МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Розробка програмного забезпечення»

Група: 4РП-05

Дипломний проект

здобувача освіти денної форми навчання
РП.05.18.000.ДП

ПЕТРУНЯ
ВЛАДИСЛАВ
СЕРГІЙОВИЧ

м. Одеса
2022 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**

Освітня програма: «**Розробка програмного забезпечення»**

Група: **4РП-05**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) на тему:

Розробка і оптимізація клієнт-серверної частини для крос-серверних запитів на базі веб-орієнтованої системи "Шахова гра"

Проектний матеріал складається з пояснювальної записки на 45 сторінках та графічного (презентаційного) матеріалу на 10 аркушах (слайдах).

Дипломник _____ (Петруня В.С.)
Керівник _____ (Медведєв А.О.)

Консультанти:
з економічної частини _____ (Копайгородська Т.Г.)
з охорони праці_____ (Чорновол Н.І.)
з дотримання вимог ЄСКД _____ (Петрашова В.І.)
старший консультант _____ (Скорнякова О.В.)

До захисту допущений
Голова циклової комісії _____ (Скорнякова О.В.)
Завідувач відділення _____ (Суліма Ю.Ю.)

Захист «____»_____ 2022 р. Протокол ДКК №_____
Оцінка ДКК _____

Секретар ДКК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

Відділення **комп'ютерних систем** Комісія **КТ та ПІ**

Спеціальність **121 «Інженерія програмного забезпечення»**

Освітня програма **«Розробка програмного забезпечення»**

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР_____

_____

"_____"_____2022 р.

З А В Д А Н Н Я

на дипломний проект (роботу)

Здобувачеві (здобувачці) освіти Петруня Владислав Сергійович

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка і оптимізація клієнт-серверної частини для крос-серверних запитів на базі веб-орієнтованої системи "Шахова гра"

затверджена наказом по коледжу від "30" грудня 2021 р. № 306-А2-ОД

2. Термін здачі закінченого проекту (роботи) _____

3. Вихідні данні до проекту (роботи) Microsoft Visual Studio, .Net, C#, MS SQL Server, HTML JavaScript, Docker, Swagger, API, HTTP

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

1. Проектування кроссерверного запит для веб-орієнтуваної системи. 2. Економічний розрахунок. 3. Охорона праці.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Презентація (10 слайдів)

6. Консультанти по проекту (роботі), із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний	Медведєв А.О.		
Економічний	Копайгородська Т.Г.		
Охорона праці	Чорновол Н.І.		
Нормоконтроль	Петрашова В.І.		
Старший консультант	Скорнякова О.В.		

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	Розділ 1. Проектування кроссерверного запиту	17.05.2002-22.05.2022	Вик.
	Для веб-орієнтованої системи		
2	Розділ 2. Економічний розрахунок	24.05.2022	Вик.
3	Розділ 3. Охорона праці	25.05.2002-27.05.2022	Вик.
4	Розробка презентації до дипломної роботи	01.06.2002-05.06.2022	Вик.
5	Чистове оформлення пояснювальної записки	05.06.2002-08.06.2022	Вик.
6	Підготовка доповіді до захисту	14.06.2022	Вик.
7	Отримання рецензії, відповіді на зауваження	16.06.2022	Вик.
	рецензента		
8	Захист роботи	24.06.2022	Вик.

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 05.18.000 ДП ПЗ	*Дипломний проект*		
А4			*РП 05.18.000 ДП ПЗ*	*Пояснювальна*	*1*	
				записка		
				Презентація		

Ізм.	Арк.	№ докум.	Підпис	Дата	*РП 05.18.000*		
Розробив		Петруня В.С.			Розробка і оптимізація клієнт-серверної частини для крос-серверних запитів на базі веб-орієнтованої системи "Шахова гра"	Літ.	Аркуш 3 / Аркушів
Перевірив		Медведєв А.О.				Н Д П	
Н. Контр.		Петрашова В.І				ВСП «ОТФК ОНТУ» гр.РП-05	
Затверд.		Скорнякова О.В.					

ЗМІСТ

ВСТУП

Разом з появою комп'ютерів з'явилися комп'ютерні ігри, які відразу знайшли масу шанувальників. Варто відзначити, що в комп'ютерних іграх моделюються найрізноманітніші ситуації, як наприклад шахова гра.

Поточні темпи комп'ютеризації перевищують темпи розвитку всіх інших галузей. Без комп'ютерів і комп'ютерних мереж не обходиться сьогодні жодна фірма, не кажучи про великих корпораціях. Сучасна людина взаємодіє з комп'ютером постійно - на роботі, вдома, в машині і навіть в літаку. Комп'ютери стрімко входять в людське життя, займаючи своє місце в нашій свідомості, а ми часто не усвідомлюємо того, що багато в чому залежимо від працездатності цих дорогих металевих коробок. Словосполучення «комп'ютерні ігри» міцно увійшло в наше життя, кожен, хто має комп'ютер напевно зміг відчути їх привабливість, мабуть, гра закладена в саму природу людини з найдавніших часів - вистежити здобич, заманити її в пастку - це теж свого роду гра. Але тепер ми позбавлені цього в повсякденному житті. З удосконаленням комп'ютерних технологій удосконалювалися і гри, привертаючи до себе все більше уваги. На поточний момент комп'ютерна техніка досягла такого рівня розвитку, що дозволяє програмістам розробляти складні ігри, задійснював штучний інтелект.

Мета роботи – розробити клієнт-серверну інформаційно-орієнтовану систему «Шахова гра».

1 ТЕХНОЛОГІЧНИЙ РОЗДІЛ. ПРОЕКТУВАННЯ КРОССЕРВЕРНОГО ЗАПИТУ ДЛЯ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ

1.1 Розбір обміну даними між клієнтом та сервером

Обмін даними між клієнтом та сервером може відбуватися за кількома різними сценаріями. Усього можна виділити три способи отримання даних від сервера та ще один спосіб, призначений для передачі даних від клієнта до сервера. У DDE розрізняють три способи отримання даних від сервера, які називаються видами зв'язку. Ці три види зв'язку називаються холодна, тепла та гаряча. Коротко пояснимо відмінності цих видів зв'язку:

• холодний зв'язок -- cold link (обмін даними відбувається лише на запит клієнта. Сервер надсилає у відповідь дані або негативне підтвердження).

• гарячий зв'язок - hot link (клієнт "підписується" на періодичне отримання даних від сервера, після чого сервер починає передавати дані клієнту, як тільки в цьому виникає потреба. Для завершення гарячого зв'язку клієнт повинен повідомити про це сервер).

• теплий зв'язок - warm link (клієнт, як і при гарячому зв'язку, підписується на отримання оновлених даних. Однак сервер передає не дані, а тільки повідомлення про те, що має дані для клієнта. Клієнт може зажадати дані у сервера в будь-який зручний для нього момент).

Останні два види зв'язку (теплий і гарячий) називаються іноді постійним (permanent) зв'язком. Передача даних від клієнта до сервера здійснюється лише одним способом, з ініціативи клієнта, який передає серверу відповідне повідомлення, що містить дані, що посилаються.

Коли два програми обмінюються даними один з одним, вони передають один одному хендли блоків даних та атоми. Для того, щоб ці дані були доступні обом програмам, вони повинні бути глобальними. Однак треба врахувати, що зазвичай глобальні блоки даних пов'язані з тим додатком, який

їх створив, тобто при завершенні цієї програми ці дані автоматично знищуються.

Так як процес DDE-розмови є асинхронним, необхідно забезпечувати збереження даних незалежно від існування програми, що створила їх. Для цього глобальні блоки даних повинні бути поділені - при їх виділенні треба вказувати прапор GMEM_DDESHARE (або GMEM_SHARE, який є синонімом).

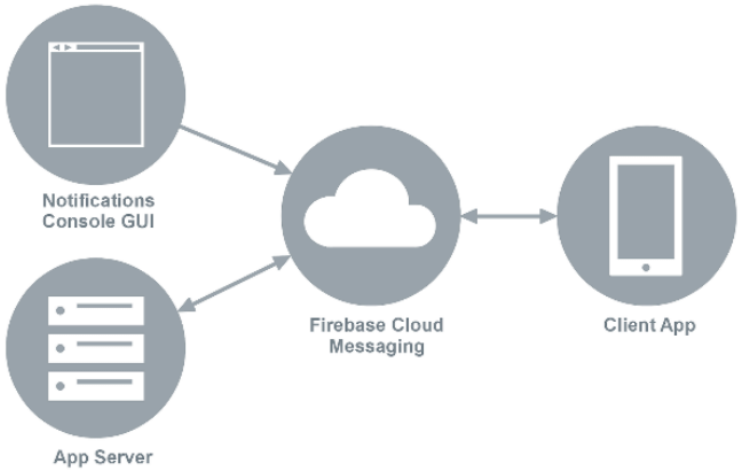

Рисунок 1.1 – Обробка даних між клієнтом та сервером

У випадку платформи Win32 використовуються попередні функції для виділення глобальних блоків даних, незважаючи на те, що блоки виділяються тільки в локальному для кожного процесу віртуальному адресному просторі. Система автоматично здійснює передачу даних з адресного простору одного процесу адресний простір іншого процесу під час передачі відповідних повідомлень.

Особливу увагу треба приділити питанню звільнення ресурсів, оскільки атоми і передані блоки даних самі не знищуються, навіть якщо всі програми, що беруть участь в DDE, завершили роботу. Усі створені об'єкти мають бути обов'язково знищені незалежно від результату операції. Складності пов'язані з тим, що той самий об'єкт може бути знищений або клієнтом, або сервером,

залежно від протікання процесу обміну, а, крім того, в процесі обміну можуть приходити різні помилки (наприклад, функція PostMessage не може надіслати повідомлення) .

Приклад базового використання подання запиту на сервер:

```
request/response paradigm ==> client/server roles
- Remote Procedure Call (RPC)
- object invocation, e.g., Remote Method Invocation (RMI)
- HTTP (the Web)
- device protocols (e.g., SCSI)
```

Рисунок 1.2 - Базовий запит на дані сервера

Деякі сокет-програми прості, і одержувач може продовжувати отримувати дані доти, доки відправник не закриє сокет, наприклад, простий додаток для передачі файлів. Більшість програм не така проста і зазвичай вимагають, щоб потік можна було розділити на кілька окремих повідомлень.

Повідомлення, яким обмінюються дві програми сокетів, має містити інформацію, щоб одержувач міг вирішити, скільки байтів очікувати від відправника та (необов'язково) що робити з отриманим повідомленням.

Для вбудовування інформації про довжину повідомлення в потік використовують кілька поширених методів, а саме:

Якщо ваші повідомлення мають фіксовану довжину, ви можете реалізувати ідентифікатор повідомлення кожного типу повідомлення, з яким працюєте. Кожен тип повідомлення має певну довжину, відому вашим клієнтським та серверним програмам. Якщо ви помістите ідентифікатор повідомлення на початок кожного повідомлення, програма-одержувач зможе визначити довжину повідомлення, якщо вона знає вміст перших декількох байтів повідомлення.

```
*-------------------------------------------------------------*
* Layout of a message between TPI client and TPI server       *
*-------------------------------------------------------------*
 01  tpi-message.
     05  tpi-message-id             pic x.
     88  tpi-request-add          value '1'.
     88  tpi-request-update       value '2'.
     88  tpi-request-update       value '2'.
     88  tpi-request-query        value '3'.
     88  tpi-request-query        value '3'.
     88  tpi-request-delete       value '4'.
     88  tpi-query-reply          value 'A'.
     88  tpi-response             value 'B'.
     05  tpi-constant             pic x(4).
     88  tpi-identifier           value 'TPI '.
```

Рисунок 1.3 – Приклад простого tcp\ip запиту

Кожен ідентифікатор повідомлення пов'язаний із фіксованою довжиною, відомою вашому додатку.

Якщо повідомлення мають змінну довжину, ви можете реалізувати поле довжини на початку кожного повідомлення. Зазвичай ви реалізували б довжину у вигляді двійкової довжини півслова зі значенням, закодованим в мережевому порядку байтів, але ви можете реалізувати його як текстове поле.

```
*-------------------------------------------------------------*
* Transaction Request Message segment                         *
*-------------------------------------------------------------*
01  TRM-message.
    05  TRM-message-length   pic 9(4) Binary Value 20.
    05  filler                    pic x(2) Value low-value.
    05  TRM-identifier            pic x(8) Value '*TRNREQ*'.
    05  TRM-trancode              pic x(8) Value '?????'.
```

Рисунок 1.4 – Створення серверу для “Гри в шахи”

Третій метод, що найчастіше зустрічається в програмах мовою C, полягає у відправленні рядка, що закінчується нулем. Рядок із завершальним нулем - це рядок байтів, що закінчується байтом двійкового 0. Програма-одержувач зчитує всі дані, що знаходяться в потоці, а потім перебирає отриманий буфер, відокремлюючи кожен запис у точці, де знайдено нульовий байт. Коли отримані записи оброблені, програма видає нове читання наступного блоку даних у потоці.

Якщо повідомлення містять лише символьні дані, ви можете вказати будь-яке байтове значення, що не відображається, як маркер кінця повідомлення. Хоча цей метод найчастіше зустрічається в програмах C, його можна використовувати з будь-якою мовою програмування.

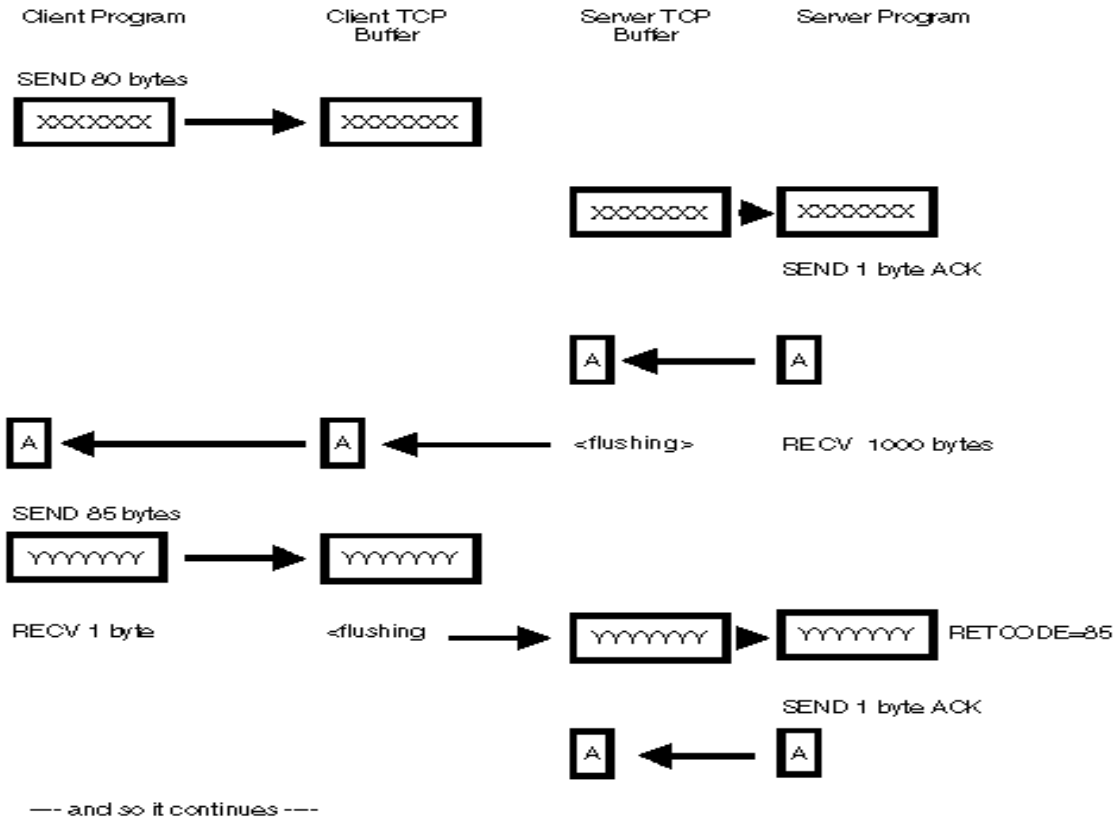

Рисунок 1.4 - Приняття запиту сервером

На рисунку 1.4 клієнт надсилає 80-байтове повідомлення. Сервер видав виклик recv() для 1000 байтів, але отримав лише 80 байтів (RETCODE=80). Це створює проблему, оскільки немає гарантії, що сервер отримає повне 80-байтове повідомлення під час дзвінка. Він може отримати лише 30 байтів, але за допомогою цієї техніки він не може знати, що йому не вистачає ще 50 байтів. Чим менше повідомлення, тим менша ймовірність того, що сервер отримає лише частину повного повідомлення. Цей метод широко використовується, але його слід використовувати тільки в контрольованих середовищах або програмах, в яких ви використовуєте неблокуючі виклики сокетів для реалізації власної логіки тайм-ауту. Методи ідентифікатора типу повідомлення та слова дескриптора запису вимагають, щоб програма, що

приймає, могла дізнатися вміст перших байтів у повідомленні, перш ніж вона прочитає все повідомлення.

Якщо це проблема для програми, використовуйте прапорець peek у виклику recv socket(). Виклик recv() з увімкненим прапором peek не видаляє дані з буферів TCP, а копіює запитану кількість байтів у буфер програми, вказаний вами при викликі recv(). Наприклад, якщо поле довжини повідомлення або поле ідентифікатора повідомлення знаходиться в межах перших 5 байтів кожного повідомлення, виконайте наступний виклик recv():

```
*-------------------------------------------------------------*
* Peek buffer and length fields for RECV call                 *
*-------------------------------------------------------------*
01  soket-recv                    pic x(16) value 'RECV'.
01  recv-flag-peek                pic 9(8) binary value 2.
01  recv-peek-len                 pic 9(8) binary value 5.
01  recv-peek-buffer.
    05  message-id                pic x value space.
        88  tpi-query-reply       value 'A'.
        88  tpi-response          value 'B'.
    05  message-constant          pic x(4).
        88  tpi-identifier        value 'TPI'.
01  socket-descriptor             pic 9(4) binary value  0.
01  errno                         pic 9(8) binary value  0.
01  retcode                       pic s9(8) binary value 0.
*-------------------------------------------------------------*
* Peek at first 5 bytes of client data                        *
*-------------------------------------------------------------*
   call 'EZASOKET' using soket-recv
       socket-descriptor
       recv-flag-peek
       recv-peek-len
       recv-peek-buffer
       errno
       retcode.
   if retcode < 0 then
      - process error -
   if retcode = 0 then
      - process client closed socket -
   if not TPI-identifier then
      - translate recv-peek-buffer from ASCII to EBCDIC -
```

Рисунок 1.5 - Для створення гри виділяємо місце на сервері

Як висновок слід зазначити, що використання баффера для створення обміну даними для гри в майбутньому буде більш спрощеною формою обробки процедури.

1.2 Клієнт-серверні запити для зберігання обробки процедур

"Клієнт-сервер" - це модель взаємодії комп'ютерів у мережі. Як правило, комп'ютери є рівноправними. Кожен з них має своє, відмінне від інших, призначення, грає свою роль. Деякі комп'ютери в мережі володіють та розпоряджаються інформаційно-обчислювальними ресурсами, такими як процесори, файлова система, поштова служба, служба друку, база даних. Інші комп'ютери мають можливість звертатися до цих служб, користуючись послугами перших. Комп'ютер, управляючий тим чи іншим ресурсом, прийнято називати сервером цього ресурсу, а комп'ютер, який бажає ним скористатися - клієнтом. Конкретний сервер визначається видом ресурсу, яким він володіє. Так, якщо ресурсом є бази даних, то йдеться про сервер баз даних, призначення якого - обслуговувати запити клієнтів, пов'язані з обробкою даних; якщо ресурс - це файлова система, то говорять про файловий сервер, або файл-сервері, і т.д.

У мережі той самий комп'ютер може виконувати роль як клієнта, і сервера. Наприклад, в інформаційній системі, що включає персональні комп'ютери, велику ЕОМ і міні-комп'ютер під управлінням UNIX, останній може виступати як сервер бази даних, обслуговуючи запити від клієнтів - персональних комп'ютерів, так і в якості клієнта, направляючи запити великий ЕОМ.

Той самий принцип поширюється і взаємодія програм. Якщо одна з них виконує деякі функції, надаючи іншим відповідний набір послуг, така програма виступає як сервер. Програми, які користуються цими послугами, називається клієнтами. Так, ядро реляційної SQL-орієнтованої СУБД часто називають сервером бази даних, або SQL-сервером, а програму, що звертається до нього за послугами з обробки даних – SQL-клієнтом.

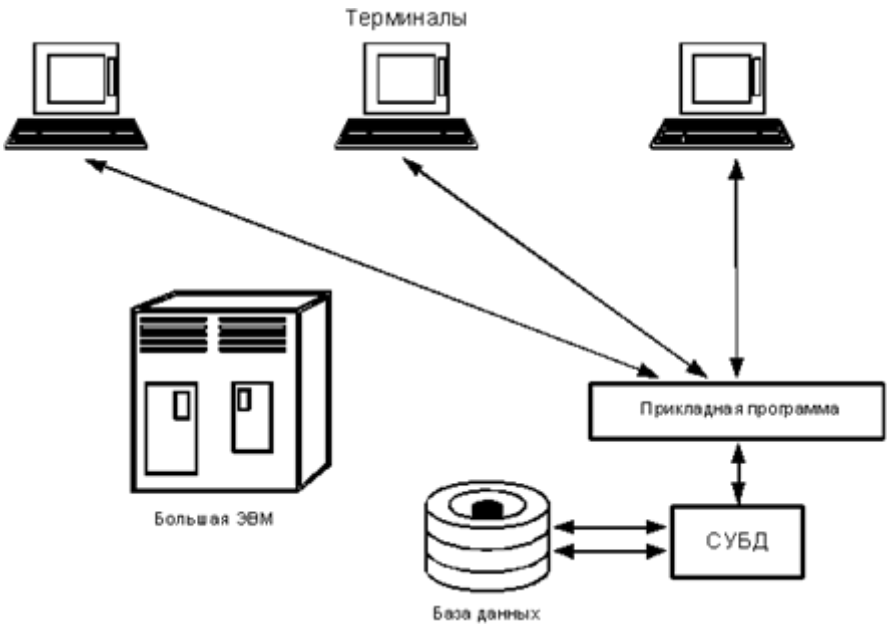

Рисунок 1.6 - Надсилання запиту на БД

Спочатку СУБД мали централізовану архітектуру (рис. 1.6). У ній сама СУБД та прикладні програми, які працювали з базами даних, функціонували на центральному комп'ютері (велика ЕОМ чи міні-комп'ютер). Там розташовувалися бази даних. До центрального комп'ютера були підключені термінали, що виступали як робочі місця користувачів. Всі процеси, пов'язані з обробкою даних, як-то: підтримка введення, здійснюваного користувачем, формування, оптимізація та виконання запитів, обмін з пристроями зовнішньої пам'яті тощо, виконувались на центральному комп'ютері, що пред'являло жорсткі вимоги до його продуктивності. Особливості СУБД першого покоління безпосередньо пов'язані з архітектурою систем великих ЕОМ та міні-комп'ютерів та й адекватно відображають усі їхні переваги та недоліки. Проте нас більше цікавить сучасний стан розрахованих на багато користувачів СУБД, для яких архітектура "клієнт-сервер" стала фактичним стандартом.

Для більш чіткого уявлення про її особливості необхідно розглянути кілька моделей технології "клієнт-сервер", що буде зроблено.

Якщо передбачається, що проектована інформаційна система (ІС) матиме технологію " клієнт-сервер " , це означає, що прикладні програми, реалізовані у межах, матимуть розподілений характер. Іншими словами,

частина функцій прикладної програми (або, простіше, програми) буде реалізована в програмі-клієнті, інша - у програмі-сервері, причому для їхньої взаємодії буде визначено деякий протокол.

Основний принцип технології " клієнт-сервер " полягає у розділенні функцій стандартного інтерактивного додатка на чотири групи, що мають різну природу. Перша група - це функції введення та відображення даних. Друга група поєднує суто прикладні функції, характерні для даної предметної області (наприклад, для банківської системи - відкриття рахунку, переведення грошей з одного рахунку на інший тощо). До третьої групи відносяться фундаментальні функції зберігання та управління інформаційними ресурсами (базами даних, файловими системами тощо). Нарешті, функції четвертої групи - це службові функції (що грають зв'язок між функціями перших трьох груп.

Відповідно до цього у будь-якому додатку виділяються такі логічні компоненти:

· компонент уявлення, що реалізує функції першої групи;

· Прикладний компонент, що підтримує функції другої групи;

· Компонент доступу до інформаційних ресурсів, що підтримує функції третьої груп, а також вводяться та уточнюються угоди про способи їх взаємодії (протокол взаємодії).

Відмінності у реалізаціях технології " клієнт-сервер " визначаються чотирма чинниками. По-перше, тим, які види програмного забезпечення інтегровані кожен із цих компонентів. По-друге, тим, які механізми програмного забезпечення застосовуються для реалізації функцій всіх трьох груп. По-третє, як логічні компоненти розподіляються між комп'ютерами у мережі. По-четверте, які механізми використовують для зв'язку компонентів між собою.

Виділяються чотири підходи, реалізовані в моделях:

• модель файлового сервера (File Server – FS);

• модель доступу до віддалених даних (Remote Data Access – RDA);

• модель півночі бази даних (DataBase Server – DBS);

• модель сервера програм (Application Server - AS).

1.3 Моделювання серверної частини для зберігання даних і запитів користувача

Більшість коду для підтримки динамічного веб-сайту має виконуватися на сервері. Створення цього коду відоме як «програмування серверної частини» (або іноді «програмування бекенда»).

Схема нижче вказує просту архітектуру динамічного веб-сайту. Як і на попередній схемі, браузери відправляють HTTP-запити на сервер, потім сервер обробляє запити та повертає відповідні HTTP-відповіді.

Запити статичних ресурсів обробляються як і, як і статичних сайтів (статичні ресурси — це будь-які файли, які змінюються, зазвичай це: CSS, JavaScript, зображення, попередньо створені PDF-файли та інше).

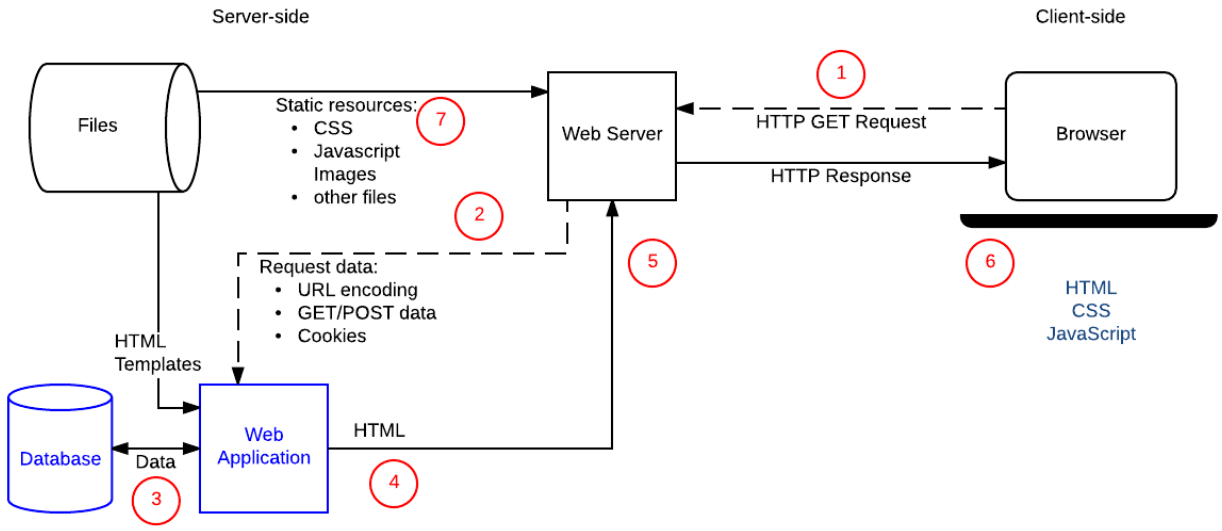

Рисунок 1.7 – Схема роботи серверної частини

Запити динамічних даних надсилаються (2) у код серверної частини (показано на діаграмі як веб-додаток). Для «динамічних запитів» сервер інтерпретує запит, читає необхідну інформацію з бази даних (3), комбінує вилучені дані з шаблонами HTML та повертає відповідь, що містить згенерований HTML (5, 6).

Тепер звернемо увагу на код, задіяний у серверній частині та клієнтській частині. У кожному разі код значно відрізняється:

Вони мають різні цілі та призначення.

Як правило, вони не використовують одні й ті ж мови програмування (виняток становить JavaScript, який можна використовувати на стороні сервера та клієнта).

Вони виконуються у різних середовищах операційної системи.

Код, який виконується в браузері, відомий як код клієнтської частини, перш за все пов'язаний з покращенням зовнішнього вигляду та поведінки відображуваної веб-сторінки. Це включає вибір і стилізацію компонентів інтерфейсу користувача, створення макетів, навігацію, перевірку форм і т. д. Навпаки, програмування веб-сайту на стороні сервера в основному включає вибір вмісту, який повертається браузеру у відповідь на запити. Код на стороні сервера обробляє такі завдання, як перевірка надісланих даних та запитів, використання баз даних для зберігання та вилучення даних та відправлення правильних даних клієнту за необхідності.

Код клієнтської частини написаний з використанням HTML, CSS та JavaScript - він запускається у веб-браузері і практично не має доступу до базової операційної системи (включаючи обмежений доступ до файлової системи).

Веб-розробники не можуть контролювати, який браузер може використовувати кожен користувач для перегляду веб-сайту - браузери забезпечують суперечливі рівні сумісності з функціями коду на стороні клієнта, і одним із завдань програмування на стороні клієнта є витончена обробка відмінностей у підтримці браузера.

Основними класами будуть:

- Клітина (клас Spot): Клітина являє собою один блок сітки та додаткові елементи.
- Фігура (клас Piece) – базовий блок системи.

Кожен об'єкт класу фігура, а точніше об'єкт спадкоємця класу Piece, буде розташований на клітці. Фігура (клас Piece) буде абстрактним класом. Класи спадкоємці (пішак (клас Pawn), король (клас King), ферзь (клас Queen), тура (клас Rook), кінь (клас Knight), слон або офіцер (клас Bishop)) реалізують абстрактні методи.

- Дошка (клас Board): набір клітинок 8x8.
- Гравець (клас Player): представляє одного із учасників гри.
- Рух (клас Move): Рух (хід) представляє хід у грі, містить стартову та кінцеву клітинку. Рух також міститиме трек гравця, який робить ходи.
- Гра (клас Game): Цей клас контролює потік гри.

Він містить шлях та відстежує шлях всіх ходів гри, які зробив гравець, а також фінальний результат гри.

```java
public class Spot {
    private Piece piece;
    private int x;
    private int y;

    public Spot(int x, int y, Piece piece)
    {
        this.setPiece(piece);
        this.setX(x);
        this.setY(y);
    }


    public Piece getPiece() // метод возвращает объект фигуру
    {
        return this.piece;
    }

    public void setPiece(Piece p)
    {
        this.piece = p;
    }

    public int getX()
    {
        return this.x;
    }

    public void setX(int x)
    {
        this.x = x;
    }

    public int getY()
    {
        return this.y;
    }

    public void setY(int y)
    {
        this.y = y;
    }
}
```

Фігура (клас Piece): абстрактний клас для представлення загальної функціональності всіх шахових фігур:

```
public abstract class Piece {

    private boolean killed = false;
    private boolean white = false;

    public Piece(boolean white)
    {
        this.setWhite(white);
    }

    public boolean isWhite()
    {
        return this.white;
    }

    public void setWhite(boolean white)
    {
        this.white = white;
    }

    public boolean isKilled()
    {
        return this.killed;
    }

    public void setKilled(boolean killed)
    {
        this.killed = killed;
    }

    public abstract boolean canMove(Board board,
                                    Spot start, Spot end);
}
```

Кінь (клас Knight): представляє шахівницю - кінь:

```
public class Knight extends Piece {
    public Knight(boolean white)
    {
        super(white);
    }

    @Override
    public boolean canMove(Board board, Spot start,
                                            Spot end)
    {
        // we can't move the piece to a spot that has
        // a piece of the same colour
        if (end.getPiece().isWhite() == this.isWhite()) {
            return false;
        }

        int x = Math.abs(start.getX() - end.getX());
        int y = Math.abs(start.getY() - end.getY());
        return x * y == 2;
    }
}
```

Так само, ми можемо створити класи для інших фігур, таких як пішака, кінь, слон, тура, ферзь, король.

1.4 Збережені процедури

ORM або Object-relational mapping (укр. Об'єктно-реляційне відображення) - це технологія програмування, яка дозволяє перетворювати несумісні типи моделей в ООП, зокрема, між сховищем даних і об'єктами програмування. ORM використовується для спрощення процесу збереження об'єктів в реляційну базу даних і їх вилучення, при цьому ORM сама піклується про перетворення даних між двома несумісними станами. Більшість ORM-інструментів значною мірою покладаються на метадані бази даних і об'єктів, так що об'єктам нічого не потрібно знати про структуру бази даних, а базі даних - нічого про те, як дані організовані в додатку. ORM забезпечує повне розділення завдань в добре спроектованих додатках, при якому і база даних, і додаток можуть працювати з даними кожен у своїй вихідній формі.

Рис. 2.1 – Принцип роботи ORM

Принцип роботи ORM. Ключовою особливістю ORM є відображення, яке використовується для прив'язки об'єкта до його даними в БД. ORM як би створює «віртуальну» схему бази даних в пам'яті і дозволяє маніпулювати даними вже на рівні об'єктів. Відображення показує як об'єкт і його властивості пов'язані з однією або декількома таблицями і їх полями в базі даних. ORM використовує інформацію цього відображення для управління процесом перетворення даних між базою і формами об'єктів, а також для

створення SQL-запитів для вставки, оновлення та видалення даних у відповідь на зміни, які додаток вносить в ці об'єкти.

Entity Framework - це платформа ORM з відкритим вихідним кодом для додатків .NET, підтримувана Microsoft. Це дозволяє розробникам працювати з даними, використовуючи об'єкти класів, специфічних для предметної області, що не зосереджуючись на базових таблицях бази даних і шпальтах, де зберігаються ці дані. З *Entity Framework* розробники можуть працювати на більш високому рівні абстракції, коли мають справу з даними, і можуть створювати і підтримувати орієнтовані на дані додатки з меншою кількістю коду в порівнянні з традиційними програмами.

Офіційне визначення: «*Entity Framework* - це об'єктно-реляційний картограф (ORM), який дозволяє розробникам .NET працювати з базою даних, використовуючи об'єкти .NET. Це усуває необхідність в більшій частині коду доступу до даних, який зазвичай припадає писати розробникам ».

Особливості Entity Framework

- Кросплатформеність: EF Core - це кроссплатформенная інфраструктура, яка може працювати в Windows, Linux і Mac.
- Моделювання: EF (Entity Framework) створює EDM (Entity Data Model) на основі об'єктів POCO (Plain Old CLR Object) з властивостями отримання / установки різних типів даних. Ця модель використовується при запиті або збереженні даних об'єкта в базовій базі даних.
- Запити: EF дозволяє нам використовувати запити LINQ (C # / VB.NET) для отримання даних з базової бази даних. Постачальник бази даних переведе ці запити LINQ на специфічний для бази даних мову запитів (наприклад, SQL для реляційної бази даних). EF також дозволяє нам виконувати необроблені запити SQL безпосередньо до бази даних.
- Відстеження змін: EF відстежує зміни, що відбулися з екземплярами ваших сутностей (значеннями властивостей), які необхідно відправити в базу даних.

- Збереження: EF виконує команди INSERT, UPDATE і DELETE для бази даних на основі змін, що відбулися з вашими сутностями при виклику SaveChanges () методу. EF також надає асинхронний SaveChangesAsync () метод.
- Паралелізм: EF за замовчуванням використовує Оптимістичний паралелізм для захисту перезаписувати змін, зроблених іншим користувачем з моменту отримання даних з бази даних.
- Транзакції: EF виконує автоматичне керування транзакціями при запиті або збереженні даних. Він також надає опції для настройки управління транзакціями.
- Кешування: EF включає перший рівень кешування з коробки. Таким чином, повторний запит поверне дані з кешу, а не потрапить в базу даних.
- Вбудовані угоди: EF слід угодами за шаблоном програмування конфігурації і включає в себе набір правил за замовчуванням, які автоматично конфігурують модель EF.
- Конфігурації: EF дозволяє нам конфігурувати модель EF, використовуючи атрибути анотації даних або Fluent API для перевизначення угод за замовчуванням.
- Міграції: EF надає набір команд міграції, які можна виконувати на консолі диспетчера пакетів NuGet або в інтерфейсі командного рядка для створення або управління базової схемою бази даних.

В цьому проекті було створено базу даних. Її підключення було реалізовано за допомогою *connectionString,* яка була добавлена да файлу *Web.config* наступним чином:

```
<connectionStrings>
<add name="EFDbContext" connectionString="Data
Source=(localdb)\MSSQLLocalDB;Initial
Catalog=Admins;Integrated Security=True;Connect
```

```
Timeout=30;Encrypt=False;TrustServerCertificate=False;Applicat
ionIntent=ReadWrite;MultiSubnetFailover=False"
providerName="System.Data.SqlClient"/>
</connectionStrings>
<appSettings>
```

Даний алгоритм звертається до EFDbContext, який використовує DbSet для створення, читання, оновлення та видалення елементів бази даних. А далі вже їде підключення до бази, якщо бази даних не існую, то при правильному вказані маршрутизації автоматично створиться нульова база даних, яка має всі властивості, які ми вказали в моделі.

Як правило, для зберігання моделей створюється в проекті окрема папка Models. Моделі подання нерідко містяться в окрему папку, яка нерідко називається ViewModels. У реальності, це можуть бути каталоги з будь-якими називання, можна поміщати моделі хоч в корінь проекту, але більш поширеним стилем є назви Models і ViewModels.

Наприклад, створимо новий проект ASP.NET Core по типу Web Application (Model-View-Controller). За замовчуванням він вже містить папку Models, в якій за замовчуванням визначено клас ErrorViewModel - модель, яка використовується для виведення інформації про помилки в додатку.

У додатку ASP.NET MVC Core моделі можна розділити за ступенем застосування на кілька груп:

• Моделі, об'єкти яких зберігаються в спеціальних сховищах даних (наприклад, в базах даних, файлах xml і т.д.)

• Моделі, які використовуються для передачі даних уявлення або навпаки, для отримання даних з вистави. Такі моделі ще називатися моделями подання

• Допоміжні моделі для проміжних обчислень.

1.5 Тестування та рефакторинг даних

До запуску програми в виробництво, коли воно стане доступним користувачам, важливо переконатися, що дане додаток функціонує, як і повинно, що в ньому немає помилок. Для перевірки додатку ми можемо використовувати різні схеми і механізми тестування. Одним з таких механізмів є юніт-тести.

Юніт-тести дозволяють швидко і автоматично протестувати окремі компоненти програми незалежно від іншої його частини. Не завжди юніт-тести можуть покрити весь код програми, але тим не менше вони дозволяють істотно зменшити кількість помилок вже на етапі розробки.

Немає необхідності тестувати код використовуваного фреймворка або використовуваних залежностей. Тестувати треба тільки той код, який був написаний.

Треба відзначити, що в цілому концепція юніт-тестів не є непорушною вимогою до веб-розробці, та й взагалі до розробки. Є думка, що юніт-тести обов'язково повинні покривати весь код проекту, хтось вважає, що юніт-тести можна використовувати переважно для особливо складних моментів у коді програми, якоїсь складної логіки. Деякі не використовують юніт-тести.

Але тим не менше юніт-тести несуть потенційні переваги при розробці, до яких слід віднести не тільки власне перевірку результату і тестування коду, але і інші, як наприклад, написання слабосвязанних компонентів відповідно до принципів SOLID. Адже щоб тестувати компоненти програми незалежно один від одного, нам треба, щоб вони були слабо зв'язаної. А подібне побудова додатки в подальшому може позитивно позначитися на його подальшій модифікації і підтримки.

Більшість юніт-тестів так чи інакше мають ряд таких ознак:

Тестування невеликих ділянок коду ("юнітів")

Для створення юніт-тестів вибираються невеликі ділянки коду, які треба протестувати. Тестований ділянку, як правило, повинен бути менше класу. У більшості випадків тестується окремий метод класу або навіть частина

функціоналу методу. Упор на невеликі ділянки дозволяє досить швидко писати прості тести.

Одного разу написаний код нерідко читають багато разів, тому важливо писати зрозумілий код. Особливо це важливо в юніт-тестах, де в разі невдачі при тестуванні розробник повинен швидко прочитати вихідний код і зрозуміти в чому проблема і як її виправити. А використання невеликих ділянок коду значно спрощує подібну роботу.

Тестування в ізоляції від решти коду

При тестуванні важливо ізолювати тестований код від решти програми, з якою він взаємодіє, щоб потім чітко визначити можливість помилок саме в цьому ізольованому коді. Що спрощує і підвищує контроль над окремими компонентами програми.

Автоматизація тестів

Створення юніт-тестів для невеликих ділянок коду веде до того, що кількість цих юніт-тестів стає дуже велике. Якщо процес отримання результатів і проведення тестів не автоматизовано, то це може привести до непродуктивної витраті робочого часу і зниження продуктивності. Тому важливо, щоб результати юніт-тестів представляли собою просте рішення, яке означає, пройдений тест чи ні. Для автоматизації процесу розробники зазвичай звертаються до фреймворками юніт-тестування.

Тестування тільки загальнодоступних кінцевих точок

Навіть невеликі зміни в класі можуть привести до невдачі багатьох юніт-тестів, оскільки реалізація використовуваного класу змінилася. Тому при написанні юніт-тестів обмежуються тільки загальнодоступними кінцевими точками, що дозволяє ізолювати юніт-тести від багатьох деталей внутрішньої реалізації компонента. В результаті зменшується ймовірність, що зміни в класах можуть привести до провалу юніт-тестів.

Фреймворки тестування

Для написання юніт-тестів можна створювати весь необхідний функціонал, використовувати якісь свої способи тестування, однак, як правило, для цього застосовуються спеціальні фреймворки. Деякі з них:

xUnit.net: фреймворк тестування для платформи .NET. Найбільш популярний фреймворк для роботи саме з .NET Core і ASP.NET Core

MS Test: фреймворк юніт-тестування від компанії Microsoft, який за замовчуванням включений в Visual Studio і який також можна використовувати з .NET Core

NUnit: портований фреймворк з JUnit для платформи .NET

Дані фреймворки надають нескладний API, який дозволяє швидко написати і автоматично перевірити тести.Разработка через тестирование **(Test-Driven-Development)**

Окремо варто сказати про концепцію TDD або розробка через тестування. TDD представляє процес застосування юніт-тестів, при якому спочатку пишуться тести, а потім вже програмний код, достатній для виконання цих тестів.

Використання TDD дозволяє знизити кількість потенційних багів в додатку. Створюючи тести перед написанням коду, ми тим самим описуємо спосіб поведінки майбутніх компонентів, не зв'язуючи себе при цьому з конкретною реалізацією цих тестованих компонентів (тим більше що реалізація на момент створення тесту ще не існує). Таким чином, тести допомагають оформити і описати API майбутніх компонентів [9].

2 ЕКОНОМІЧНИЙ РОЗРАХУНОК

2.1 Резюме

В даному дипломному проекті розроблене та оптимізовано клієнт-серверної частини для кросерверних запитів на базі веб-орієнтованої системи "Шахова гра", за допомогою якої можна грати в шахи та розставляти свої позицію, що може бути корисним для вирішення задач.

Ефективність кожного програмного продукту визначається його якістю та ефективністю процесу розробки. Якість ПП визначається наступними складовими: з точки зору користувача; з позиції використання ресурсів; виконання вимог до програмного забезпечення.

Оцінка якості програмного продукту з точки зору користувача визначається необхідним на стадії функціонування розміром оперативної пам'яті ЕОТ, витратами машинного часу, пропускною спроможністю каналів передачі даних. Оцінка якості програмного продукту включає визначення трудомісткості і вартості його створення.

2.2. Визначення трудомісткості розробки програмного забезпечення.

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку. Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначається обсяг програмних засобів, у тисячах умовних машинних команд програми аналога

Каталог аналогів

Таблиця 4.1

Найменування ПП	Обсяг функції ПП – V_o, усл. машинних командах.
1. ПП автоматизованих розрахунків	1300 – 8600
2. ПП імітаційного моделювання	1800 – 8800
3. ПП введення інформації	1060 – 5750

У таблиці 4.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт. Для нашого варіанта виділено сірим кольором.

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл.4.2

Таблиця.4.2

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
1.00	229
2.00	244
3.00	262

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): T^a = 229 x 0,8 = 183.2 (люд/годин).

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

$T_{ТЗ} = T^a p \times L_1 \times К_Н$ = 15,38

$T_{ТП} = T^a p \times L_2 \times К_Н$ = 14,10

$T_{РП} = T^a p \times L_3 \times К_Н \times К_Т$ = 78,22

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 4.2.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 4.3.);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 4.4.).

Таблиця 4.2.Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП.

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12

ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Для нашого варіанта виділено сірим кольором.

Таблиця 4.3. Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	Ступінь новизни	Значення $К_н$
А	Принципово нові ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП маючий аналог	0,7

Для нашого варіанта виділено сірим кольором.

Таблиця 4.4. Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	Значення $К_т$
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Для нашого варіанта виділено сірим кольором.

Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$Т_{тз}=Т^а *L_1*К_н= 183.2*0,12*0,7= 15,38 \quad (люд/годин)$$

Трудомісткість розробки технічного проекту

$$Т_{тп}=Т^а *L_2*К_н =183.2*0,11*0,7 = 14,10 \ (люд/годин)$$

Трудомісткість розробки робочого проекту

$$Т_{рп}=Т^а *L_3*К_н*К_т=183.2*0,61*0,7*0,7= 54,76 \ (люд/годин)$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}$= 3 (стр), розробка ТП $N_{тп}$= 10(стр), розробка робочого проекту $N_{рп}$=22 (стр), пояснювальна записка відповідно $N_{пз}$=32 (стр) Розрахунок зведений у таблицю 4.5

Таблиця 4.5. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{Pтз}$=15,38	$T_{кк}$=0,7∗$N_{тз}$ =0,7∗3=2,1	$T_{нк}$=0,15∗$N_{тз}$=0,15∗3=0,45
2.Розробка ТП	$T_{Pтп}$=14,10	$T_{кк}$=0,7∗$N_{тп}$=0,7∗10=7	$T_{нк}$=0,15∗$N_{тп}$=0,15∗10=1,5
3.Розробка РП	$T_{Pрп}$=54,76	$T_{кк}$=0,7∗$N_{рп}$=0,7∗22= 15,4	$T_{нк}$=0,15∗$N_{рп}$= 0,15∗22= 3,3
4.Розробка ПЗ	$T_{пз}$=1,5 ∗ ∗$N_{пз}$=1,5∗32 = 52,5	$T_{кк}$=0,7∗$N_{пз}$=0,7∗32= 22,4	$T_{нк}$=0,15∗$N_{пз}$= 0,15∗32= 4,8
Усього, в т.ч.:	ΣT=136,74		
- на розробку	$ΣT_р$= 84,24		
- контроль керівника		$ΣT_{кк}$= 46,9	
- нормоконтроль			$ΣT_{нк}$= 10,05

2.3 Розрахунок ціни програмного продукту.

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПО. Розрахунок основної заробітної плати виконавців приведений у таблиці 4.6. Відповідно до статті 8 «Закону про Державний бюджет України на 2022» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2022 року - 6500 гривень; мінімальну погодинну тарифну ставку – 39.26 грн.

4.3 Розрахунок ціни програмного продукту.

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПО. Розрахунок основної заробітної плати виконавців приведений у таблиці 4.6. Відповідно до статті 8 «Закону про Державний бюджет України на 2022» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2022 року - 6500 гривень; мінімальну погодинну тарифну ставку – 39.26 грн.

Таблиця 4.6 Розрахунок основної заробітної плати виконавців.

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн.	Розрахунок, грн.
1.Розробка ПП	84,24	39.26	3 307,26
2.Контроль керівника	46,9	50.00	2 345
3.Нормоконт-роль	10,05	50.00	502,5
Усього	-	-	ΣЗо= 6154,76

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в таблицю 4.7

Таблиця 4.7 Розрахунок матеріальних витрат на розробку ПО

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	52	0,4	
				$В_{м\,і}$ = 20,8
Транспортно– заготівельні витрати (10%)				$Втр_з = 0,1 \times Вм1$= 0,1*20,8= 2,08
Усього				$В_м=В_{мі}+В_{тр\_з}$= 20,8+2,08=22,16

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 4.8.

Таблиця 4.8. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	22,16	$В_м$ = 22,16
2. Основна заробітна плата	6154,76	$З_о$ = 6154,76
3.Додаткова заробітна плата	923,21	Зд = 0,15 × Зо= 0,15*6 154,76= 923,21
4.Відрахування до єдиного фонду соціального внеску	1557,15	Вє.с.в. = 0,22 × (Зо + Зд)= 0,22*(6154,76+923,21)=1557,15
5. Накладні витрати	1846,42	Внак. = 0,3 × Зо=0,3*6154,76= 1846,42

6. Повна собівартість	10503,7	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.= 22,16+6154,76+923,21+1557,15+ 1846,42=10503,7

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

$$П=(С_{пов}*Р)/100=(10\ 503,7*10)/100=1050,37$$

Де Р– плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$$Ц_о = С_{пов}+П = 10503,7+1050,37=11554,07$$

Податок на додану вартість визначаємо по наступній формулі:

$$ПДВ = 0,2 * Ц_о = 0,2*11554,07=2310,81$$

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$$Ц_р=Ц_о+ПДВ = 11554,07+2310,81=13864,88$$

3 ОХОРОНА ПРАЦІ

Вступ

Створення на робочому місці в кожному структурному підрозділі умови праці необхідно у відповідності до нормативно-правових актів, а також забезпечити додержання вимог законодавства щодо прав працівників у галузі охорони праці.

У дипломному проекті розглядається розробка і оптимізація клієнт-серверної частини для крос-серверних запитів на базі веб-орієнтованої системи "Шахова гра", тому до розгляду буде взяте робоче місце програміста.

3.1.1 Шкідливі фактори використання комп'ютерної техніки, основні заходи та засоби щодо збереження здоров'я та підвищення рівня працездатності.

Шкідливими факторами при роботі з персональним комп’ютером є неіонізуюче випромінювання промислової частоти, збільшене нервово-емоційне навантаження на оператора, збільшення навантаження на органи зору та дрібні стерео статичні рухи кінцівок. Ці фактори можуть викликати у працівника певні розлади здоров’я, зокрема підвищення артеріального тиску, кон’юнктивіти, тендовагініті та інші захворювання.

3.2 Розробка заходів з охорони праці

3.2.1 Приміщення для роботи з комп’ютером.

Приміщення, де планується діяльність працівника з використанням комп'ютера, повинні відповідати нормативно-правовим актам. Крім того, роботодавцю необхідно враховувати взаємодіючі санітарні норми освітлення, вимоги до показників мікроклімату (тепловіддача, відносна вогкість), вібрації, вібраційного шуму та вогнестійкості приміщення. Заборонено встановлювати комп'ютери у приміщеннях, розташованих у підвалах будинків.

3.2.2 Електрозабезпека

У всьому офісі лінії електромережі мають бути захищені від виникнення короткого замикання, а також від перепадів мережевої напруги, що може спричинити збоїв в роботі електронно–обчислювальної техніки.

Міцно заховані під протирадіаційними щитками, які перешкоджатимуть допустимому попаданню страйкового під напругу.

3.2.3 Освітлення.

Найкраще, якщо воно буде штучним. Сонячне світло, звичайно, хороше, але дуже непостійне і дає відблиски; тому ідеальним приміщенням буде простора світла кімната, де комп'ютер розташований далеко від вікна. Якщо такої можливості немає, треба спробувати обирати відповідне місце; Але треба пам'ятати, що в загальному випадку надлишок світла краще, ніж його недолік, лампу краще розташовувати зліва, а світильники «денного світла» можуть мерехтіти, що не є добре.

3.2.3 Мікроклімат.

Мікроклімат визначається за такими параметрами:

- температура;
- вологість;
- рухливість повітря;
- чистота повітря.

Чинники мікроклімату впливають на стан здоров'я людини, і його працездатність. Зокрема, високі температури призводять до теплових ударів, підвищення тиску, низькі – до простудних захворювань, переохолодження, низька вологість провокує пересихання слизових оболонок дихальних шляхів. Все це може призвести до професійних захворювань. У рамках принципів охорони праці першорядним заходом вважається забезпечення правильного мікроклімату робочого місця.

Для постійних робочих місць, якими є робочі місця операторів ПК, встановлені оптимальні параметри мікроклімату, а за неможливості їх дотримання використовують допустимі параметри

Холодний: Температура повітря в приміщенні 22...24°С;
відносна вологість 40... 60%;
швидкість руху повітря до 0,1…0,2 м/с.

Теплий: Температура повітря в приміщенні 23...25°С;
відносна вологість 40...60%;
швидкість руху повітря 0,1...0,2 м/с.

В приміщеннях, де розташовані робочі місця користувачів комп'ютерів використовується припливно-витяжна система вентиляції та застосування кондиціонерів.

Для збереження здоров'я співробітників та для забезпечення комфортних умов праці та працездатності персоналу необхідно забезпечити здоровий мікроклімат. А системи обігріву, охолодження, вентиляції та кондиціювання допоможуть дотримуватися необхідних параметрів мікроклімату.

3.2.5 Вимоги безпеки під час роботи з комп'ютером:

При виконанні робіт за персональним комп'ютером працівник повинен:

- витримувати відстань від очей до екрана комп'ютером в межах 60 - 70см;
- дотримуватися внутрішньо змінного режиму праці та відпочинку, регламентованих перерв у роботі, а саме (при 8-годинній денній робочій зміні):
- для розробників програм - тривалістю 15 хвилин через кожну годину роботи;
- для інших категорій працівників - тривалістю 15 хвилин через кожні дві години роботи;
- для операторів комп'ютерного набору - тривалістю 10 хвилин, після кожної години роботи.

Всі вище зазначені правила виконуються на підприємстві. Все обладнання працює так, щоб створити комфортні умови для праці і при цьому не наражатися на будь-яку небезпеку.

Робота програмістом, передбачає сидячий спосіб життя.. Для запобігання втомі рук потрібно використовувати зручні мишу та клавіатуру. Треба тримати спину прямо

Можна виділити 3 правила збереження зору при тривалій роботі за комп'ютером:

Робити невелику перерву кожні півгодини.

Намагатися виконувати вправи для очей – вони справді допомагають.

Налаштувати монітор так, щоб його яскравість та контрастність були комфортними для зору працівника .

3.2.6 Електробезпека

Приміщення (окрім тих, де розташовуються сервери) мають бути оснащені системою автоматичної пожежної сигналізації та вогнегасниками. Під час монтажу та експлуатації ліній електромережі необхідно повністю унеможливити виникнення електричного джерела загоряння внаслідок короткого замикання та перевантаження проводів, обмежувати застосування проводів з легкозаймистою ізоляцією і, за можливості, застосовувати негорючу ізоляцію. У приміщенні, де одночасно експлуатуються понад п'ять комп’ютерів, на помітному та доступному місці встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення.

3.3 Пожежна безпека.

З метою попередження виникнення пожежі в приміщенні виконується ряд організаційних заходів:

- розміщення кабелів живлення поза зонами переміщення персоналу;
- легкогорючі матеріали розміщуються на відстані від кабелів живлення;

- виконання робіт з модифікації мережної проводки проводиться відповідальними за виконання подібних робіт людьми.

Для гасіння пожеж на початкових стадіях широко застосовуються вогнегасники.

Газові вогнегасники застосовуються для гасіння рідких і твердих речовин, а також електроустановок, що знаходяться під напругою.

У виробничих приміщеннях з ЕОМ застосовуються головним чином вуглекислотні вогнегасники, перевагою яких є висока ефективність гасіння пожежі, збереження електронного устаткування, діелектричні властивості вуглекислого газу, що дозволяє використовувати ці вогнегасники навіть у тому випадку, коли не вдається знеструмити електроустановку відразу.

ВИСНОВКИ

В ході представленої роботи було досліджено місце інформаційно-орієнтованих систем в сьогоднішньому світі. При розробці системи було розглянуто існуючі інтернет рушії – клієнт-серверна розробка. Було виявлено найбільш найпопулярніші та найбільш потужні рушії, якими стали .Net та C#. Було проведено їх порівняння і обрані найбільш актуальні засоби розробки для веб-ресурсыв а також було проаналізовано кожен із них, чим вони кращі та які в них недоліки. При аналізі можливостей кожного з ігрових рушіїв було створено проект «Шахова гра», відповідно до поставленої задачі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Martin L. Abbott. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise. –Addison-Wesley Professional, 2015.–624 с.

2. Vikram Murugesan, Microservices Deployment Cookbook. –Packt Publishing, 2017. –378 с.

3. Sam Newman. Building Microservices: Designing Fine-Grained Systems. –O'Reilly Media, 2015. –280 с.

4. Sourabh Sharma, Rajesh RV, David Gonzalez. Microservices: Building Scalable Software. –Packt Publishing Limited, 2017. –919 с.

5. Rajesh RV, Spring Microservices. –Packt Publishing, 2016. –436 с.

6. Pethuru Raj. Docker: Creating Structured Containers. –Packt Publishing, 2016. –320 с.

7. Web Call Server 5: [Электронный ресурс]. URL: https://flashphoner.com/

8. 40 open source, free and top UML tools: [Электронный ресурс]. URL: https://www.predictiveanalyticstoday.com/open-source-free-unified-modeling-language-uml-tools/

9. Что такое ASP.NET: [Электронный ресурс]. URL: http://www.internet-technologies.ru/articles/lekciya-1-chto-takoe-asp-net-installyaciya-i-testovyy-proekt.html

10. Web Call Server - Руководство Разработчика: [Электронный ресурс]. URL: https://flashphoner.com/docs/wcs5/wcs_docs/html/ru/wcs-developer-guide-2/

11. Which is Best for Web Application Development—Dot Net, PHP, Python, Ruby, or Java: [Электронный ресурс]. URL: https://www.addonsolutions.com/blog/which-is-best-for-web-application-development-dot-net-php-python-ruby-or-java.html/ (Дата обращения 15.04.2018);