентралізований програмний продукт, обмін інформацією через сокети TCP-IP, науково-технічна література та наукові статті з питань розробки розподілених програмних продуктів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1. Аналіз особливостей спілкування співробітників, задіяних у галузі управління проектами.

4.2. Обґрунтування проектних рішень при розробці децентралізованого месенджеру для використання в управлінні проектами.

4.3. Реалізація децентралізованого месенджеру для галузі управління проектами.

4.4. Тестування продукту, аналіз результатів роботи, формування висновків.

5. Перелік графічного матеріалу:

5.1. Архітектура системи.

5.2. Алгоритм типового процесу роботи користувача месенджеру у цьому програмному продукті.

5.3. Алгоритм процесу проходження нового повідомлення через систему

5.4. Алгоритм процесу синхронізації клієнта, що підключається до системи.

5.5. Інтерфейс користувача серверної частини

5.6. Інтерфейс користувача клієнтської частини

6. Дата видачі завдання: «19» квітня 2021

КАЛЕДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	19.04.21	Виконано
2	Розробка архітектури додатку	23.04.21	Виконано
3	Проектування алгоритмів роботи програми	27.04.21	Виконано
4	Аналіз особливостей та вибір засобів розробки	2.05.21	Виконано
5	Програмна реалізація програмного продукту	15.05.21	Виконано
6	Тестування програмного продукту	19.05.21	Виконано
7	Розробка обов’язкових демонстраційних матеріалів	21.05.21	Виконано
8	Вступ, висновки, реферат	23.05.21	Виконано
9	Попередній захист роботи	30.05.21	Виконано

Студент_____ Коденцев М.І.

Керівник роботи_____ Довженко Т.П.

РЕФЕРАТ

Текстова частина бакалаврської роботи: 54 ст., 7 рис., 16 джерел.

Ключові слова: управління проектами, месенджер, цифрова комунікація, розподілене програмне забезпечення, децентралізовані системи, клієнт-серверні технології, об’єктно-орієнтоване програмування, передача повідомлень.

Об’єкт дослідження - процес текстового обміну інформацією між особами, задіяними в управлінні проектами.

Предмет дослідження - методи та засоби ведення такого обміну.

Мета роботи - підвищення ефективності спілкування учасників процесу управління проектом за рахунок створення можливості майже миттєвого вирішення виробничих питань невеликої та середньої складності, чого можна досягти шляхом розробки та впровадження децентралізованого програмного комплексу для обміну текстовими повідомленнями.

Методи дослідження – методи галузі програмної інженерії, різноманітні технології програмування, а також загальнонаукові методи аналізу та синтезу.

В результаті дослідження: розроблено архітектуру програми децентралізованого месенджеру, відповідні алгоритми та реалізовано мовою C# готовий до використання програмний продукт, який доцільно впровадити у роботу будь-якої невеликої та середньої компанії та надавати зручні можливості по ефективній комунікації в діяльності по управлінню проектами, а також і інших видах спілкування.

ЗМІСТ

ВСТУП

Успішність бізнесу в сфері ІТ обумовлена багатьма різноманітними факторами, причому одним із найважливіших, безперечно, є здатність компанії вести розробку кожного власного продукту виключно за попередньо розробленим та узгодженим із замовником проектом. При цьому мають в точності виконуватися терміни надання кінцевого продукту, затребувані ресурси, відповідність якісних та кількісних показників кінцевого продукту початково заявленим вимогам (технічному завданню). Очевидно, виконання усіх цих умов забезпечується шляхом застосування методів такої важливої галузі, як управління проектами.

Важливою складовою будь-якого проекту є ефективна комунікація між виконавцями, що у ньому задіяні. При цьому чим більшою є команда виконавців проекту, тим більш важливим стає питання забезпечення ефективної комунікації між ними. Таким чином, актуальною є задача мінімізації величини сумарних витрат часу на комунікацію між виконавцями проектів у сфері ІТ та суміжних галузях, що і буде розглядатися у даній роботі.

Метою даної роботи є підвищення ефективності спілкування учасників процесу управління проектом за рахунок створення можливості майже миттєвого вирішення виробничих питань невеликої та середньої складності, чого можна досягти шляхом розробки та впровадження децентралізованого програмного комплексу для обміну текстовими повідомленнями.

Для досягнення окресленої мети, необхідно здійснити вирішення ряду **задач дослідження**, які є наступними:

- здійснити докладний аналіз предметної галузі та встановити наявність уже готових рішень даної проблеми, виділити їх недоліки (якщо є) та оцінити доцільність проведення власної розробки;

- беручи до уваги недоліки існуючих рішень, результати аналізу предметної галузі та усю необхідну інформацію, слід розробити архітектуру

програмного комплексу та супутню інформацію (алгоритми, описи схем і т.п.), бажано у наочному вигляді;

- на основі розробленої архітектури та супутньої інформації необхідно здійснити реалізацію системи, для чого обґрунтувати вибір технологій та засобів розробки, та впровадити усі розроблені проектні рішення у програмних кодах;

- провести тестування готового програмного продукту, розробити комплект супутньої документації, зробити висновки про можливості подальшого впровадження розробки, її перспективи та варіанти розвитку.

Об'єктом дослідження є процес текстового обміну інформацією між особами, задіяними в управлінні проектами.

Предметом дослідження є методи та засоби ведення такого обміну.

Новизна роботи полягає у розробці відповідної децентралізованої архітектури, застосуванні простого інтерфейсу клієнтської частини при використанні надійних «бек-енд» технологій (наприклад, багатократного дублювання ведення бази даних повідомлень, що підвищує надійність та швидкість роботи та ін.).

Практичне значення роботи полягає у створенні готового для використання програмного продукту – децентралізованого месенджеру, що реально може бути легко впроваджений у роботу будь-якої невеликої та середньої компанії та надавати зручні можливості по ефективному спілкуванню в діяльності по управлінню проектами, а також і інших видах спілкування.

Методи досліджень: методи галузі програмної інженерії, різноманітні технології програмування, а також, загальнонаукові методи аналізу та синтезу.

Перспектива роботи полягає у можливості її розширення у більш повну систему обміну інформацією та, можливо, комерціалізації (адже значна частка таких систем є досить коштовними по своїй ліцензії).

1 АНАЛІЗ ОСОБЛИВОСТЕЙ СПІЛКУВАННЯ СПІВРОБІТНИКІВ, ЗАДІЯНИХ У ГАЛУЗІ УПРАВЛІННЯ ПРОЕКТАМИ

1.1 Види діяльності в управлінні проектами та зв’язок їх ефективності з особливостями процесу спілкування учасників робіт

В першу чергу слід згадати суть поняття «управління проектами» та пов’язаних із ним.

Проект - це захід або процес з чітко визначеними термінами, мета якого полягає у створенні унікального продукту або отриманні якихось інноваційних результатів. У бізнесі метою проекту часто називають рішення конкретних завдань. Відповідно, управління проектами (або, як прийнято це називати, Project Management) - це конкретна діяльність, мета якої - реалізувати всі поставлені проектом завдання. Для цього прикладається максимальна кількість зусиль, знань, досвіду, методик та інструментарію.

Самі по собі проекти - невід'ємна частина реального життя будь-якої організації. У кожній компанії є своя стратегія розвитку і породжувані нею цілі, які і формуються в окремі проекти. Важливо розуміти, чим вони відрізняються від повсякденних дій в організації.

По-перше, у кожного проекту є своя унікальна мета і тимчасові обмеження на її досягнення. У повсякденних діях мета повторюється, дедлайни також.

По-друге, проект має властивість завершуватися, досягнувши заданої мети. Повсякденні ж дії мають нескінченний характер і їх мета - підтримка нормального руху бізнесу.

Управління проектами (або проектний менеджмент) як раз і допомагає швидко і ефективно досягати заданих цілей. Крім того, в процесі цього формується ціла система комплексів, які можуть бути задіяні для загальних цілей компанії, і розробляється схема грамотного розподілу ресурсів.

У складну на перший погляд систему управління проектами входить ряд послідовних дій:

- визначення і формування вимог до проекту;

- постановка максимально чітких і зрозумілих цілей;

- установка і реалізація комунікації між задіяними в проекті сторонами;

- урівноваження проектних обмежень: зокрема бюджету, ресурсів, ризиків, дедлайнів, якості;

- спілкування з командою, облік їх потреб / побажань / очікувань і корекція існуючих планів відповідно до отриманих матеріалів.

Всі ці дії сегментуються на окремі етапи: ініціація проекту, планування, виконання та контроль, завершення. Саме ретельне планування, організація завдань і проектних складових, забезпечення необхідними ресурсами і контроль дієвості обраної стратегії - це і є те головне, що входить в управління проектами з перспективою досягнення поставленої мети. Розберемо вказані етапи докладніше.

Ініціація (тобто старт проекту) є певним знайомством з проектом. Визначається його суть і цілі, формується відповідна під нього команда.

Планування - найважливіша частина в управлінні проектом. Як стверджується в класичній методології по PMBOK, це повинно зайняти приблизно 50% всього часу в процесі реалізації проекту. Складність в тому, що під час цього етапу ретельно прописуються всі дії, які повинна здійснити команда для досягнення заданої мети. Для цього проект спочатку розділяється на частини і набір дрібних завдань. Створюється певний «графік робіт», в якому прописуються дедлайни для кожної з задач. Також опрацьовується список необхідних ресурсів. При цьому планування включає в себе періодичне коригування, адже в процесі роботи постійно з'являються нові нюанси і підзадачі, стають явними якісь «підводні камені» проекту.

Виконання і контроль є етапом, який слід чергувати з попереднім. В ідеальній системі управління проектом все виглядає так: поставили завдання, зробили її, проконтролювали, внесли в план необхідні корективи, поставили

таку задачу і так далі. На етапі виконання зазвичай в хід йдуть якісь інструменти для полегшення перебігу процесів: делегування, тайм-менеджмент, матриця Ейзенхауера і викреслювання справ, та багато інших.

На етапі завершення проекту проводиться контрольна перевірка виконаної роботи і обов'язково зберігаються вихідні дані, задіяні інструкції і регламенти. Це потрібно для того, щоб навіть нова людина в команді змогла легко розібратися, що і як робили до неї.

Існує досить велика кількість методів, як здійснюється управління проектами. Крім класичного, який як раз і описано вище, використовується:

- Agile, коли один великий проект ділиться на багато міні-проектів з поетапною реалізацією;

- Scrum - поділ проекту на складові частини;

- Lean - розподіл проекту на дрібні пакети робіт;

- Kanban - варіант для проектів, що не обмежені по дедлайнам, і їх можна ставити на паузу.

У кожної з методик існують свої нюанси, переваги та недоліки. Вибір підходящої системи залежить від специфіки організації і команди, яка буде працювати над конкретним проектом.

Аналізуючи предметну галузь та розглядаючи усі наведені особливості дій з управління проектами, можна виділити наступні особливості комунікації між їх учасниками, що прямим чином впливають на ефективність всього процесу:

- спілкування має бути швидким і відбуватися без суттєвих затримок, для чого цілком підходить месенджер по пересиланню електронних цифрових повідомлень;

- пересилання повідомлень має бути безпечним, для чого бажано використовувати обмежену кількість сховищ для їх збереження та застосовувати криптографічні прийоми при транспортуванні відкритим каналами зв’язку;

- процес доставки повідомлень має бути надійним, тобто, навіть при відсутності адресата в онлайні, мають працювати механізми доставки повідомлення після того, як він вийде на зв’язок;

- система має бути максимально простою та забезпечувати лише необхідну функціональність, аби не відволікати працівників від виконання дій по роботі над проектом та не заплутувати їх складним інтерфейсом.

Забезпечення вказаних вимог дозволяє проводити розглянуті вище процеси по роботі над проектами максимально ефективним чином і без втрат робочого часу на затримки при комунікації учасників проекту між собою.

1.2 Аналіз можливих варіантів інтенсифікації комунікаційного процесу учасників процесу управління проектами

Задача обміну текстовими повідомленнями, актуальність якої була обґрунтована вище, при першому погляді може здатися простою, але при глибшому аналізі стає очевидним, що існує цілий ряд проблем, які потрібно вирішити розробнику для створення такої ефективної системи.

В першу чергу, визначенню підлягає архітектура або концепція продукту на самому високому рівні абстракції, а саме спосіб взаємодії учасників процесу спілкування.

Традиційним варіантом тут є використання технології «клієнт-сервер», коли на одній стороні працює спільний для усіх клієнтів сервер (часто – це потужний комп’ютер відповідного призначення), а з іншого – різнопланові клієнти, у кожного з яких можуть бути свої особливості, але які вміють спілкуватися із згаданим сервером на спільній мові (по заданому протоколу).

Можливий і інший варіант організації системи для спілкування – однорангова логічна мережа, у якій відсутній виділений сервер, а його функції динамічно розподіляються між активними клієнтами мережі. Такий підхід вимагає значно більш витончених алгоритмічних підходів (зокрема, із галузі кібербезпеки, криптографії, розподілених баз даних та обчислень, і т.п.) і

застосовується в основному для заборонених, незаконних або напівлегальних ресурсів, коли слід позбутися одного центрального сервера, при блокуванні якого «падає» уся система. Однак, і цілком законні ресурси для підвищення загальної надійності процесу спілкування можуть будуватися за описаною схемою. Саме такий варіант буде прийнятий за основу у даному дослідженні відповідно до його теми.

Відмітимо, що незважаючи на в цілому розподілений (децентралізований) характер проектованого рішення, певна частина інформації все ж буде виноситися на сервер, тому тут і далі у тексті розділів періодично мова буде іти про серверні елементи.

Взагалі кажучи, можливі і інші варіанти організації систем текстового зв'язку, але у більшості випадків, розробники не бажають відходити від простої та надійної, перевіреної роками технології «клієнт-сервер».

1.3 Аналітичний огляд існуючих програмних продуктів для спілкування співробітників сучасних компаній

Зрозуміло, що тема обміну текстовими повідомленнями (в т.ч., можливо, із мультимедійними вкладеннями) вже багато разів піднімалася розробниками при створенні нового ПЗ, в результаті чого були створені такі, всесвітньо відомі продукти, як:

- Інтернет-пейджер ICQ, що був дуже поширеним для настільних ПК, але не отримав такого розвитку для мобільних систем. На сьогоднішній день може вважатися застарілим (принаймні в умовах української Інтернет-спільноти) та таким, що поступився місцем більш сучасним (молодим) продуктам;

- практично повний аналог ICQ – російська програма QIP, що у 2000-х та на початку 2010-х рр. була дуже широко поширена на ПК серед української Інтернет-спільноти (в першу чергу, через підтримку інших, крім ICQ, протоколів, наприклад, Jabber, а також масі іншої, додаткової у порівнянні з

ICQ, функціональності), але на даний момент, у зв'язку із бурхливим розвитком мобільних месенджерів ця програма мало використовується навіть і для ПК;

- програма для відозв'язку Skype із самого свого виникнення має функцію передачі виключно текстових повідомлень, однак через маркетингову політику виробника не асоціюється у широких мас користувачів із текстовим обміном, і цю можливість в реальності використовує порівняно мало людей;

- програма-месенджер Viber, яка початково створювалася для смартфонів, але через свою шалену популярність отримала і поширену на сьогоднішній день версію для ПК. Цей продукт ізраїльського походження в умовах України на сьогоднішній день є дійсно лідером по організації в основному текстового обміну, і, в результаті саме його використання, обмін текстовими повідомленнями і став тією процедурою, що сьогодні вже зручно підходить не тільки для спілкування друзів та знайомих між собою, а й керівників високого рівня зі своїми підлеглими (тобто сучасні поняття про субординацію вже давно дозволяють такий обмін), не говорячи вже про спілкування колег по роботі між собою та із кінцевими замовниками (в рамках такого програмного продукту, розробці якого присвячена дана робота);

- майже повний аналог Viber - програмний продукт WhatsApp від компанії Facebook, що раніше був значно популярнішим, особливо у закордонних країнах Заходу, однак після його купівлі гігантом Facebook розвивається значно гірше, ймовірно, не витримуючи внутрішньої конкуренції з Facebook Messenger;

- Facebook Messenger має очевидний і дуже серйозний плюс, який полягає у тотальній інтеграції із обліковим записом Facebook, тобто усі дії, що виконуються у цій, нібито окремій програмі Facebook Messenger, одразу відбиваються у Facebook-акаунті користувача, що є дуже зручним за умови інтенсивного використання цієї соціальної мережі;

- російська розробка Telegram, яка надає набагато більш широкі можливості, ніж простий текстовий обмін, зокрема по організації Telegram-каналів, а також за рахунок використання стійких криптоалгоритмів які нібито не можуть зламати державні спецслужби, на сьогоднішній день перетворилася більше на притулок напівкримінальних елементів, що користуючись цією програмою, активно продають наркотики, зброю, рекламують протизаконні послуги і т.п. Дуже часто назва цього продукту миготить у ЗМІ у кримінальних новинах, а це спричинює доцільність використання інших програмних засобів законослухняними громадянами, які законно ведуть власний бізнес і з маркетингових (іміджевих) причин не хочуть використовувати поширені у зловмисників програмні засоби;

- існують і інші засоби текстового обміну, які, однак, не отримали великої популярності у широкого загалу, тому їх навряд чи доцільно використовувати і для організації спілкування з усіма клієнтами компанії.

Із наведеного переліку можна зробити висновок, що на сьогоднішній день реально в умовах України для робочого спілкування, зокрема і в процесах управління проектами, використовується програмний продукт Viber від Rakuten. Очевидним недоліком такого способу спілкування є іміджеві втрати для компанії, тобто сам такий спосіб спілкування свідчить про мализну підприємства, його кустарність, відсутність власної ІТ-політики розвитку та мале переймання питаннями цифрового маркетингу. Насправді, якщо можливим є створення власного програмного продукту спілкування в рамках управління проектами, причому цей процес не потребує великих не підйомних ресурсів, то доцільність такої розробки є очевидною. Наявність такого власного програмного продукту можна, навіть, використовувати для рекламних цілей.

У даному випадку – при виконанні розробки в рамках написання магістерської дисертації – фінансові ресурси, необхідні для проведення розробки, рівні нулю (організаційно, початковою ціллю розробника є написання якісної роботи та отримання високої оцінки, а не фінансове

заохочення), а отже, створення власної, оригінальної системи спілкування (месенджеру) є безперечно доцільним і актуальним,

Ще одним важливим фактором, що у великій мірі штовхає на розробку власного месенджеру, є практично повна і тотальна залежність від волі власників програми. На власному прикладі багато користувачів стикнулися з проблемою блокування облікового запису Facebook, в результаті чого повністю втрачається переписка, що виконувалася засобами Facebook Messenger (нажаль реальність такої ситуації усвідомлюється тільки після цієї «втрати»). Досить часто цілком законні акаунти можуть бути заблоковані назавжди через невідповідність інформації, що ними публікується, загальній політиці компанії, або через фонове розсилання від їх імені спаму не доброчинними додатками самого сайту Facebook (чи аналогічного ресурсу), або через скарги інших користувачів (від тимчасового аж до повного блокування), тощо. Таким чином, можливість втрати абсолютно усіх переписок, що розміщуються в одній із сторонніх програм є цілком реальною, тому більш надійно використовувати власний програмний продукт, особливо, якщо принципово така можливість існує.

Беручи до уваги наведену інформацію, необхідність розробки вважаємо обґрунтованою, і можна переходити до наступного етапу, яким є деталізація вимог до такого запланованого програмного продукту.

1.4 Виділення невирішених раніше питань проблеми, що розглядається та постановка задачі дослідження

Таким чином, розробці підлягає програмний продукт для обміну текстовими повідомленнями, причому зведений на базі використання сучасних інформаційно-комунікаційних технологій.

Вид продукту – програмний комплекс із двох окремих рішень, що спілкуються по заданому протоколу.

Тип архітектури продукту – однорангова мережа клієнтів, що зберігають переважну більшість інформації на власник ПК, але з присутніми елементами технології «клієнт-серверне» для визначення поточних IP-адрес (які в загальному випадку є динамічними) учасників процесу спілкування.

Цільова платформа сервера – будь-який достатньо потужний комп'ютер, що відповідає загальним вимогам до серверу мережі Інтернет, та працює під управлінням ОС Linux.

Цільова платформа клієнта – будь-який пристрій, що працює під управлінням Windows-сумісної ОС (настільний ПК, ноутбук, планшет, смартфон).

Мови програмування та засоби розробки – сучасні, достатньо популярні, рішення на базі яких можуть в подальшому підтримуватися широким колом програмістів (без необхідності залучення «вузьких» спеціалістів).

Протокол взаємодії – оригінальний, за основу можна взяти один із існуючих протоколів прикладного рівня моделі ISO OSI, за допомогою якого проводити з'єднання клієнтів між собою, а також клієнтів із сервером, та розвинути його або розробити деталі процесу самостійно використовуючи певні методи.

До програмного комплексу висувається вимога дублювання інформації (переписок) одного клієнта на інших, отже на кожному клієнті слід використати СУБД, що підвищує надійність та швидкість обробки даних (у порівнянні, наприклад, із збереженням інформації у звичайному текстовому файлі). Таке сховище даних – зважаючи на особливості задачі (збереженню підлягають прості текстові дані, без застосування складних процедур запису, редагування та збереження) – має бути однією із нескладних СУБД, основною вимогою до якої є «легкість», тобто малі обсяги оперативної пам'яті та ресурсів процесора, що вона споживає. Аналогічну СУБД можна використати і на сервері для збереження списку активних на даний момент користувачів.

Керуючись даними вимогами, можна переходити до наступного етапу процесу створення програмного продукту.

Висновки по розділу

Таким чином, у даному розділі проведено аналіз предметної галузі і встановлено наступні її особливості:

- діяльність по управлінню проектами здійснюється не одним, а, зазвичай, великою кількістю учасників, тому включає їх активну комунікацію між собою;

- ефективність процесів спілкування прямим чином залежить від методів та засобів, які при цьому застосовуються;

- достатньо ефективно комунікацію під час управління проектом можна вести з використанням обміну повідомленнями в електронній цифровій формі (в першу чергу – текстовій).

Встановлено ряд вимог до такої системи комунікації, в першу чергу, надійність обміну повідомленнями та достатній захист інформації, яка у них міститься, чого можна досягти шляхом використання системи децентралізованого месенджеру, тобто такого, в якому повідомлення не зберігаються в одному місці (на сервері), а розосереджені по всім учасникам системи обміну. При цьому злом хакерами одного комп’ютера, навіть такого важливого як сервер, не приводить до розкриття всієї текстової інформації, що супроводжує роботу по управлінню проектом, у чому і полягає суттєва перевага децентралізації роботи месенджеру. Також були встановлені вимоги щодо типу архітектури для платформи сервера та платформи клієнтської частини і протокол взаємодії серверної та клієнтської частини програми.

Тим не менше, встановлено, що серверна частина все ж є необхідною, оскільки, наприклад, зберігати перелік активних користувачів разом із їх поточними IP-адресами, краще на одному центральному сервері, аніж розподіляти між клієнтами, що складно алгоритмічно та менш надійно з організаційної точки зору. Повністю децентралізовані системи, засновані на технологіях типу blockchain, можуть ефективно працювати при кількості

активних учасників, що вимірюється тисячами й більше, а при спілкуванні в управлінні проектами одночасне вимкнення персональних комп’ютерів декількох десятків учасників на ніч є цілком ймовірним і подальший нормальний запуск системи спілкування у такому випадку може стати й неможливим. Отже, ступінь децентралізації рішення має бути високою, але не максимально можливою.

Крім того, у розділі розглянуто існуючі програмні продукти які використовуються різними компаніями, що у більшій, чи меншій мірі відповідають заявленим вимогам (Viber, Telegram, WhatsApp і т.д.). Але, керуючись міркуваннями забезпечення високого іміджу компанії (через наявність власного месенджера), надійності у довготривалому використанні та зберіганні всіх даних (через неможливість власників стороннього месенджера змінити ліцензійні вимоги або взагалі заборонити його використання), а також більшій гнучкості (через доступ до висхідних текстів та внесення усіх необхідних змін та доповнень), можна стверджувати, що доцільною є розробка власного програмного продукту – децентралізованого месенджера для вирішення робочих питань або просто для спілкування всіх учасників процесів управління проектами.

В кінці розділу сформовано технічне завдання на розробку і встановлені конкретні вимоги до відповідного програмного продукту який буде розроблений.

2 ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ ПРИ РОЗРОБЦІ ДЕЦЕНТРАЛІЗОВАНОГО МЕСЕНДЖЕРУ ДЛЯ ВИКОРИСТАННЯ В ГАЛУЗІ УПРАВЛІННЯ ПРОЕКТАМИ

2.1 Розробка архітектури додатку

Задача обміну текстовими повідомленнями, актуальність якої під час роботи над спільними проектами була обґрунтована вище, при першому погляді може здатися простою, але при глибшому розгляді стає очевидним, що існує цілий ряд проблем, які потрібно вирішити розробникам для створення такої ефективної системи.

В першу чергу, визначенню підлягає архітектура або концепція продукту на самому високому рівні абстракції, а саме спосіб взаємодії учасників процесу спілкування.

Традиційним варіантом тут є використання технології «клієнт-сервер», коли на одній стороні працює спільний для усіх клієнтів сервер (часто – це потужний комп’ютер відповідного призначення), а з іншого – різнопланові клієнти, у кожного з яких можуть бути свої особливості, але які вміють спілкуватися із згаданим сервером на спільній мові (по заданому протоколу).

Можливий і інший варіант організації системи для спілкування – однорангова логічна мережа, у якій відсутній виділений сервер, а його функції динамічно розподіляються між активними клієнтами мережі. Такий підхід вимагає значно більш витончених алгоритмічних підходів (зокрема, із галузі кібербезпеки, криптографії, розподілених баз даних та обчислень, і т.п.) і часто застосовується для заборонених, незаконних або напівлегальних ресурсів, коли слід позбутися одного центрального сервера, при блокуванні якого «падає» уся система.

Можливі і інші варіанти організації систем текстового зв’язку, але у випадку, що досліджується, немає жодних причин відходити від простої та надійної, перевіреної роками технології «клієнт-сервер», яку і будемо

використовувати у подальшій роботі. Однак, з іншого боку, існує необхідність позбутися залежності від його центрального сховища, адже при блокуванні цього серверу можуть бути втрачені усі переписки із потенційно цінною інформацією. Відповідно, слід внести певний рівень децентралізації, що також відповідає темі дослідження. У першому наближенні будемо вважати, що повідомлення зберігатимуться на комп'ютерах клієнтів, причому необхідно проводити дублювання інформації, щоби один і той самий текст зберігався мінімум на 2 (а краще і більшій кількості клієнтів). Сервер же будемо використовувати виключно для контролю активних на даний момент користувачів та отримання їх адрес.

Наступним кроком є вибір цільових платформ для сервера та, особливо, для клієнтів.

Оскільки сервер у системі лише один, то питання вибору його програмно-апаратної платформи є більш простим (обравши один конкретний варіант, усю систему можна далі адаптувати під нього та користуватися усіма його особливостями). У якості серверу можна обрати довільне апаратне рішення, яке, наприклад, уже працює як сервер мережі Інтернет (наприклад, комп'ютер, на якому вже завантажений сервер баз даних, або веб-сервер, і т.д.). У якості системного програмного забезпечення серверу слід орієнтуватися на довільний різновид ОС Linux (із популярних – наприклад, Ubuntu, останньої версії 20). На цьому «стандартному» безкоштовному ПЗ сервера можна встановлювати програми для PHP, Python, Perl та ін. серверних мов, а головне, сервер баз даних (конкретний вибір буде обґрунтовано пізніше).

З клієнтами ситуація є більш складною, оскільки одні співробітники використовують мобільний смартфон, інші – планшет, треті – ноутбук, і т.д. Однак, ніхто не буде заперечувати, що будь-яка серйозна робота (створення текстів, схем, написання програм і т.п.) проводиться на персональному комп'ютері (або ноутбуку, що в даному випадку можуть вважатися еквівалентами).

Таким чином, цілком логічною є орієнтація на персональний комп’ютер, як на пристрій, з якого до проектованої системи обміну повідомленнями зможе підключитися майже довільний співробітник (причому саме під час робочого процесу, коли він знаходиться саме за ПК). Також відмітимо, що більше 90 % усіх ПК на сьогоднішній день працюють під управлінням ОС Windows. Отже, саме на таке рішення будемо орієнтуватися як на типового клієнта програмного комплексу, що розробляється.

Таким чином, архітектура системи обміну текстовими повідомленнями, що відповідає умовам задачі дослідження та ринку, є такою, як на рис. 2.1.

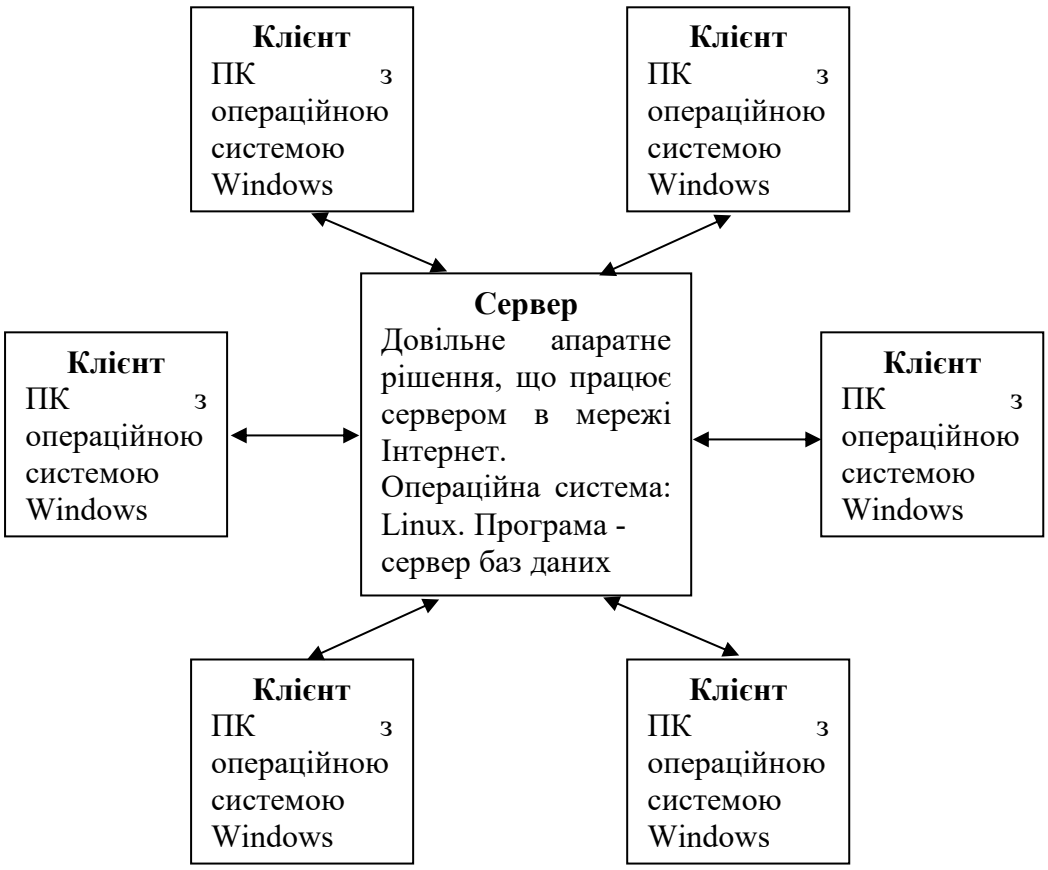


Рис. 2.1 Архітектура проектованої системи на найвищому рівні абстракції

2.2 Обґрунтування вибору засобів розробки

2.2.1 Вибір технології програмування

Першочерговим питанням, яке постає перед розробниками будь-якого програмного забезпечення, є вибір моделі або технології, парадигми його розробки. Такими, що широко використовуються на сьогоднішній день у виробничій практиці, є технології структурного (процедурного) та об’єктно-орієнтованого програмування. Кожна з них має свої особливості, переваги і недоліки, які розглянемо докладніше.

Структурне, або як його ще називають практикуючі програмісти, процедурне програмування засноване на використанні окремих структурних блоків - в першу чергу, підпрограм (процедур і функцій).

Історично перші комп’ютерні програми були відносно простими і мали пакетний режим роботи: отримуючи на вхід якусь інформацію (можливо, навіть на перфокарті) вони виконували певний обсяг операцій з обробки цих даних і видавали результат. Такі програми в основному мали просту лінійну послідовність виконання, тобто в них практично були відсутні будь-які підпрограми, принаймні, використання підпрограм не було наріжним каменем самої методики програмування.

В міру ускладнення функціональності програмного забезпечення, змінювалася і його внутрішня структура, програмний код ускладнювався, і його ставало все більше і більше (у перерахунку на один готовий програмний продукт). Багато для того, щоб людина-програміст без всяких спеціальних хитрувань швидко і легко розібралася з незнайомим кодом (або згадала суть свого власного коду, але написаного тривалий період часу назад). Розміщувати код у простій лінійній послідовності без виділення великих блоків стало незручно, в першу чергу, для розуміння цього коду.

Тут слід зазначити психологічні особливості сприйняття людиною складних «великих» завдань. Неструктуроване «велике» завдання (наприклад, написання дипломної роботи) зазвичай викликає певний психологічний

ступор і, як результат, повну неможливість розібратися з ним. Людині зручно розбити проблему на не надто велику (зазвичай до десятка, а краще 3-4) кількість завдань (наприклад, розділів у дипломній роботі), не замислюючись про реалізацію кожного з них. Коли є ясність і розуміння проблеми на найвищому рівні абстракції, слід приступати до деталізації підзадач, кожну з яких слід розбити на окремі «підпідзадачі» тобто підзадачі нижчого рівня, більш дрібні. Уже після такого розбиття слід аналізувати всі перераховані підзадачі. На певному етапі зупиняються і виконують не розбиття чергової підзадачі на більш дрібні, а безпосередню її реалізацію в програмних кодах.

Говорячи більш строго, структурне програмування має на увазі побудову програми відповідно до трьох основних принципів: слідування, розгалуження, повторення.

Слідування означає, що оператори і блоки програмного коду слідують і виконуються один за іншим. Розгалуження реалізується різними умовними операторами типу if (а також, оператор «?», switch/case і т.п.), і дозволяє вибирати один з декількох подальших варіантів виконання програми. Повторення зазвичай відносять на рахунок циклів (що йдуть підряд багаторазових повторів одного і того ж ділянки коду), хоча цей же принцип можна віднести і до підпрограм.

Взагалі ж, структурне програмування є більш старшою методикою програмування, на зміну якій поступово прийшло об’єктно-орієнтоване програмування (деякі сучасні мови програмування загального призначення навіть не дозволяють створити структурну програму, тільки об’єктно-орієнтовану – як, наприклад, Visual C# чи Java). Проте, при створенні невеликих програм (наприклад, до 10000 рядків коду і без передбачуваного розширення) застосування цієї методики програмування все ще виправдано і такий код краще сприймається, ніж його об’єктно-орієнтований варіант.

Суть же методології об’єктно-орієнтованого програмування полягає в тому, що система розглядається, як сукупність окремих активних сутностей - об’єктів, які мають набір якихось своїх внутрішніх параметрів - властивостей,

а також можуть взаємодіяти між собою за допомогою деяких дій - викликів методів (або більш непрямим чином - шляхом надсилання повідомлень, оброблюваних методами об’єктів; для цього необхідна присутність активної сутності, яка роздає повідомлення адресатам, як, наприклад, менеджер вікон в ОС Windows, який надсилає вікнам програм адресовані до них повідомлення).

Якщо говорити про програмний код, то для того, щоб оперувати об’єктом, його спочатку потрібно створити. Об’єкти створюються як змінні, у яких типом виступає клас об’єкта. Клас являє собою опис, які властивості можуть мати об’єкти такого типу (тобто яку інформацію вони можуть зберігати), і які у них є методи (тобто які дії вони можуть виконувати). Об’єктом же є набір значень, чому саме рівні властивості даного об’єкта (а свої методи кожен об’єкт отримує від свого класу, тобто методи однакові у всіх об’єктів, що належать даному класу).

Для чого потрібен цей специфічний підхід, адже самі по собі об’єкти не додають нічого корисного (навпаки, введення об’єктів ускладнює програму, вносить в неї нові складені сутності)? Виявляється, реалізовуючи всі сутності, необхідні, згідно з алгоритмом, для роботи програми, у вигляді класів і об’єктів, ми спрощуємо її розуміння для самих себе. Саме тому ОО-підхід рекомендується до застосування для великих проектів (більше десятків тисяч рядків коду), коли утримувати «в голові» всю систему цілком стає важко. Можна сказати, що розбиття програми на об’єкти і проектування їх класів наближає розуміння предметної області до звичного людського образу мислення (в разі «великих» проектів). Людина мислить класами, об’єктами і зв’язками між ними.

Відзначимо, що часто, крім розглянутих міркувань, також на вибір методики програмування впливають інші чинники, наприклад, можливість майбутнього розширення функціональності, створення якомога більш зрозумілого коду (для роботи над проектом цілої команди, а не одного програміста), або просто побажання замовника застосувати найбільш сучасний підхід до програмування та ін.

Крім розбиття (декомпозиції) всієї предметної області на об’єкти (класи) і співвідношення між ними, також ОО-підхід має на увазі дотримання трьох основних його принципів: інкапсуляція, наслідування, поліморфізм.

Під інкапсуляцією мається на увазі об’єднання даних (значення властивостей класу у деякого конкретного об’єкта) та засобів їх обробки (методи класу). Це знову ж таки зручно психологічно, так як дозволяє реалізовувати окремі завершені сутності - класи, які самі обробляють свої дані. Звернення до об’єктів цих класів відбувається за допомогою методів, що утворюють інтерфейс класу.

Наслідування є дуже корисною можливістю, тому що дозволяє значно скорочувати обсяги повторюваного коду (до чого потрібно завжди прагнути при розробці будь-якого програмного забезпечення). Згідно з цим принципом виділяється клас, який має спільний набір властивостей і методів для декількох більш розширених класів. Цей клас оголошується батьком, базовим класом для декількох похідних від нього (нащадків, спадкоємців). Всі класи-нащадки успадковують від базового всі його властивості та методи, але до цього ще мають свої власні оригінальні властивості і / або методи.

Наприклад, клас Студент є похідним від класу Людина, тому що кожна людина має властивість Ім’я, Прізвище, метод Відпочити(). Однак у Студента є свої специфічні властивості і / або методи, які як раз і відрізняють його від просто Людини: СереднійБал, НомерЗаліковки, ЗдатиЕкзамен(), і т.д.

При наслідуванні іноді методи батьківського і похідного класу мають однакове призначення, але реалізовуються по-різному. Такі методи називаються перевантаженими. Наприклад, метод Відпочити() у класу Людина реалізується як відпочинок на дивані, а у класу Студент - як похід у нічний клуб. При цьому ще раз підкреслимо, що призначення методу в обох випадках одне і те саме.

Поліморфізм є можливістю деякої функції (яка належить якомусь класу, тобто функції-методу, або окремій, розміщеній поза усіма класами) приймати об’єкти як батьківського, так і похідних класів, і вміти викликати

перевантажені методи саме того класу, об’єкт якого був переданий у функцію. Слід сказати, що це досить специфічна можливість і в загальному багато програмістів використовують ОО-підхід і без звернення до поліморфізму.

Нехай, наприклад, у програмі є функція ПровестиВихідні(), припустимо яка не належить якомусь класу (хоча це і не принципово). Нехай аргументом цієї функції є об’єкт класу Людина. Тоді в неї можна передавати об’єкти всіх похідних від Людини класів: Студент, Службовець, Пенсіонер, і т.д., тому що всі вони є Людиною (спадкоємці цього класу). Ясно, що ця функція повинна включати різні дії: ПрибратиКвартиру(), ПітиНаРинок(), і в тому числі Відпочити(). Так ось поліморфізм дозволяє всередині цієї функції просто вказати назву методу Відпочити(), не вказуючи якого саме класу він повинен бути викликаний, а вже в процесі виконання програми, якщо в функцію переданий об’єкт класу Студент, то викликається саме його метод Відпочити(), а якщо переданий об’єкт класу Пенсіонер, то автоматично викликається саме його метод Відпочити(), і т.д. Кажуть, що функція ПровестиВихідні() - поліморфна, і вона є такою завдяки тому, що реалізує принцип поліморфізму.

Важливими поняттями в ООП також є: статичні члени класу, абстрактні методи і класи, дружба функцій і класів, і т.д.

Грунтуючись на перерахованих особливостях двох найпоширеніших методик програмування, вибираємо об’єктно-орієнтований підхід як більш сучасний, гнучкий та прогресивний, а також такий, що відповідає середнім масштабам проектованого ПЗ.

2.2.2 Вибір мови програмування

При розробці програмного забезпечення після обрання технології програмування слід встановити, які із численних наявних на ринку чи у вільному доступі засобів розробки надають достатні можливості для реалізації

цього алгоритму у програмних кодах. Слід відмітити, що різні засоби розробки, чи навіть цілі мови програмування, з самого початку свого створення мали різне призначення і деякі з них є спеціалізованими, тобто пристосованими до виконання лише певних задач окремих прикладних галузей, а деякі – засобами загального призначення. Серед останніх можна виділити такі, що містять окремі уже готові, більше чи менше пророблені, компоненти для вирішення заданої прикладної задачі, а також такі, в яких таких компонентів немає.

Також важливим фактором при виборі засобів розробки виступає його популярність (зокрема, мови програмування), тобто поширеність фактів застосування цього засобу у професійній діяльності. Не слід недооцінювати цей момент, оскільки в процесі розробки нового програмного забезпечення більш-менш складної функціональності, у будь-якого програміста виникатимуть питання щодо її реалізації, причому якраз за допомогою обраного засобу розробки. У даний час найпершим і найефективнішим місцем, де можна отримати інформацію про питання розробки програмного забезпечення є глобальна мережа Інтернет (на відміну від часів 90-х років, коли найкращим способом вирішення проблеми пошуку відповідей на свої запитання було живе спілкування з експертом, програмістом-старшим товаришем). Сьогоднішні програмісти порівняно легко і швидко знаходять відповіді на практично будь-які питання, пов’язані із професійною діяльністю, в Інтернет (середній час пошуку складає порядку кількох хвилин, максимум – до 10). Така, практично ідеальна, ситуація реалізується тільки за умови поточної популярності мови програмування (або середовища розробки, бібліотеки, фреймворку, і т.п.), що використовується. Наприклад, існують десятки (якщо не сотні) тисяч ресурсів присвячених мові програмування PHP, причому деякі з них мають як докладну теоретичну базу у вигляді пакетів документації, інформаційних статей, відеолекцій, тощо, так і засоби для «живого» спілкування програмістів між собою від форумів до онлайн-чатів. В той же час, зважаючи на те, що всього у світі існує вже 500-2000 (за даними

різних джерел) більш-менш активно використовуваних мов програмування, очевидним є те, що деякі з них використовуються надзвичайно рідко і знайти необхідну інформацію про них у мережі Інтернет іноді не уявляється можливим.

Відмітимо, що на сьогоднішній день достатньо популярними є такі мови програмування загального призначення як: C++, Java, C#, Delphi.

Перший варіант, мова C++, є надзвичайно потужною, яка має стабільну популярність вище середнього рівня, але і досить складною для сприйняття; фактично, це інструмент для професіоналів найвищого рівня. В той же час на сьогоднішній день більш популярними є мови Java та C#. Мова Delphi (або точніше – середовище розробки, оскільки у ньому використовується мова програмування Object Pascal), не зважаючи на свої зручність та простоту (а, можливо, навпаки, через них), стабільно не має широкої популярності серед професійних розробників програмного забезпечення, а використовується в основному для малих проектів та навчальних цілей.

Таким чином, при реальному процесі вибору мови програмування для професійної розробки програмістом середнього рівня на сьогоднішній день слід розглядати два основних варіанти – Java та C#. Дуже багато полеміки було присвячено порівнянню цих двох мов, але беззаперечним фактом є значна перевага продукту від Microsoft над конкурентом у частині зведення користувацького інтерфейсу. У цьому питанні середовище Visual Studio надає практично той же рівень зручності, що й Delphi, але використовує популярні мови, а не Паскаль, до якого традиційно укорінилося ставлення як до навчальної мови програмування для новачків. Отже, зважаючи на необхідність створення програмного засобу для операційної системи Windows, кінцево обираємо мову програмування C#, що і відповідає завданню на розробку в рамках даної роботи. Коротко розглянемо особливості цієї мови програмування.

C# (вимовляється як «сі шарп») - проста, сучасна об'єктно-орієнтована і типобезпечна мова програмування. C# відноситься до широко відомого

сімейства мов C, і буде здаватися добре знайомою кожному, хто працював з C, C++, Java або JavaScript.

C# є об'єктно-орієнтованою мовою, але підтримує також і компонентно-орієнтоване програмування. Розробка сучасних додатків все більше тяжіє до створення програмних компонентів у формі автономних пакетів, що реалізують окремі функціональні можливості. Важлива особливість таких компонентів - це модель програмування на основі властивостей, методів і подій. Кожен компонент має атрибути, які надають декларативні відомості про нього, а також вбудовані елементи документації. C# надає мовні конструкції, які безпосередньо підтримують таку концепцію роботи, і завдяки цьому мова відмінно підходить для створення і застосування програмних компонентів.

Наведемо декілька функцій мови C#, що забезпечують надійність і стійкість програм, розроблених за допомогою неї:

- прибирання сміття автоматично звільняє пам'ять, зайняту знищеними і невикористовуваними об'єктами;

- обробка виключень надає структурований і розширюваний спосіб виявляти і обробляти помилки;

- сувора типізація мови не дозволяє звертатися до неініціалізованих змінних, виходити за межі індексованих масивів або виконувати неконтрольоване приведення типів.

У C# існує єдина система типів: усі типи, включаючи типи-примітиви, такі як int і double, успадковуються від одного кореневого типу object. Таким чином, всі типи використовують загальний набір операцій, і значення будь-якого типу можна зберігати, передавати і обробляти схожим чином. Крім того, C# підтримує визначені користувачем посилальні типи і типи значень, дозволяючи як динамічно виділяти пам'ять для об'єктів, так і зберігати спрощені структури у стеку.

Щоб забезпечити сумісність програм і бібліотек C# при подальшому розвитку, при розробці C# багато уваги було приділено управлінню версіями. Багато мов програмування обходять увагою це питання, і в результаті

програми на цих мовах ламаються частіше, ніж хотілося б, особливо, при виході нових версій залежних бібліотек. Питання управління версіями істотно вплинули на такі аспекти розробки C#, як:

- роздільні модифікатори virtual і override;
- правила вирішення перевантаження методів;
- підтримка явного оголошення членів інтерфейсу.

Для першого знайомства з мовою програмування традиційно використовується програма «Hello, World», і, зважаючи на її простоту й компактність, можемо розглянути тут її приклад на C#:

```
using System;
class Hello
{
    static void Main ()
    {
        Console.WriteLine ( "Hello, World");
    }
}
```

Файли висхідного коду C# зазвичай мають розширення .cs. Якщо код програми «Hello, World» зберегти в файлі hello.cs, то для її компіляції можна виконати таку команду з командного рядка:

```
csc hello.cs
```

В результаті буде отримана виконувана збірка з ім'ям hello.exe. Ця програма при запуску виводить наступний результат:

```
Hello, World
```

Команда csc фактично і є компілятором мови C# (csc.exe), тобто виконує компіляцію на повній версії платформи, і вона може існувати не для усіх програмно-апаратних систем (початково призначалася для програмування для ОС Windows).

Коротко проаналізуємо особливості наведеної програми «Hello, World». Вона починається з директиви using, яка посилається на простір імен System. Простори імен дозволяють ієрархічно впорядковувати програми і бібліотеки C#; вони можуть містити типи і інші простори імен. Наприклад, простір імен System містить кілька типів (в тому числі, який використовується в нашій програмі клас Console) і кілька інших просторів імен, таких як IO і Collections. Директива using, яка посилається на простір імен, дозволяє використовувати типи з цього простору імен без вказівки повного імені. Дякуючи директиві using в коді програми можна використовувати скорочене ім'я Console.WriteLine замість повного варіанту System.Console.WriteLine.

Клас Hello, оголошений в програмі «Hello, World», має тільки один член - це метод з ім'ям Main. Метод Main оголошений з модифікатором static. Методи об’єкта можуть посилатися на конкретний його екземпляр, використовуючи ключове слово this, а статичні методи працюють без посилання на конкретний об'єкт. За стандартною угодою, точкою входу програми на C# є статичний метод з ім'ям Main (аналогічно, як раніше, при структурному програмуванні точкою входу була функція main, а потім – WinMain).

Вихідні дані програми створюються в методі WriteLine класу Console з простору імен System. Цей клас надається бібліотеками стандартних класів, посилання на які компілятор за замовчуванням додає автоматично.

Як видно із наведеного прикладу, програмування мовою C# є досить простим та зручним. В цілому, можна констатувати, що мова програмування C# на сьогоднішній день є найкращою для програмування сучасних додатків загального призначення, тому саме її обрано у якості основного засобу для реалізації програмного засобу, що розробляється.

Визначившись із мовою програмування, слід провести вибір засобів розробки, що залишилися (конкретний компілятор та компонувальник, текстовий редактор, система допомоги, наладчик і т.д.). Найкраще всього

використовувати для цього якесь інтегроване середовище розробки (Integrated Development Environment, або IDE).

2.2.3 Вибір середовища розробки та допоміжних засобів

Зважаючи на те, що обрана мова програмування C# з'явилася уже майже 20 років тому назад (у 2002 році), для неї за цей час з'явилися різноманітні альтернативні компілятори та середовища розробки потреба у яких сильно зменшилася, коли Microsoft почала на постійній основі (а не епізодично, під якусь тимчасову акцію) видавати безкоштовні версії власного продукту Microsoft Visual Studio.

Цей інструмент беззаперечно є найпотужнішою платформою для програмування додатків, який пропонується корпорацією Microsoft, що уже більше 20 років розвиває свій продукт Microsoft Visual Studio (далі – MVS). За цей час він від простого середовища розробки із трьома мовами (C++, Java та Visual Basic) перетворився на супергіганта, повна інсталяція якого у 2019 році займає більше 200 Гб (!!!) і підтримує десятки різних мов програмування. Слід відмітити, що серцем середовища MVS є .NET Framework – програмна платформа, випущена компанією Microsoft в 2002 році. Основою платформи є загальномовне середовище виконання Common Language Runtime (CLR), яке підходить для різних мов програмування. Функціональні можливості CLR доступні в будь-яких мовах програмування, що використовують це середовище, однак основною мовою із самого початку тут позиціонувалася саме C#. Вважається, що платформа .NET Framework стала відповіддю компанії Microsoft на платформу Java компанії Sun Microsystems (нині належить Oracle), що набрала на той час дуже велику популярність. Дійсно, докладний аналіз показує, що Microsoft використовує дуже схожі по своїй суті технології, але називаючи їх по-іншому.

Величезною перевагою продукту від Microsoft стала на порядок більша зручність використання середовища MVS, у порівнянні з продуктами для Java-розробки. Наприклад, середовище Eclipse потребує численних налаштувань

для кожного більш-менш специфічного проекту. Очевидно, що при наявності досвіду використання у декілька років усі ці дії виконуються за кілька секунд, однак сьогодні часто зустрічається ситуація (особливо у невеликих компаніях), коли один програміст у різні періоди використовує різні інструменти, причому значну їх кількість. Відповідно, не лише для новачків, а й для т.зв. «багатоверстатників» актуальним є питання якомога меншої кількості налаштувань, інсталяцій додаткового програмного забезпечення (бібліотек, модулів, і т.п.), низького порогу входження – по всім цим показникам Microsoft Visual Studio значно випереджає своїх конкурентів, зокрема продукти для розробки на платформі Java.

І ще одним немаловажним фактором, який також діє у плюс продукту від Microsoft, є вимоги до швидкості роботи у даному середовищі розробки при заданому обсязі оперативної пам’яті та характеристиках процесора. MVS, наприклад, виконує такі базові речі, як компіляція проекту в рази швидше (і за цілком прийнятний час порядку секунд чи десятків секунд для проектів середньої складності), у порівнянні із дійсно зручними середовищами розробки для платформи Java на зразок IntelliJ IDEA (компіляція проекту у якій може займати час порядку хвилин і навіть їх десятків).

Суттєвою перевагою Java спочатку була відкритість засобів розробки, але на сьогоднішній день існують повністю вільні для використання (навіть при створенні комерційних програм) версії MVS, тому відповідно у цьому показникові дві даних платформи є рівними. Також хоча .NET є патентованою технологією корпорації Microsoft і офіційно розрахована на роботу під операційними системами сімейства Microsoft Windows, існують незалежні проекти (перш за все це Mono і Portable.NET), що дозволяють запускати програми .NET на деяких інших операційних системах. В даний час .NET Framework отримує розвиток у вигляді .NET Core, яка з самого початку передбачає кросплатформенну розробку і експлуатацію. Таким чином, і за показником кросплатформенності дві цих платформи стали приблизно рівними у останні роки.

Беручи до уваги усі особливості (зручності) середовища MVS, обираємо його за основу для подальшої розробки програмного модуля, що проектується.

2.3 Проектування алгоритмічної складової децентралізованого месенджеру

Після обрання засобів розробки можна переходити до розробки алгоритмічної складової проекту. В першу чергу, розглянемо алгоритм роботи користувача із програмою, блок-схема якого показана на рис. 2.2. Аналіз наведеної схеми свідчить, що сам спосіб роботи користувача із системою з є більш-менш стандартним, а особливості криються у механізмах реалізації деяких із наведених пунктів, зокрема, «Провести синхронізацію повідомлень» та «Надіслати нове вихідне повідомлення». Так, при проведенні цих процесів слід враховувати розподілений характер всього програмного продукту.

Розпочнемо аналіз вказаних двох процесів зі створення нового повідомлення. При цьому, як було зазначено вище, текст цього повідомлення має зберігатися на клієнтах, але не на сервері, в чому і полягає суть децентралізації проектованого рішення. Відповідно, алгоритм роботи програми при надсиланні нового повідомлення від користувача можна представити схемою рис. 2.3.

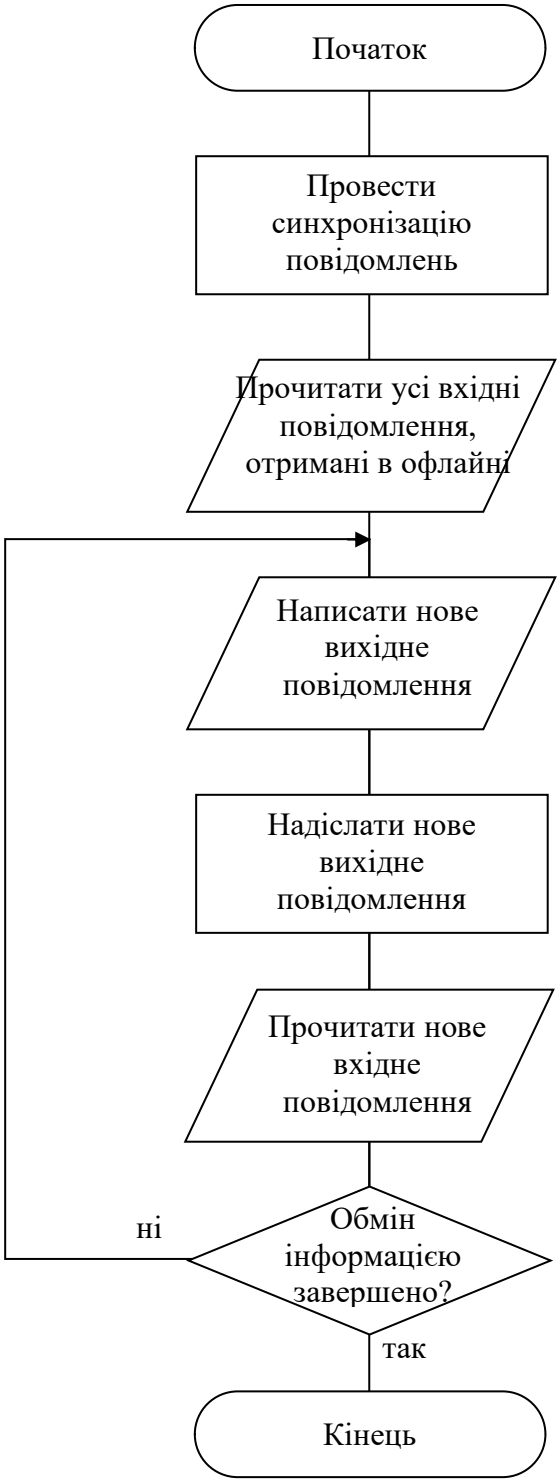


Рис. 2.2 Блок-схема алгоритму типового процесу роботи користувача месенджеру у цьому програмному продукті

На рис.2.3 користувач, що генерує нове повідомлення називається відправником, а той, що його отримує, адресатом. Алгоритм враховує, що адресат під час відправки може знаходитися оффлайн і тоді повідомлення тимчасово зберігатиметься на сервері, але після відправки – видаляється.

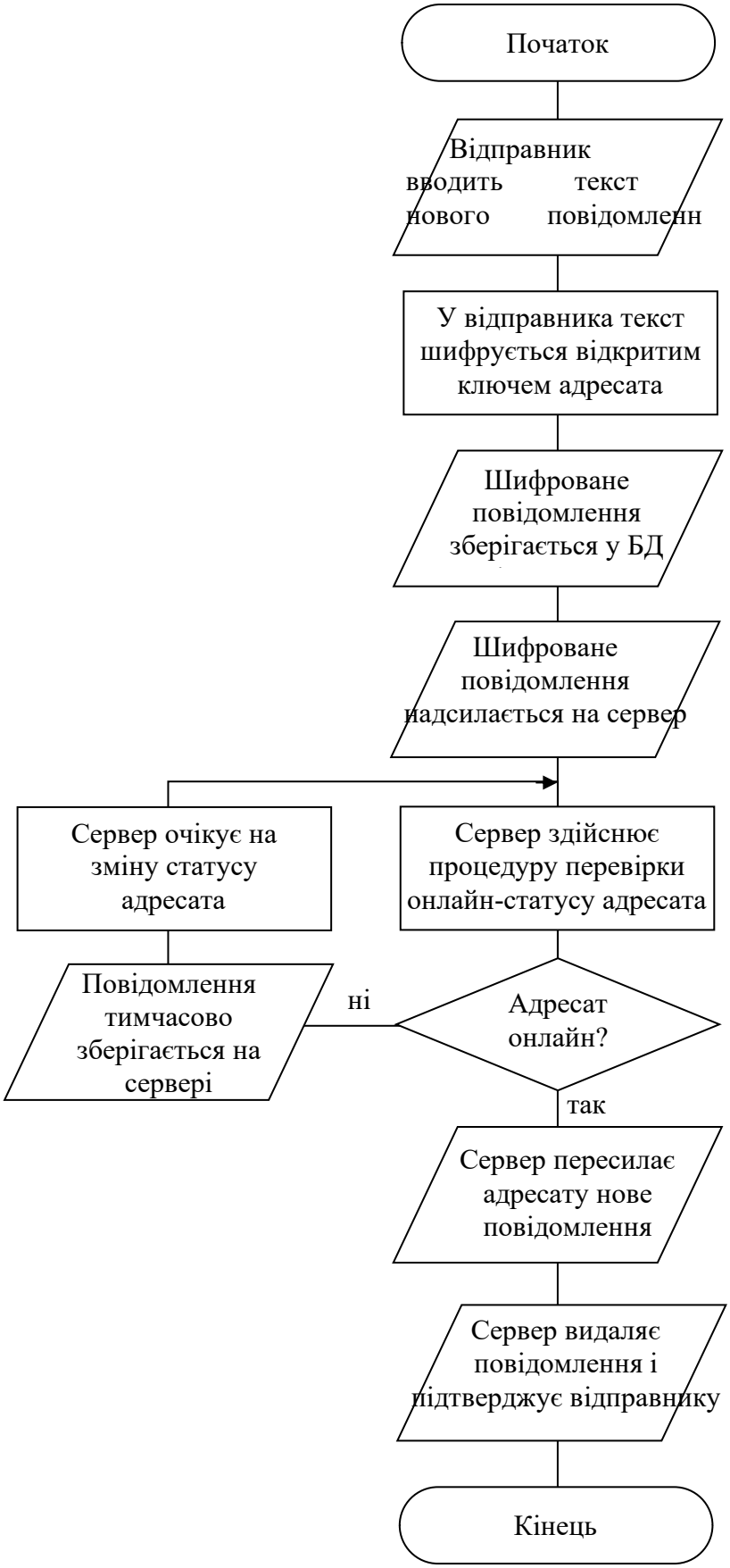


Рис. 2.3 Блок-схема алгоритму процесу проходження нового повідомлення через систему

Важливим з алгоритмічної точки зору процесом є також проведення синхронізації поточного стану клієнтської частини месенджеру при її включенні користувачем. Схема відповідного алгоритму наведена на рис. 2.4.

Рис. 2.4 Блок-схема алгоритму процесу синхронізації клієнта, що підключається до системи

Керуючись наведеними алгоритмами, можна переходити до процесу виконання програмної реалізації децентралізованого месенжеру для ефективного проведення управління проектами.

Висновки по розділу

Таким чином, аналізуючи все вище перераховане, у даному розділі проведено обґрунтування проектних рішень при розробці децентралізованого месенджеру для забезпечення використання в галузі управління проектами.

В першу чергу, розроблено архітектуру та концепцію рішення, в якому виділено серверну частину, але її призначення, відповідно до теми роботи (яка включає слово «децентралізований»), полягає виключно у веденні обліку бази даних всіх зареєстрованих користувачів, а також переліку поточних активних користувачів разом із їх IP-адресами. У якості серверу нами була обрана досить популярна операційна система Linux на якій і буде працювати наша серверна частина продукту. Для клієнтської частини ми орієнтувались на використання персонального комп’ютера під операційною системою Windows, так як більше 90 % усіх ПК працюють під ОС Windows. Клієнтські частини отримуючи IP-адресу бажаного адресата від сервера, після цього переходять до прямого спілкування між собою, реалізуючи концепцію децентралізації.

Далі обрано технології та засоби розробки. По-перше, було проаналізовано особливості найбільш поширених на сьогодні технологій структурного та об’єктно-орієнтованого програмування та встановлено, що у даному випадку останній підхід є найбільш ефективним і прийнятий за основу в даній роботі.

Серед дуже великої кількості мов програмування загального призначення, що підтримують об’єктно-орієнтовану розробку на сьогодні досить популярною є мова C#, яка і була обрана у якості основного засобу для

виконання програмної реалізації серверної та клієнтської частини у даному дослідженні.

Обрана мова програмування може передбачати використання декількох різних засобів розробки (починаючи від консольного використання компілятора csc.exe), однак найбільш ефективною є робота із «рідним» інтегрованим середовищем розробки та певну кількість інших інструментів Microsoft Visual Studio 2019. У роботі використано вільну для використання версію Community.

Також у роботі проведено розробку алгоритмів, відповідно до яких функціонуватиме програмне забезпечення, що розробляється: створено схему типового процесу роботи користувача месенджеру у цьому програмному продукті з механізмами для «Проведення синхронізації», «Створенні та надсиланні нових повідомлень». Схему алгоритму процесу проходження нового повідомлення через систему в якій обов’язково повинен використовуватись механізм шифрування, збереження та можливість надсилання повідомлень адресатам, які можуть знаходитись оффлайн, а також схема алгоритму процесу синхронізації клієнта, що підключається до системи з збереженням відповідних даних клієнта який підключився та надсиланні шифрованих повідомлень, які були йому адресовані під час перебування в оффлайн.

Вказані проектні рішення є основою для проведення програмної реалізації, яка описується у наступному розділі.

3 РЕАЛІЗАЦІЯ ДЕЦЕНТРАЛІЗОВАНОГО МЕСЕНДЖЕРУ ДЛЯ ГАЛУЗІ УПРАВЛІННЯ ПРОЕКТАМИ

3.1 Розробка інтерфейсу користувача програмного продукту

Як відомо, одним із перших кроків при створенні нового програмного забезпечення є проектування його інтерфейсу користувача. У системі, що розробляється, наявні 2 частини: серверна і клієнтська, тому розглянемо їх інтерфейс окремо.

3.1.1 Інтерфейс серверної частини

Серверну частину реалізуємо так, щоб вона відображувала поточний перелік активних користувачів, тобто таких, у яких на даний момент часу є відкритим вікно клієнтської частини. За необхідністю можна відображувати перелік усіх користувачів, обираючи відповідну радіоконпку – рис. 3.1.

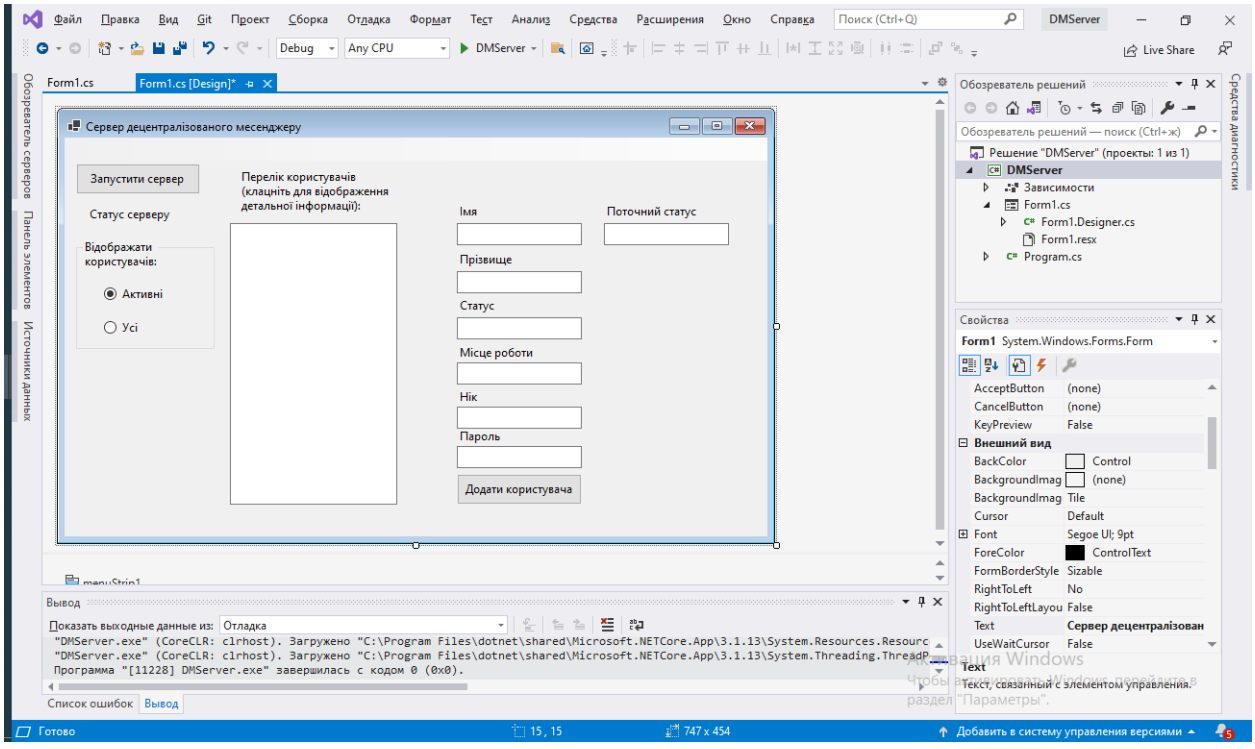


Рис. 3.1 Вікно середовища розробки Visual Studio під час проектування інтерфейсу користувача серверної частини месенджеру, що розробляється

3.1.2 Інтерфейс клієнтської частини

Клієнтську частину зводимо у більш звичайному для месенджерів вертикальному вигляді – рис. 3.2.

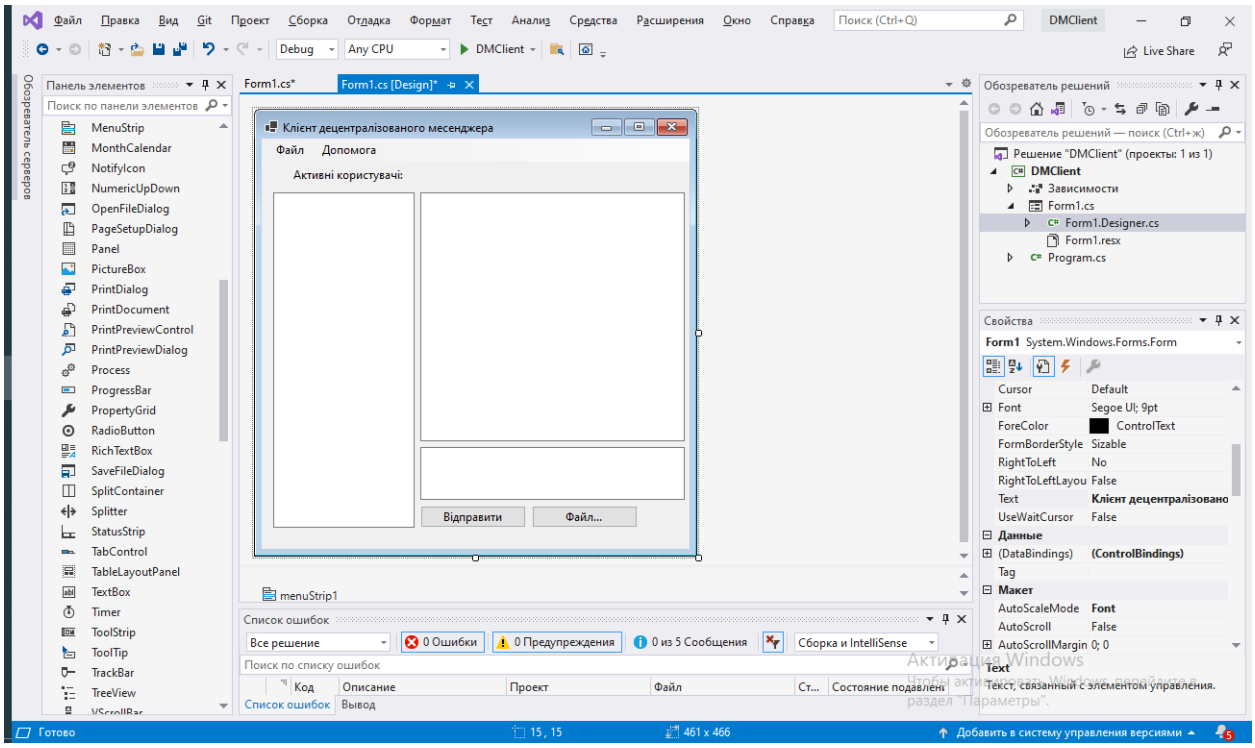


Рис. 3.2 Вікно середовища розробки Visual Studio під час проектування інтерфейсу користувача клієнтської частини месенджеру, що розробляється

3.2 Програмна реалізація алгоритмів роботи децентралізованого месенджера

3.2.1 Особливості реалізації серверної частини

В першу чергу, слід визначити спосіб організації зв’язку між сервером та клієнтами, що реалізується у даній роботі.

Зв’язок реалізується за допомогою механізму сокетів TCP-IP. При цьому серверна частина має весь час прослуховувати певний порт, а при надходженні запиту на спілкування від нового клієнту, завантажувати новий потік, який займатиметься обробкою запитів від цього клієнту. У мові C# для цього

впроваджено дуже зручний механізм, який не потребує створення нового потоку (наприклад, за допомогою функції CreateThread() або аналогічних). Для цього достатньо вказати біля імені такої функції, що завантажуватиметься в окремому потоці слово Async, як наприклад:

```
private async Task ListenAsync()
```

При такому завданні у тілі даної функції можна створити вічний цикл (або майже вічний – допоки не зміниться змінна run, значення якої можна міняти із зовнішнього по відношенню до даної функції коду) виду:

```
while (run)
  {
  ...
  }
```

У тілі даного циклу постійно намагаємося читати вхідні повідомлення від клієнтів, що підключаються, і при успіху, передаємо їм у відповідь статусні та інформаційні повідомлення. Власне, підключення нових клієнтів проводиться за допомогою змінних client і stream спеціальних типів:

```
TcpClient client = await server.AcceptTcpClientAsync();
NetworkStream stream = client.GetStream();
```

Відмітимо, що читання із створеного потоку stream здійснюється командою

```
stream.Read(myReadBuffer, 0, myReadBuffer.Length);
```

Запис відповіді у цей потік stream здійснюється за допомогою коду:

```
Byte[] responseData = Encoding.UTF8.GetBytes(textBox1.Text);
stream.Write(responseData, 0, responseData.Length);
```

3.2.2 Особливості реалізації клієнтської частини

Клієнтська частина використовує схожі принципи прийому та надсилання інформації, що й серверна, за тим виключенням, що в неї немає циклу очкування вхідних повідомлень, а вона здійснює циклічні підключення до серверу, тим самим отримуючи нові повідомлення, а також і оновлюючи свій статус у серверній базі даних – точно у відповідності до рис. 2.3.

3.3 Тестування та оцінка ефективності створеного програмного продукту

Після створення і установки всіх частин розробленої системи її робота повинна бути протестована. Існують різні методики і варіанти тестування, особливості яких розглянемо докладніше.

Тестування програмного забезпечення (далі – ПЗ) є важливим невід'ємним етапом його розробки, який в окремих випадках може тривати періоди часу, порівнянні з часом кодування усіх вихідних текстів проекту. Саме поняття тестування можна трактувати у досить широких межах, причому ще і описувати можливі його варіанти за різними ознаками. Уся багатоплановість цього процесу спричинює необхідність детального аналізу та виділення окремих видів тестування ПЗ. В першу чергу, наведемо ознаки, за якими можна проводити класифікацію видів тестування – рис. 3.3.

Рис. 3.3 Ознаки, за якими може проводитися класифікація видів тестування програмного забезпечення

Розглянемо докладніше варіанти поділу усіх видів тестування на класи при використанні різних ознак з рис. 3.1.

Найбільш обширним є перший варіант класифікації, коли різними можуть бути цілі самого тестування. Очевидно, найпершим по важливості у переліку цілей є належне виконання програмою задач, покладених на неї розробником, іншими словами – належне виконання власних функцій. Таке тестування, відповідно, називають функціональним. Наступними цілями, що відносять процес тестування до нефункціонального класу, є:

а) тестування продуктивності, адже одна програма може виконувати певну задану обробку вхідних даних протягом хвилини, а інша – за десять хвилин. В той же час перша програма при цьому може затребувати для своєї роботи 8 Гб оперативної пам’яті, а друга – всього лише 1 Гб. Очевидно, що продуктивність сильно залежить від алгоритмів, які були застосовані розробниками при реалізації програмного продукту, в т.ч. і другорядних, тобто таких, що не були окреслені у технічному завданні (ТЗ) на розробку програми. Саме на цьому етапі також уточнюються системні вимоги програми, тобто мінімальні характеристики ПК, на якому ще буде забезпечуватися виконання програмою своєї функціональності при адекватних часах обробки даних. Зазвичай, мінімальні системні вимоги прописуються у ТЗ, але вони можуть бути переузгоджені після проведення тестування продуктивності, причому як у меншу, так і у більшу сторони.

б) тестування інтерфейсу користувача (User Interface, UI), при якому оцінюється виконання розробниками стандартних (загальноприйнятих) правил створення графічних інтерфейсів користувача, або у випадку застосування незвичайних (інноваційних) елементів – комфортність їх використання «середнім» користувачем. Тут оцінюються кольорові схеми, розміри елементів управління, їх взаємне розташування, та інші аспекти зовнішнього вигляду програми.

в) тестування доступності функцій ПЗ та зручності користування ними для «середнього» користувача, яке включає перевірку інтуїтивності та зрозумілості елементів інтерфейсу програми, а при використанні незвичайних рішень – їх адекватність функціональності самого ПЗ (наприклад, для дитячої пізнавальної програми використання елементу управління «рожева повітряна куля, що стрибає» є надзвичайно ефективним, а для корпоративного ПЗ – недопустимим, і т.п.).

г) тестування захищеності ПЗ, тобто його здатність протидіяти як випадковим, так і зловмисним діям користувача, що можуть викликати збої в роботі ПЗ. В першу чергу, тут мають тестуватися усі введення, що здійснює користувач, особливо на традиційну помилку переповнення буферу (SQL-ін’єкцію, якщо використовується база даних, і т.п.).

д) тестування процесу встановлення програми, на якому слід упевнитися, що при усіх можливих варіантах опцій, виставлених користувачем, а також допустимих конфігураціях апаратного забезпечення ПК процес інсталяції проводиться адекватно і, не викликаючи помилок, приводить до бажаного результату.

е) тестування сумісності, яке в першу чергу стосується різних версій цільової операційної системи (наприклад, Windows 7, 8, 10), а також бібліотек, драйверів та іншого стороннього програмного забезпечення, яке використовується програмою.

є) тестування надійності, яка, як і у випадку технічних систем, описує роботу програми на тривалих інтервалах часу (в першу чергу, слід встановити, чи не накопичуються помилки при довгому використанні програми).

ж) тестування локалізації, направлене на перевірку якості та адекватності перекладу усього тексту, вбудованого у програму. Очевидно, має виконуватися людиною, що добре знає цільову іноземну мову, а краще – її носієм.

Далі можна повернутися до переліку ознак, висвітлених на рис. 3.3, і наступною ознакою є ступінь автоматизації при проведенні тестування.

Очевидно, що робота по тестуванню може бути повністю ручною, мануальною, коли усі дії по перевірці програми виконує людина. В той же час, не має фізичної можливості людині перевірити, наприклад, тисячі варіантів вводу користувача у певне числове поле (а таких полів, зазвичай, у програмі є досить багато!), але за допомогою засобів автоматизації процесу тестування така перевірка стає можливою: необхідно лише, щоби завдання цього числа виконувалося спеціально написаною програмою, яка би і перевіряла отриманий після цього результат. Або задавати вхідні дані може одне програма тестування, а перевіряти, наприклад, файли-результати може інша програма.

Третьою ознакою для класифікації є орієнтування на нормальні, чи незвичайні (екстремальні) умови роботи з програмою. Позитивною називають перевірку, направлену на оцінку дій програми у стандартних умовах її використання. Негативним є тестування програми у нестандартних умовах, коли поведінка користувача, або інших впливових зовнішніх факторів відрізняється від звичайних їх значень.

За рівнем доступу до коду ПЗ тестування може розділятися на наступні типи:

- по схемі чорного ящика, коли тестувальник взагалі немає доступу до коду ПЗ, а працює лише задаючи різні вхідні дані, та оцінюючи результат;

- по схемі білого ящика, коли людина-тестувальник має повний доступ до коду;

- по схемі сірого ящика, коли доступу до коду немає, але тестувальник добре представляє структуру програми, склад її модулів, класів, функцій чи менших блоків коду (без надання самого вихідного тексту).

За рівнем блоку коду, що підлягає тестуванню, виділяють наступні варіанти:

- модульне або юніт-тестування, при якому перевірці підлягає один блок, як, наприклад, клас, чи функція, а іноді – окремий модуль програми;

- інтеграційне тестування дозволяє перевірити взаємодію окремих модулів, адже навіть при їх ідеальному відлагодженні, можлива наявність неузгодженості між даними, що передаються, і в результаті програма працюватиме невірно;

- системне тестування направлене на перевірку роботи всієї програми в цілому;

- приймальне тестування передбачає перевірку того, чи відповідає розроблена програма усім пунктам наявного ТЗ.

Різні виконавці тестування визначають його розділ на альфа та бета тестування. Альфа-тестування виконують самі розробники і воно направлене на виключення усіх очевидних помилок, які змогли знайти власне програмісти. Після його завершення розробники вважають, що програмний продукт уже завершений і передають його іншим особам – бета-тестувальникам, які можуть утворювати певну фокус-групу, або бути окремими віддаленими добровольцями, і т.п.

Також тестування може розділятися за формальністю цього процесу, що, взагалі кажучи, визначається ступенем підготовки тестувальника до самого процесу. Тут можна виділити:

- проведення роботи по тестам, що заранні розроблені (можливо, одразу після формування ТЗ, або пізніше) та утворюють певний банк тест-кейсів;

- дослідницьке тестування, що характеризується одночасною розробкою тестів та їх негайним використанням;

- вільне тестування, що проводиться без розробки спеціальних тестів, а виключно у режимі користувача, який просто використовує ПЗ для своїх різноманітних цілей (якість цього варіанту сильно залежить від досвіду тестувальника, та, взагалі кажучи, її неможливо перевірити, тому користуватися цим типом тестування слід із обережністю, виключно із повністю перевіреними тестувальниками).

Нарешті, тестування також може розділятися на класи по важливості тієї функціональності, що підлягає перевірці під час даного процесу:

- димове тестування, коли оцінюється лише виконання самих важливих складових функціональності;

- тестування критичного шляху перевіряє типові послідовності дій, які виконують у повсякденній роботі з програмою звичайні її користувачі;

- розширене тестування – повна і найбільш глибока перевірка усієї функціональності програми.

Як зазначено при аналізі ознаки «2.Ступінь автоматизації» досить важливим варіантом тестування є автоматизований. При цьому виникає можливість здійснити набагато більше перевірок комбінацій вхідних даних, аніж при ручній праці. Однак, при цьому виникає ряд труднощів наступного характеру.

В першу чергу, існує така принципова проблема, що автоматизоване тестування виконується за допомогою спеціально розроблених програм, які часто пишуться самим тестувальником задля того, щоби урахувати специфіку того ПЗ, яке розглядається. Очевидно, при цьому необхідно бути впевненим, що самі програми для тестування не мають помилок, про що можна із впевненістю говорити у випадку їхньої крайньої простоти, але є сумнівним у випадку використання складних алгоритмів.

По-друге, сучасні програми мають багато різних варіантів використання, але для кожного з них завжди можна виділити якісь вхідні дані, що надходять із зовнішніх по відношенню до програми, що тестується, джерел: від користувача, по локальній мережі, із Інтернет, і т.п. Проблема полягає у адекватному виділенні тих дійсно важливих вхідних даних, що підлягають автоматизованому завданню, та відсіканні таких, які задавати автоматизовано не доцільно (і, можливо, які взагалі не повинні підлягати перевірці). Частинним випадком цієї проблеми є виділення адекватних діапазонів, що підлягають задаванню під час тестування, для таких вхідних величин, що обрані, як важливі.

Нарешті, ще однією проблемою автоматизованого тестування може бути складність оцінки кінцевого результату, який часто необов’язково

представляється у зручній числовій формі. Навпаки, результатом може бути оброблене зображення, набір різнорідних даних (масив або вектор), текстовий рядок (або навіть великий текст) і т.п. Для оцінки деяких із таких результатів необхідно залучати методи галузі штучного інтелекту, що дозволяють проводити такі оцінки, однак складність алгоритмів, що мають бути при цьому застосовані, може бути порівнянною до складності самого ПЗ, яке підлягає тестуванню. При цьому має підлягати ретельному розгляду питання про доцільність автоматизованої перевірки в цілому, адже у деяких типах слабко формалізованих задач відповідні алгоритми можуть бути навіть не розробленими.

Загадаємо також, що іноді озвучуються і цілком непередбачувані проблеми автоматизованого тестування, наприклад, така, як його велика трудомісткість. Здається, що уся суть автоматизації будь-якого процесу направлена на зменшення трудомісткості певних операцій ручної праці, які, власне, і підлягають автоматизації. Але у даному випадку (автоматизація саме процедури тестування програмного забезпечення), мова йде про те, що у великій кількості випадків написати якісну програму-тест стає за трудомісткістю не набагато легше, аніж провести ручне тестування продукту (тим більше, немає повної, стовідсоткової впевненості, що сама програма-тест працює у точності відповідно до свого призначення).

Усі озвучені труднощі призводять до того, що деякі компанії взагалі відмовляються від автоматизованого тестування, віддаючи перевагу використанню дешевої праці найнятих віддалених тестувальників із країн, що розвиваються.

Розглянувши дуже докладно особливості сучасних видів тестування програмних продуктів, перейдемо до опису процесу тестування саме децентралізованого месенджеру для забезпечення ефективного управління проектами. Зважаючи на наведену інформацію, було проведено бета-тестування розробленого програмного продукту системного характеру, тобто всього програмного продукту (як комплексу) в цілому. Нагадаємо, що система

була реалізована у вигляді розподіленого програмного продукту із двох частин (серверної та клієнтської), і його робота була піддана тестуванню, яке показало наступні результати:

- робота програмного забезпечення проходить стабільно, без виникнення системних помилок, аварійних завершень програми і інших позаштатних ситуацій (причому як серверної, так і клієнтської частин);

- продуктивність створеного програмного забезпечення знаходиться на достатньому рівні: зависання, зупинки або помітні паузи в роботі програми відсутні при проведенні довільних санкціонованих операцій. Максимальна затримка при пересиланні повідомлень є помітною для людини, але складає не більше величин порядку 1 секунди;

- в цілому програмний продукт адекватно виконує поставлене перед ним завдання утворення ефективного комунікаційного середовища для спілкування осіб, задіяних в процесі управління проектом;

- працювати з продуктом зручно, інформація добре сприймається і процес комунікації відбувається достатньо ефективно.

3.4 Розробка супроводжувальної документації

Програмний продукт, що розроблявся, являє собою програмний комплекс, в якому присутній один сервер та мінімум два клієнти. Отже, існує певна послідовність у встановленні його компонентів, які приводимо у даному пункті у вигляді міні-інструкції адміністратора по встановленню та розгортанню системи:

1) виділити комп’ютер-сервер, що має постійний доступ до Інтернет зі статичною IP-адресою і працює під управлінням ОС Windows, версії не нижче Windows 7;

2) скопіювати у робочий каталог програми файл server.exe, що являє собою серверну частину розробленого програмного комплексу;

3) перевірити роботу сервера, завантаживши його локально;

4) за допомогою програмного забезпечення серверу створити необхідну кількість облікових записів для користувачів системи;

5) встановити розроблену раніше клієнтську частину комплексу на декілька персональних комп'ютерів, що іноді мають доступ до мережі Інтернет;

6) у меню «Файл – Налаштування» слід ввести IP-адресу серверу, яка є постійною;

7) передивитися перелік активних користувачів і відправити одному з них декілька тестових повідомлень, отримавши тестове повідомлення у відповідь;

8) перевіривши роботу комплексу, завершити інсталяцію та розпочати робоче використання програмного продукту.

Для користування системою слід виконати наступні кроки:

1) Завантажити розроблену клієнтську частину комплексу для обміну повідомленнями;

2) Обрати одного з користувачів системи, які відображаються у лівій частині вікна програми;

3) При необхідності відправлення текстового повідомлення, слід набрати у нижньому полі правої частини вікна програми нове текстове повідомлення і натиснути кнопку «Відправити»;

4) При необхідності відправлення файлу, слід натиснути кнопку «Файл…» внизу нижньої частини вікна програми, та у діалоговому вікні, що відкривається обрати файл і натиснути кнопку ОК;

5) При отриманні нових повідомлень імена відповідних користувачів змінюють зовнішній вигляд у загальному переліку, що розміщений у лівій частині вікна клієнтської частини. Для перегляду цих повідомлень слід клацнути на імені відповідного користувача.

6) При завершенні сеансу роботи над проектом, слід закрити вікно клієнтської частини.

Висновки по розділу

Таким чином, у даному розділі описано особливості реалізації програмного комплексу децентралізованого месенжеру, розробленого для ефективного управління проектами.

По-перше, розроблено графічний інтерфейс користувача системи, причому як для клієнтської, так і для серверної частин. Далі наводяться особливості програмної реалізації цих двох складових. Описуються результати тестування, що, коротко кажучи, показало достатню ефективність розробленого програмного продукту. Для зручності користування продуктом створено комплект документації із інструкції по встановленню системи та користування її клієнтською частиною.

ВИСНОВКИ

У даній роботі розроблено програмний комплекс, що являє собою децентралізований месенжер, створений з метою отримання ефективного засобу комунікації учасників процесу управління проектами.

У роботі здійснено докладний аналіз предметної галузі та встановлено наявність уже готових рішень, які однак мають ряд недоліків, що визначають доцільність проведення власної розробки (іміджеве питання, проблема довгострокової стабільності та доступності продукту, гнучкості та свободи у необхідних доробках). Відповідно, була розроблена архітектура програмного комплексу яка базується на технології «клієнт-сервер» у її спрощеному варіанті, коли на сервері зберігаються тільки дані про користувачів месенджеру, а усі повідомлення зберігаються виключно на клієнтах. Клієнтські частини отримуючи IP-адресу бажаного адресата від сервера, після цього переходять до прямого спілкування між собою, реалізуючи концепцію децентралізації.

Далі обґрунтовано вибір об’єктно-орієнтованої технології розробки та мови загального призначення C#. Дана мова є однією з найбільш популярних на сьогоднішній день при розробці ПЗ для настільних систем під управлінням ОС Windows (що і є цільовою платформою під час роботи з питань управління проектами). Відповідно до обраної мови програмування, використано інтегроване середовище розробки Microsoft Visual Studio 2019 Community, що є вільним для використання окремими розробниками, навіть і для комерційних цілей. Після обрання засобів розробки було спроектовано необхідні для роботи системи алгоритмічні складові (представлені у вигляді блок-схем алгоритмів).

Потім було проведено програмну реалізацію продукту, в результаті чого отримано готове рішення у вигляді програмного комплексу з двох частин: клієнтської та серверної. Системне тестування роботи усієї системи показало

належний рівень стабільності її роботи, та загальну ефективність виконання продуктом поставленої мети.

Дане програмне рішення може використовуватися під час спільної роботи колективів в галузі управління проектами.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Воронцов Ю.А., Козинец А.В. Стандарты веб-сервисов для создания распределенных информационных систем / Ю.А.Воронцов, А.В.Козинец, // Век качества. 2015. №3.

2. Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen, Kelli A. Houston. Object-Oriented Analysis and Design with Applications. – Boston: Addison-Wesley Professional; 3 edition, 2007. – 720 p.

3. Brett D. McLaughlin, Gary Pollice, Dave West. Head First Object-Oriented Analysis and Design. - Sebastopol: O'Reilly Media; 1 edition, 2006. – 636 p.

4. Jon Skeet. C# in Depth. – New-York: Manning Publications; 4 edition, 2019. – 528 p.

5. Joseph Albahari, Ben Albahari. C# 7.0 in a Nutshell: The Definitive Reference. – Sebastopol: O'Reilly Media; 1 edition, 2017. – 1088 p.

6. Christian Nagel. Professional C# 7 and .NET Core 2.0. – Birmigham: Wrox; 7 edition, 2018. – 1440 p.

7. Alan Cooper, Robert Reimann, Dave Cronin. About Face 3: The Essentials of Interaction Design. — Wiley, 2007. - 610 p.

8. Rogers, Y. and Sharp, H. and Preece, J. Interaction Design: Beyond Human-Computer Interaction. — John Wiley and Sons Ltd, 2009. – 455 p.

9. C. J. Date, A. Kannan and S. Swamynathan, An Introduction to Database Systems, Pearson Education, Eighth Edition, 2009. – 872 p.

10. Abraham Silberschatz, Henry F. Korth and S. Sudarshan, Database System Concepts, McGraw-Hill Education (Asia), Fifth Edition, 2006. – 378 p.

11. Shio Kumar Singh, Database Systems Concepts, Designs and Application, Pearson Education, Second Edition, 2011. – 490 p.

12. Peter Rob and Carlos Coronel, Database Systems Design, Implementation and Management, Thomson Learning-Course Technology, Seventh Edition, 2007. – 702 p.

13. Агуров, Павел C#. Сборник рецептов / Павел Агуров. - М.: "БХВ-Петербург", 2012. - 432 с.

14. Албахари, Джозеф C# 3.0. Справочник / Джозеф Албахари , Бен Албахари. - М.: БХВ-Петербург, 2012. - 944 с.

15. Альфред, В. Ахо Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - М.: Вильямс, 2015. - 266 с.

16. Бишоп, Дж. C# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином. Лаборатория знаний, 2013. - 472 с.

ДОДАТОК А. КОД РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Основний файл проекту клієнтської частини Form1.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Text.RegularExpressions;

namespace DMClient
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Exchange("127.0.0.1", 8888, textBox1.Text);
        }

        private string Exchange(string address, int port, string
outMessage)
        {
            TcpClient client;
            // Инициализация
            try
            {
                client = new TcpClient(address, port);
            }
            catch
            {
                MessageBox.Show("No connection to server!");
                return "";
            }
            string myIP = "";
            /*
            // =
Dns.GetHostEntry(string.Empty).AddressList[0].MapToIPv4().ToStri
ng();// Dns.GetHostByName(hostName).AddressList[0].ToString();
```

```
            IPHostEntry host;
            host = Dns.GetHostEntry(string.Empty);
            foreach (IPAddress ip in host.AddressList)
            {
                label1.Text += "\n" + ip.ToString();
                if (ip.AddressFamily ==
System.Net.Sockets.AddressFamily.InterNetwork)
                {

//System.Diagnostics.Debug.WriteLine("LocalIPadress: " + ip);
                    myIP = ip.ToString();
                }
            }
            label1.Text = myIP;
            */
            WebClient wclient = new WebClient();
            Stream sdata =
wclient.OpenRead("http://whatismyip.org/");
            StreamReader reader = new StreamReader(sdata);
            string newLine="";
            while ((newLine = reader.ReadLine()) != null)
                myIP += newLine;
            myIP = myIP.Substring(myIP.IndexOf("<a href=\"/my-
ip-address\">"));
            myIP = myIP.Substring(myIP.IndexOf(">")+1,
myIP.IndexOf("</a>")- myIP.IndexOf(">")-1);
            sdata.Close();
            label1.Text = myIP;


            /*String host = System.Net.Dns.GetHostName();
            // Получение ip-адреса.
            System.Net.IPAddress ip =
System.Net.Dns.GetHostByName(host).AddressList[1];
            label1.Text = ip.ToString();
            myIP= ip.ToString();*/

            Byte[] data = Encoding.UTF8.GetBytes(myIP);//
outMessage);
            NetworkStream stream = client.GetStream();
            try
            {
                // Отправка сообщения
                stream.Write(data, 0, data.Length);
                // Получение ответа
                Byte[] readingData = new Byte[256];
                String responseData = String.Empty;
                StringBuilder completeMessage = new
StringBuilder();
                int numberOfBytesRead = 0;
                do
                {
```

```
                    numberOfBytesRead = stream.Read(readingData,
0, readingData.Length);
                    completeMessage.AppendFormat("{0}",
Encoding.UTF8.GetString(readingData, 0, numberOfBytesRead));
                }
                while (stream.DataAvailable);
                responseData = completeMessage.ToString();
                textBox2.Text = responseData;
                return responseData;
            }
            finally
            {
                stream.Close();
                client.Close();
            }
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void textBox3_TextChanged(object sender,
EventArgs e)
        {

        }
    }
}
```

Основний файл проекту серверної частини Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace DMServer
{

    public partial class Form1 : Form
    {
        private bool run = false;
        public Form1()
        {
            InitializeComponent();
        }

        private void menuStrip1_ItemClicked(object sender,
ToolStripItemClickedEventArgs e)
        {

        }
        private async Task ListenAsync()
        {
            IPAddress localAddr = IPAddress.Parse("127.0.0.1");
            int port = 8888;
            TcpListener server = new TcpListener(localAddr,
port);
            server.Start();
            // Выполняем цикл только, если серверная часть
«включена»
            while (run)
            {
                try
                {
                    // Асинхронное подключение клиента
                    TcpClient client = await
server.AcceptTcpClientAsync();
                    NetworkStream stream = client.GetStream();
                    // Обмен данными только, если серверная
часть «включена».
                    try
                    {
                        // Читаем данные
                        if (stream.CanRead && run)
```

```
                        {
                            byte[] myReadBuffer = new
byte[1024];
                            StringBuilder myCompleteMessage =
new StringBuilder();
                            int numberOfBytesRead = 0;
                            do
                            {
                                numberOfBytesRead =
stream.Read(myReadBuffer, 0, myReadBuffer.Length);

myCompleteMessage.AppendFormat("{0}",
Encoding.UTF8.GetString(myReadBuffer, 0, numberOfBytesRead));
                            }
                            while (stream.DataAvailable);
                            textBox1.Text += "\n" +
myCompleteMessage.ToString();

                            Byte[] responseData =
Encoding.UTF8.GetBytes(textBox1.Text);
                            stream.Write(responseData, 0,
responseData.Length);
                        }
                    }
                    finally
                    {
                        stream.Close();
                        client.Close();
                    }
                }
                catch
                {
                    server.Stop();
                    break;
                }
            }
            // Если серверная часть «выключена», обязательно
останавливаем прослушивание порта.
            // Иначе потом серверная часть не «включится».
            if (!run)
            {
                server.Stop();
            }
        }
        private void button1_Click(object sender, EventArgs e)
        {
            if (!run)
            {
                label1.Text = "Running...";
                button1.Text = "Stop server";
                run = true;
                ListenAsync();
            }
```

```
            else
            {
                label1.Text = "STOPPED";
                button1.Text = "Start server";
                run = false;
            }
        }
    }
}
```

ДОДАТОК Б. ПЕРЕЛІК ГРАФІЧНОГО МАТЕРІАЛУ, ЩО ВИНОСИТЬСЯ НА ЗАХИСТ

Презентація дипломної роботи на тему:

Розробка децентралізованого месенжеру з метою ефективного управління проектами на мові C#

1

Актуальність роботи

Ефективна комунікація - важлива складова діяльності з управління проектами.

Для робочих процесів комунікацію доцільно вести за допомогою настільних програм-месенджерів.

Використання власної розробки доцільніше, ніж існуючих програм, оскільки надає іміджеві плюси, надійність у довготривалому використанні, високу гнучкість рішення.

Для більшої надійності та захисту інформації продукт доцільно реалізувати розподіленим/децентралізованим

Таким чином, розробка власного програмного продукту – децентралізованого месенджера для спілкування учасників процесів управління проектами – **актуальна задача для галузі IT.**

2

Характеристики роботи

Мета: підвищення ефективності спілкування учасників процесу управління проектом за рахунок створення можливості майже миттєвого вирішення виробничих питань невеликої та середньої складності, чого можна досягти шляхом розробки та впровадження децентралізованого програмного комплексу для обміну текстовими повідомленнями.

Задачі дослідження:

-аналіз предметної галузі та недоліки наявних рішень;

-розробити архітектуру програмного комплексу та супутню інформацію (алгоритми, описи схем і т.п.);

-здійснити програмну реалізацію;

-провести тестування програмного продукту, розробити комплект документації, зробити висновки.

3

Характеристики роботи (2)

Об'єкт дослідження: процес текстового обміну інформацією між особами, задіяними в управлінні проектами.

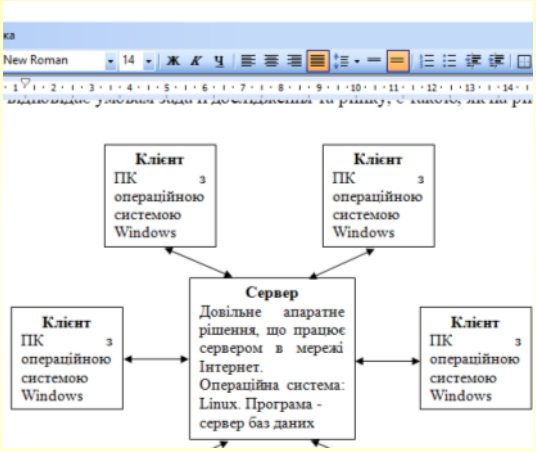
Предмет дослідження: методи та засоби ведення такого обміну.

Новизна роботи полягає у розробці відповідної децентралізованої архітектури, застосуванні простого інтерфейсу клієнтської частини при використанні надійних «бек-енд» технологій (наприклад, дублювання бази даних повідомлень, що підвищує надійність та швидкість роботи та ін.).

Практичне значення: створено робочий програмний продукт – децентралізований месенджер, що може бути легко впроваджений у роботу будь-якої невеликої та середньої компанії та надавати зручні можливості по ефективному спілкуванню в діяльності по управлінню проектами, а також і інших видах спілкування.

4

Архітектура системи



- Повідомлення зберігаються на ПК клієнтів
- Сервер потрібний виключно для ведення обліку користувачів, включаючи їх поточні IP-адреси

5

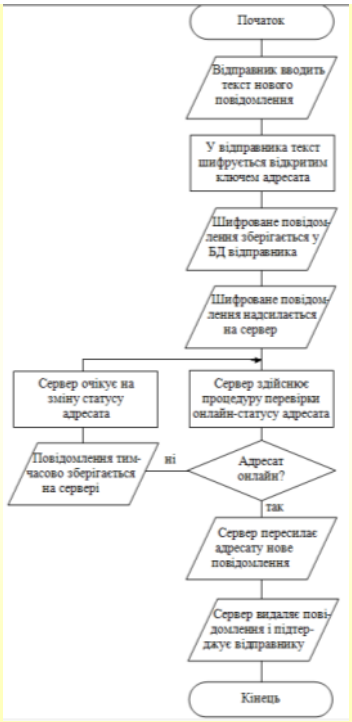
Алгоритми системи

Алгоритм типового процесу роботи користувача месенджеру у цьому програмному продукті →

6

Алгоритми системи

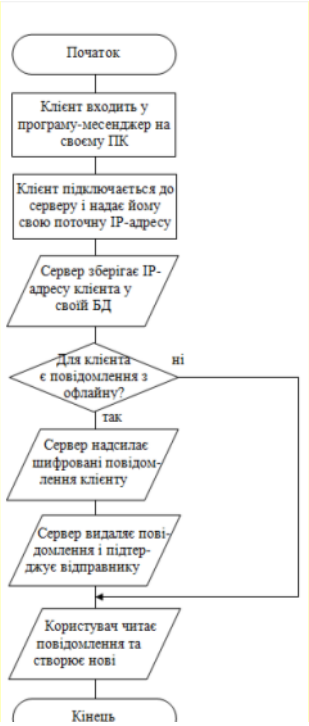
Алгоритм процесу проходження нового повідомлення через систему →



7

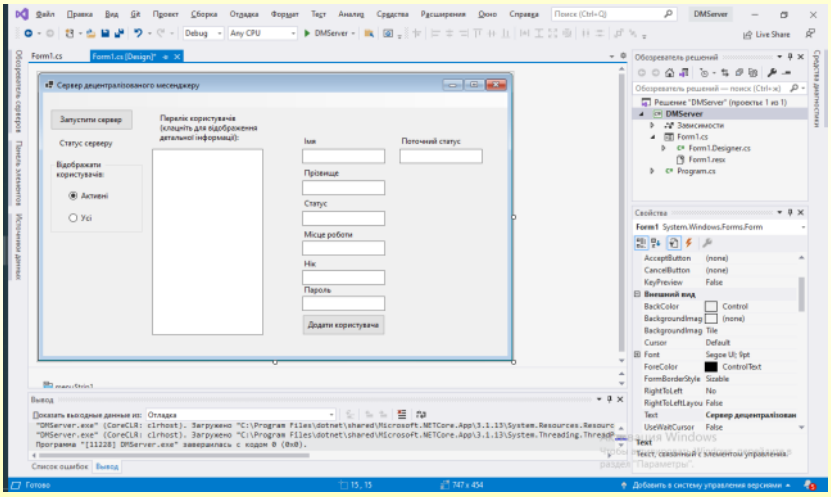
Алгоритми системи

Алгоритм процесу синхронізації клієнта, що підключається до системи →



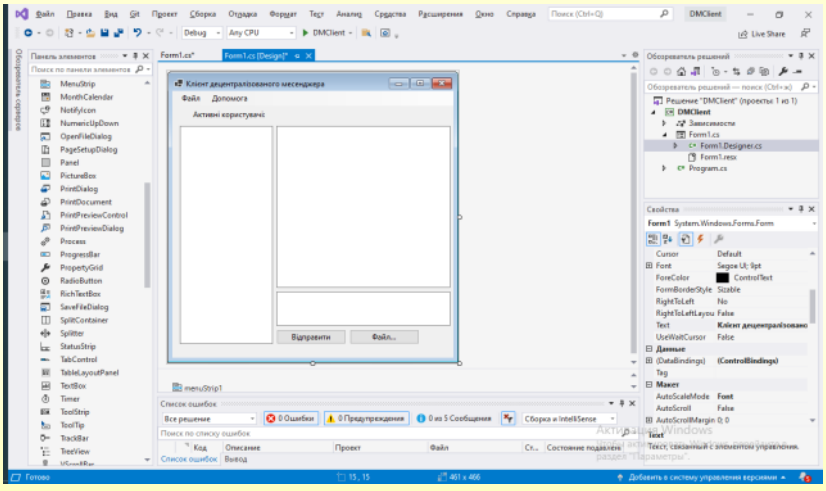
8

Інтерфейс користувача серверної частини



9

Інтерфейс користувача клієнтської частини



10

Результати тестування продукту

Системне тестування дало наступні результати:

-робота ПЗ є стабільною, без системних помилок, аварійних завершень, інших позаштатних ситуацій (і серверної, і клієнтської частин);

-продуктивність ПЗ знаходиться на достатньому рівні: зависання, зупинки або помітні паузи в роботі програми відсутні. Максимальна затримка при пересиланні повідомлень складає не більше 1 секунди;

-в цілому працювати з продуктом зручно, інформація добре сприймається і він адекватно виконує поставлене перед ним завдання утворення ефективного комунікаційного середовища для спілкування осіб, задіяних в процесі управління проектом.

11

Висновки

У роботі розроблено програмний комплекс, що являє собою децентралізований месенжер, створений з метою отримання ефективного засобу комунікації учасників процесу управління проектами.

Здійснено докладний аналіз предметної галузі та встановлено наявність уже готових рішень, які однак мають ряд недоліків, що визначають доцільність проведення власної розробки (іміджеве питання, проблема довгострокової стабільності та доступності продукту, гнучкості та свободи у необхідних доробках). Відповідно, була розроблена архітектура програмного комплексу яка базується на технології «клієнт-сервер» у її спрощеному варіанті, коли на сервері зберігаються тільки дані про користувачів месенджеру, а усі повідомлення зберігаються виключно на клієнтах. Клієнтські частини отримуючи IP-адресу бажаного адресата від сервера, після цього переходять до прямого спілкування між собою, реалізуючи концепцію децентралізації.

Далі обґрунтовано вибір об’єктно-орієнтованої технології розробки та мови загального призначення C#. Спроектовано необхідні для роботи системи алгоритмічні складові.

Проведено програмну реалізацію продукту, в результаті чого отримано готове рішення у вигляді програмного комплексу з двох частин: клієнтської та серверної. Системне тестування роботи усієї системи показало належний рівень стабільності її роботи, та загальну ефективність виконання продуктом поставленої мети.

Дане програмне рішення може використовуватися під час спільної роботи колективів в галузі управління проектами.

12