Київський національний торговельно-економічний університет
Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка інформаційної системи розподілу навантаження викладачів кафедри»

Студента 4 курсу, 10 групи,
спеціальності
122 «Комп'ютерні науки» Цопа Антон Миколайович

_____
підпис студента

Науковий керівник
доктор технічних наук, професор Краскевич Валерій Євгенович

_____
підпис керівника

Гарант освітньої програми
кандидат технічних наук, доцент Демідов Павло Георгійович

_____
підпис керівника

Київ 2021

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
Спеціальність 122 «Комп'ютерні науки»

Затверджую
Зав. кафедри _____Пурський О.І.
« » грудня 2020р.

Завдання
на випускну кваліфікаційну роботу студенту
Цопі Антону Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи

«Розробка інформаційної системи розподілу навантаження викладачів кафедри»

Затверджена наказом ректора від *«04» грудня 2020 р. № 4111*

2. Строк здачі студентом закінченої роботи *28 травня 2021 року*

3. Цільова установка та вихідні дані до роботи

Мета роботи: обґрунтування та розробка інформаційної системи розподілу навантаження викладачів кафедри з урахуванням сучасних тенденцій застосування мов програмування.

Об'єкт дослідження: процес розробка інформаційної системи розподілу навантаження викладачів кафедри

Предмет дослідження: засоби створення інформаційної системи.

4. Перелік графічного матеріалу_____
_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Краскевич В.Є.	22.12.2020 р.	22.12.2020 р.
2	Краскевич В.Є.	22.12.2020 р.	22.12.2020 р.
3	Краскевич В.Є.	22.12.2020 р.	22.12.2020 р.

6. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. Теоретичні методи створення системи розподілу навантаження викладачів кафедри

1.1 Поняття і зміст системи розподілу навантаження

1.2 Методи і особливості створення системи розподілу навантаження викладачів

Висновки до розділу

РОЗДІЛ 2. Математична модель створення інформаційної системи розподілу навантаження викладачів

2.1. Формування системи показників для створення інформаційної системи розподілу навантаження викладачів кафедри

2.2 Математична модель створення системи розподілу навантаження викладачів кафедри

Висновки до розділу

РОЗДІЛ 3. Розробка програмного забезпечення для розподілу навантаження викладачів кафедри

3.1. Інформаційно-логічна модель створення сайту

3.2. Програмно-апаратна реалізація інформаційної моделі додатку для розподілу навантаження викладачів кафедри

Висновки до розділу

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Календарний план виконання роботи

№ Пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми випускної кваліфікаційної роботи*	*05.10.2020*	*05.10.2020*
2	*Розробка та затвердження завдання на випускну кваліфікаційну роботу*	*18.12.2020*	*18.12.2020*
3	*Вступ*	*03.02.2021*	*03.02.2021*
4	*РОЗДІЛ 1 Теоретичні методи створення системи розподілу навантаження викладачів кафедри*	*26.02.2021*	*26.02.2021*
5	*РОЗДІЛ 2 Математична модель створення інформаційної системи розподілу навантаження викладачів*	*06.04.2021*	*06.04.2021*
6	*РОЗДІЛ 3 Розробка програмного забезпечення для розподілу навантаження викладачів кафедри*	*12.05.2021*	*12.05.2021*
7	*Висновки*	*14.05.2021*	*14.05.2021*
8	*Здача випускної кваліфікаційної роботи на кафедру науковому керівнику*	*20.05.2021*	*20.05.2021*
9	*Попередній захист випускної кваліфікаційної роботи*	*26.05.2021*	*26.05.2021*
11	*Виправлення зауважень, зовнішнє рецензування випускної кваліфікаційної роботи*	*27.05.2021*	
12	*Представлення готової зшитої випускної кваліфікаційної роботи на кафедру*	*31.05.2021*	
13	*Публічний захист випускної кваліфікаційної роботи*	*За розкладом роботи ЕК*	

8. Дата видачі завдання «15» грудня 2021 р.

9. Керівник випускної кваліфікаційної роботи

Краскевич В.Є.

(прізвище, ініціали, підпис)

10. Гарант освітньої програми Демідов П.Г.
(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент-дипломник
Цопа А.М.
(прізвище, ініціали, підпис)

12. Відгук керівника випускної кваліфікаційної роботи

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Керівник випускної кваліфікаційної роботи

_____ р.

(підпис, дата)

13. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента

_____

(прізвище, ініціали)

може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми_____Демідов П.Г.____

(підпис, прізвище, ініціали)

Завідувач кафедри _____Пурський О.І.____

(підпис, прізвище, ініціали)

«_____»_____2021 р.

АНОТАЦІЯ

Тема дипломної роботи: «Розробка інформаційної системи розподілу навантаження викладачів кафедри»

Виконав:

У випускному кваліфікаційному проекті розглядається питання створення системи системи розподілу навантаження викладачів кафедри. Дипломна робота складається з восьми частин, а саме зі: вступу; першого розділу, в якому розглядається і описуються теоретико-методологічні основи дослідження, поняття та зміст розподілу навантаження, методи і особливості системи розподілу навантаження викладачів кафедри; другого розділу, у якому проведено побудову моделі системи розподілу навантаження викладачів кафедри, проведено аналіз системи показників для розподілу навантаження викладачів кафедри, сформовано моделі розподілу навантаження викладачів кафедри; третього розділу, в якому відбувається вибір засобів програмної реалізації системи, її проектування за допомогою діаграм та, власне, розробка інформаційної системи; висновків, списку використаних джерел та додатків.

До ключових слів належать наступні: прикладне програмне забезпечення, C#, Visual Studio, розподіл навантаження.

SUMMARY

Thesis topic: "Development of information system for load distribution of teachers of the department"

Completed:

The final qualification project considers the issue of creating a system of load distribution system of teachers of the department. Thesis consists of eight parts, namely: introduction; the first section, which considers and describes the theoretical and methodological foundations of the study, the concept and content of load distribution, methods and features of the load distribution system of teachers of the department; the second section, in which the model of the load distribution system of the teachers of the department was built, the analysis of the system of indicators for the load distribution of the teachers of the department was carried out, the models of the load distribution of the teachers of the department were formed; the third section, in which there is a choice of means of software implementation of the system, its design using diagrams and, in fact, the development of the information system; conclusions, list of sources and appendices used.

Keywords include application software, C #, Visual Studio, load balancing.

ЗМІСТ

ВСТУП

В сучасному світі все частіше спостерігається тенденція автоматизації і переведення трудового процесу в онлайн формат, що обумовлено сучасними умовами взаємодії між людьми з однієї сторони і перевагами, що надає онлайн формат з іншої.

В усі сфери людського життя проникає поступова заміна або доповнення існуючих технологій різноманітними програмними рішеннями, які дозволяють спростити ті чи інші процеси робочого, соціального або економічного характеру. Такого роду технологічний процес пускає коріння в усі сфери діяльності сучасної людини, від освіти до побуту, від спілкування між керівником і підлеглим в державній структурі, до контролю навантаження викладачів кафедри.

Саме про останнє і піде мова в даній роботі. Автоматизація розподілу навантаження на викладачів кафедри призведе до:

- рівномірного навантаження;
- відсутності робочих перекосів;
- запобіганню емоційного вигорання.

Об’єктом дослідження є засоби і методи розробки інформаційних систем.

Предметом дослідження є створення інформаційних систем.

Виходячи з усього вищесказаного, важко недооцінити мету даної роботи, створення системи розподілу навантаження викладачів кафедри.

Для досягнення поставленої мети слід виконати наступні завдання:

- проаналізувати теоретичні методи створення системи розподілу навантаження викладачів кафедри;
- проаналізувати математичну модель;
- сформувати систему показників;

- обрати інструментарій розробки;
- розробити і описати систему.

Завдяки чіткому виконанню поставлених завдань в результаті буде отримана повноцінна система розподілу навантаження викладачів кафедри.

РОЗДІЛ 1. ТЕОРЕТИЧНІ МЕТОДИ СТВОРЕННЯ СИСТЕМИ РОЗПОДІЛУ НАВАНТАЖЕНЯН ВИКЛАДАЧІВ КАФЕДРИ

1.1. Поняття і зміст системи розподілу навантаження

Розподіл навантаження — це процес побудови оптимального розподілу завдань між співробітниками, спрямований на адекватне завантаження роботою кожного співробітника роботою, відповідно до його спеціалізації, кваліфікації, тощо.

Затриматися на роботі час від часу, коли є виробнича необхідність - не проблема для більшості людей. Так вважають 59%. Соціологи провели опитування і представили статистику:

- щодня допізна засиджуються в своїх кабінетах понад 25% персоналу;
- роблять це кілька разів на місяць і в тиждень - трохи більше 20%;
- в тому, що потрібно працювати ще і вдома, впевнені 8%.

Паралельно кожен третій заявляє, що з обов'язками слід вкладатися в законодавчий трудовий день. Лояльність до роботи понад графіком падає, як тільки вона стає вимогою, та ще не оплачується. Негативу додає такий показник: понад 60% службовців недосипають через стреси, пов'язаних з професійною діяльністю. Що безпосередньо знижує працездатність до 50%. І це самий явний мінус. До негативних наслідків переробки відносяться:

- складність контролю термінів виконання завдання, якщо роботу забирають додому;
- підвищена старанність може маскувати недостатність компетенцій;
- зниження мотивацій, моральне вигорання;
- фінансові втрати від спаду продуктивності.

Виходячи з усього вищесказаного, можна зробити висновок, що розподіл навантаження співробітників будь-якої структури є абсолютно необхідним явищем, на яке слід звертати підвищену увагу.

1.2. Методи і особливості створення системи розподілу навантаження викладачів

Особливості створення системи

Процес створення системи розподілу навантаження викладачів полягає в розробці інформаційної системи, яка буде зберігати інформацію щодо викладачів, предметів, груп студентів.

На основі наявних даних система буде допомагати створити розклад, виходячи з основних обмежень:

- Викладачі мають бути навантажені приблизно однаково
- Один викладач не може проводити 2 заняття одночасно
- Одна група студентів не може бути присутня на двох заняттях одночасно
- У групи не може бути більше 4-х пар в день
- У викладача не може бути більше 4-х пар в день

Інструментарій розробки системи. Мова програмування C#.

C # (вимовляється як музична нота C♯, але написаний зі знаком числа) - це універсальна багатопарадігмальна мова програмування, що включає строгу типізацію, лексичну область видимості, імператив, декларативний, функціональний, універсальний об'єкт -орієнтовані (на основі класів) та компонентно-орієнтовані дисципліни програмування. Він був розроблений Microsoft приблизно в 2000 році в рамках ініціативи .NET, а потім затверджений в якості міжнародного стандарту Ecma (ECMA-334) і ISO (ISO / IEC 23270: 2018). Mono - це назва безкоштовного проекту з відкритим вихідним кодом для

розробки компілятора і середовища виконання для мови. C # є одним з мов програмування, розроблених для Common Language Infrastructure (CLI).

C # був розроблений Андерсом Хейлсбергом, а його командою розробників в даний час керує Мадс Торгерсен. Остання версія - 8.0, випущена в 2019 році разом з Visual Studio 2019 версії 16.3.

Під час розробки .NET Framework бібліотеки класів спочатку були написані з використанням компілятора системи керованого коду під назвою «Simple Managed C» (SMC). У січні 1999 року Андерс Хейлсберг сформував команду для створення нової мови під назвою Cool, який позначав «C-подібна об'єктно-орієнтована мова». Microsoft розглядала збереження назви «Cool» в якості остаточної назви мови, але вирішила не робити цього з причин, пов'язаних з товарними знаками. До того часу, коли проект .NET був публічно оголошено на конференції професійних розробників в липні 2000 року, мову був перейменований в C #, а бібліотеки класів і середовище виконання ASP.NET були портований на C #.

Хейлсберг - головний дизайнер і головний архітектор C # в Microsoft, раніше займався проектуванням Turbo Pascal, Embarcadero Delphi (раніше CodeGear Delphi, Inprise Delphi і Borland Delphi) і Visual J ++. У своїх інтерв'ю і технічних статтях він заявив, що недоліки в більшості основних мов програмування (наприклад, C ++, Java, Delphi і Smalltalk) призвели до основ Common Language Runtime (CLR), що, в свою чергу, призвело до розробки сама мова C #.

Джеймс Гослінг, який створив мову програмування Java в 1994 році, і Білл Джой, співзасновник Sun Microsystems, творця Java, назвали C # «імітацією» Java; Далі Гослінг сказав, що «C # - це свого роду Java з надійністю, продуктивністю і безпекою видалений». Клаус Крефт і Анджеліка Лангер (автори книги про потоках C ++) заявили у своєму блозі, що «Java і C #» це

майже ідентичні мови програмування. Нудне повторення без інновацій », « Навряд чи хто-небудь буде стверджувати, що Java або C # є революційними мовами програмування, які змінили спосіб написання програм », а« C # багато запозичив у Java - і навпаки Тепер, коли C # підтримує упаковки і розпаковування, у нас буде дуже схожа функція в Java. "У липні 2000 року Хейлсберг сказав, що C #" не є клоном Java "і" набагато ближче до C ++ "по своєму дизайну.

З часу випуску C # 2.0 в листопаді 2005 року мови C # і Java розвивалися за все більш і більш розбіжним траєкторіях, ставши двома зовсім різними мовами. Одним з перших основних відхилень стало додавання узагальнень до обох мов з абсолютно різними реалізаціями. C # використовує reification для надання «першокласних» універсальних об'єктів, які можна використовувати як будь-який інший клас, з генерацією коду, що виконується під час завантаження класу. Крім того, в C #, додано кілька основних функцій для програмування функціонального стилю, кульмінацією яких є розширення LINQ, випущені в C # 3.0, і підтримуюча його структура лямбда-виразів, методів розширення і анонімних типів. Ці функції дозволяють програмістам C # використовувати методи функціонального програмування, такі як замикання, коли це вигідно для їх застосування. Розширення LINQ і функціональний імпорт допомагають розробникам скоротити обсяг стандартного коду, який включається в загальні завдання, такі як запити до бази даних, аналіз файлу XML або пошук в структурі даних, зміщуючи акцент на реальну логіку програми, щоб поліпшити читаність і ремонтопридатність.

По своєму дизайну C # є мовою програмування, який безпосередньо відображає основну інфраструктуру спільної мови (CLI). Більшість його внутрішніх типів відповідають типам значень, реалізованих середовищем CLI. Однак специфікація мови не містить вимог до генерації коду компілятора, тобто

не вказує, що компілятор C # має орієнтуватися на середовище виконання спільної мови, або генерувати загальний проміжний мова (CIL), або генерувати будь-який інший конкретний формат. Теоретично, компілятор C # може генерувати машинний код, як традиційні компілятори C ++ та Fortran.

C # підтримує строго типізовані оголошення неявних змінних з ключовим словом var, а також неявно типізовані масиви з ключовим словом new [], за яким слід ініціалізатор колекції.

C # підтримує суворий логічний тип даних, bool. Оператори, які приймають умови, такі як while і if, вимагають вирази типу, що реалізує оператор true, такого як логічний тип. Хоча C ++ також має логічний тип, він може вільно перетворюватися в цілі і з цілих чисел, а вирази, наприклад, якщо (a) вимагають тільки, щоб a був перетворений в bool, дозволяючи a бути int або покажчиком. C # забороняє цей підхід «цілочисельне значення істина або брехня» на тій підставі, що примус програмістів використовувати вирази, які повертають саме bool, може запобігти певні типи помилок програмування, таких як if (a = b) (використання присвоювання = замість рівності ==).

C # більш безпечний для типів, ніж C ++. Єдиними неявними перетвореннями за замовчуванням є ті, які вважаються безпечними, наприклад, розширення цілих чисел. Це застосовується під час компіляції, під час JIT і, в деяких випадках, під час виконання. Не відбувається неявного перетворення між логічними значеннями і цілими числами, а також між членами перерахування і цілими числами (за винятком літерала 0, який може бути неявно перетворений в будь-який перераховується тип). Будь кероване перетворення повинне бути чітко позначено як явне або неявне, на відміну від конструкторів копіювання C ++ та операторів перетворення, які за замовчуванням неявні.

C # має явну підтримку коваріації і контраваріантності в універсальних типах, на відміну від C ++, який має деяку ступінь підтримки контраваріантності просто завдяки семантиці повертаються типів віртуальних методи.

Учасники перерахування розміщуються у своїй області видимості.

Мова C # не допускає глобальних змінних або функцій. Всі методи і члени повинні бути оголошені всередині класів. Статичні члени відкритих класів можуть замінювати глобальні змінні і функції.

Локальні змінні не можуть приховувати змінні вміщує блоку, на відміну від C та C ++.

C # має підтримку строго типізованих покажчиків на функції через ключове слово делегат. Як і псевдо-C ++ фреймворк Qt, в C # є семантика, зокрема навколишнє події стилю публікації-підписки, хоча C # використовує для цього делегати.

Керована пам'ять не може бути звільнена; замість цього він автоматично збирається. Збірка сміття вирішує проблему витоків пам'яті, позбавляючи програміста від відповідальності за звільнення пам'яті, яка більше не потрібна.

На відміну від C ++, C # не підтримує множинне успадкування, хоча клас може реалізовувати будь-яку кількість інтерфейсів. Це було дизайнерське рішення провідного архітектора мови, щоб уникнути ускладнення і спростити архітектурні вимоги у всьому CLI. При реалізації декількох інтерфейсів, які містять метод з однаковою сигнатурою, і. тобто два методу з одним і тим же ім'ям, що приймають параметри одного й того ж типу в одному і тому ж порядку, C # дозволяє реалізовувати кожен метод в залежності від того, через який інтерфейс викликається цей метод, або, як Java, дозволяє реалізувати метод один раз і мати один виклик за викликом через будь-який з інтерфейсів класу.

Однак, на відміну від Java, C # підтримує перевантаження операторів. Тільки найбільш часто перевантажені оператори в C ++ можуть бути перевантажені в C #.

C # має можливість використовувати LINQ через .NET Framework. Розробник може запросити будь-який об'єкт IEnumerable <T>, документи XML, набір даних ADO.NET і бази даних SQL. [60] Використання LINQ в C # дає такі переваги, як підтримка Intellisense, потужні можливості фільтрації, безпека типів з можливістю перевірки помилок компіляції і узгодженість даних для запиту з різних джерел. Є кілька різних мовних структур, які можна використовувати з C # з LINQ, і вони є виразами запитів, лямбда-виразами, анонімними типами, неявно типізованими змінними, методами розширення і ініціалізаторами об'єктів

У серпні 2001 року корпорації Microsoft, Hewlett-Packard і Intel Corporation виступили співавторами в уявленні специфікації для C #, а також інфраструктури спільної мови (CLI) в організацію по стандартизації Ecma International. У грудні 2001 року ECMA випустила специфікацію мови ECMA-334 C #. C # став стандартом ISO у 2003 році (ISO / IEC 23270: 2003 Інформаційні технології. Мови програмування - C #). ECMA раніше прийняла еквівалентні специфікації як друге видання C # в грудні 2002 року.

У червні 2005 року ECMA схвалив видання 3 специфікації C # і оновило ECMA-334. Доповнення включали в себе часткові класи, анонімні методи, нульові типи і універсальні шаблони (щось схоже на шаблони C ++).

У липні 2005 року ECMA представила в ICO / МЕК JTC 1 за допомогою прискореного процесу останнього стандарти та відповідні ТЗ. Цей процес зазвичай займає 6-9 місяців.

Визначення мови C # і CLI стандартизовані у відповідності зі стандартами ISO і Ecma, які забезпечують розумну та недискримінаційну ліцензійну захист від патентних претензій.

Microsoft погодилася не пред'являти позов розробникам програмного забезпечення з відкритим вихідним кодом за порушення патентів в некомерційних проектах в частині структури, охоплюваній OSP. Microsoft також погодилася не застосовувати патенти на продукти Novell щодо платять клієнтів Novell, за винятком списку продуктів, в яких явно не згадується C#.NET або реалізація Novell .NET (The Mono Project). Тим не менш, Novell стверджує, що Mono не порушує жодних патентів Microsoft. Microsoft також уклала конкретну угоду про відмову у захисту патентних прав, пов'язаних з плагіном браузера Moonlight, який залежить від Mono, якщо він отриманий через Novell

Microsoft очолює розробку еталонного компілятор C # з відкритим вихідним кодом і набору інструментів, що раніше носили кодова назва "Roslyn". Компілятор, який повністю написаний на керованому коді (C #), був відкритий, а функціональність відображена як API. Це дозволяє розробникам створювати інструменти рефакторінгу та діагностики.

Інші компілятори C # (деякі з яких включають реалізацію інфраструктури спільної мови і бібліотек класів .NET):

• Проект Mono надає компілятор C # з відкритим вихідним кодом, повну реалізацію загальномовної інфраструктури з відкритим вихідним кодом, включаючи необхідні бібліотеки інфраструктури, як вони зазначені в специфікації ECMA, і майже повну реалізацію пропрієтарних бібліотек класів Microsoft .NET до .NET 3.5. Починаючи з Mono 2.6, планів по впровадженню WPF не існує; WF планується до більш пізнього випуску; і є лише часткові реалізації LINQ to SQL і WCF.

• Проект DotGNU (в даний час припинений) також надав компілятор C # з відкритим вихідним кодом, майже повну реалізацію інфраструктури спільної мови, включаючи необхідні бібліотеки інфраструктури, як вони зазначені в специфікації ECMA, і підмножина деяких залишилися пропрієтарних класів Microsoft .NET. бібліотеки .NET 2.0 (не документовані або не включені в специфікацію ECMA, але включені в стандартний дистрибутив Microsoft .NET Framework).• Загальна мовна інфраструктура загального ресурсу Microsoft під кодовою назвою «Rotor» забезпечує реалізацію спільного джерела середовища CLR і компілятора C #, ліцензованого тільки для освітніх і дослідницьких цілей, а також підмножина необхідних бібліотек інфраструктури Common Language Infrastructure в специфікації ECMA (вгору в C # 2.0 і підтримується тільки в Windows XP).

C# була обрана основною мовою розробки даного проекту з огляду на усе вищеописане, а саме — на функціонал, який підтримує дана мова, операційні системи, які підтримуються та простота вивчення.

Загалом — функціонал C# найкраще підходить під дану розробку, за рахунок простої реалізації основних принципів ООП та зрозумілого інтерфейсу.

Інструментарій розробки системи. Середовище розробки Visual Studio.

Microsoft Visual Studio - це інтегроване середовище розробки (IDE) від Microsoft. Він використовується для розробки комп'ютерних програм, а також веб-сайтів, веб-додатків, веб-служб та мобільних додатків. Visual Studio використовує платформи Microsoft для розробки програмного забезпечення, такі як API Windows, Windows Forms, Foundation Presentation Foundation, Windows Store та Microsoft Silverlight. Він може створювати як власний код, так і керований код.

Visual Studio включає редактор коду, що підтримує IntelliSense (компонент доповнення коду), а також рефакторинг коду. Інтегрований

налагоджувач працює як налагоджувач на рівні джерела, так і налагоджувач на рівні машини. Інші вбудовані інструменти включають кодовий профілер, конструктор для побудови програм GUI, веб-дизайнер, дизайнер класів та дизайнер схем бази даних. Він приймає плагіни, які покращують функціональність майже на кожному рівні - включаючи додавання підтримки для систем керування джерелами (наприклад, Subversion і Git) та додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для мов, що задаються доменом або набори інструментів для інших аспектів розробки програмного забезпечення життєвий цикл (як клієнт Azure DevOps: Team Explorer).

Visual Studio підтримує 36 різних мов програмування та дозволяє редактору коду та налагоджувачу підтримувати (в різній мірі) майже будь-яку мову програмування за умови існування послуги, що залежить від мови. Вбудовані мови включають C, C ++, C ++ / CLI, Visual Basic .NET, C #, F #, JavaScript, TypeScript, XML, XSLT, HTML та CSS. Підтримка інших мов, таких як Python, Ruby, Node.js та M серед інших, доступна через плагіни. Java (і J #) підтримувалися в минулому.

Найбільш основне видання Visual Studio, спільноти, доступне безкоштовно. Гасло для видання Visual Studio Community - "Безкоштовне повнофункціональне IDE для студентів, відкритих джерел та індивідуальних розробників".

На даний момент підтримується версія Visual Studio - 2019 рік.

Visual Studio не підтримує жодної мови програмування, рішення чи інструменту внутрішньо; натомість це дозволяє підключати функціональність, кодовану як VSPackage. Після встановлення функціональність доступна як Сервіс. IDE надає три послуги: SVsSolution, яка надає можливість перераховувати проекти та рішення; SVsUIShell, який забезпечує функцію вікон

та інтерфейсу користувача (включаючи вкладки, панелі інструментів та вікна інструментів); та SVsShell, яка займається реєстрацією VSPackages. Крім того, IDE також відповідає за координацію та забезпечення зв'язку між службами. Всі редактори, дизайнери, типи проектів та інші інструменти реалізовані як VSPackages. Visual Studio використовує COM для доступу до VSPackages. SDK Visual Studio також включає в себе керовану рамку пакетів (MPF), яка є набором керованих обгорток навколо COM-інтерфейсів, які дозволяють писати пакети будь-якою мовою, сумісною з CLI. Однак MPF не забезпечує всю функціональність, що піддається інтерфейсам Visual Studio COM. Потім послуги можуть бути використані для створення інших пакетів, які додають функціональність Visual Studio IDE.

Підтримка мов програмування додається за допомогою спеціального VSPackage, який називається Language Service. Мовна служба визначає різні інтерфейси, які може реалізувати реалізація VSPackage, щоб додати підтримку різних функціональних можливостей. Функції, які можна додати таким чином, включають забарвлення синтаксису, завершення оператора, відповідність дужок, підказки інформації про параметри, списки членів та маркери помилок для складання фону. Якщо інтерфейс буде реалізований, функціонал буде доступний для мови. Мовні послуги реалізуються на мовній основі. Реалізації можуть повторно використовувати код з аналізатора або компілятора для мови. Мовні сервіси можуть бути реалізовані як у рідному коді, так і керованому коді. Для нативного коду можуть бути використані або рідні інтерфейси COM, або Babel Framework (частина Visual Studio SDK). Для керованого коду MPF включає обгортки для написання сервісів керованої мови.

Visual Studio не включає вбудовану підтримку управління джерелами, але вона визначає два альтернативних способи інтеграції систем управління джерелами з IDE. VSPackage управління джерелом може забезпечити власний

індивідуальний інтерфейс користувача. Навпаки, плагін управління джерелом за допомогою MSSCCI (Microsoft Source Code Control Interface) надає набір функцій, які використовуються для реалізації різних функцій управління джерелом, зі стандартним інтерфейсом користувача Visual Studio. MSSCCI вперше використовувався для інтеграції Visual SourceSafe з Visual Studio 6.0, але згодом був відкритий через SDK Visual Studio. Visual Studio .NET 2002 використовував MSSCCI 1.1, а Visual Studio .NET 2003 використовували MSSCCI 1.2. Visual Studio 2005, 2008 та 2010 використовує MSSCCI версії 1.3, яка додає підтримку перейменування та видалення розповсюдження, а також асинхронного відкриття.

Visual Studio підтримує запуск декількох екземплярів середовища (кожен зі своїм набором VSPackages). Екземпляри використовують різні вулики реєстру (див. Визначення MSDN терміна "вулик реєстру" у сенсі, що використовується тут) для зберігання конфігураційного стану та диференціюються їх AppId (ідентифікатор програми). Екземпляри запускаються специфічним для AppId .exe, який вибирає AppId, встановлює кореневий вулик та запускає IDE. VSP-пакети, зареєстровані для одного AppId, інтегруються з іншими VSP-пакетами для цього AppId. Різні випуски продуктів Visual Studio створені за допомогою різних додатків. Продукти випуску Visual Studio Express встановлюються разом із власними AppIds, але продукти Standard, Professional та Team Suite мають однаковий AppId. Отже, видання Express можна встановити поряд з іншими виданнями, на відміну від інших видань, які оновлюють ту саму установку.

Професійне видання включає в себе набір VSPackages у стандартному виданні, а командний набір включає суперсет VSPackages в обох інших виданнях. Система AppId використовується Visual Studio Shell в Visual Studio 2008.

Дане середовище розробки було обрано по декількох критеріях:

- Підтримка мови C#
- Інтуїтивно зрозумілий інтерфейс
- Можливості графічного редактору, підходящі до задач

Так як усі критерії задовільнено — середовищем розробки було обрано саме Visual Studio 2019.

РОЗДІЛ 2. МОДЕЛЬ СТВОРЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ РОЗПОДІЛУ НАВАНТАЖЕННЯ ВИКЛАДАЧІВ

2.1. Формування системи показників для створення інформаційної системи розподілу навантаження викладачів кафедри

При розробці інформаційної системи розподілу навантаження викладачів кафедри є ряд критеріїв і показників, на які слід звернути особливу увагу.

- відповідність ступеня завантаженості працівників оптимальному значенню поточних обсягів робіт;
- розвиток спектру компетенцій у наявних співробітників або розширення штату за рахунок працевлаштування нових фахівців, що мають вузькоспрямовані, але затребувані на ринку професійні знання і досвід.

На сьогоднішній день практика вирішення даної проблематики полягає в приблизному підході при розподілі навантаження між співробітниками і відповідної оптимізації кадрового складу, що базується на емпіричному досвіді керівництва.

2.2. Модель створення системи розподілу навантаження викладачів кафедри

У даному розділі пропонується математично обґрунтований механізм перерозподілу навантаження між фахівцями кафедри в залежності від набору їх компетенцій і обсягу що надходить потоку різнотипних завдань.

При розгляді запропонованого методу вирішення поставленого вище завдання приймемо такі припущення:

- Обсяг робіт, достатній для обробки
- Спектр компетенцій, якими володіють фахівці кафедри, достатній для виконання завдань, поставлених керівництвом.

Нехай для певної кафедри є набір з n видів робіт, що мають обсяги A1, A2, ..., An (в розрахунку на робочий тиждень або робочий місяць), для реалізації яких є m співробітників, для кожного з яких унормовано їх загальний робочий час B1 , B2, ..., Bn (також в розрахунку на робочий тиждень або робочий місяць). Крім того, відомо, що кожен із співробітників володіє певним набором компетенцій, тобто набором видів робіт, які він здатний виконувати.

Потрібно скласти алгоритм для оптимізації розподілу видів робіт по кожному із співробітників для наступних випадків:

- номінальної завантаження (штатного режиму співробітників, а також штатного потоку по всіх видах робіт);
- завантаження в разі вибуття з виробничого процесу будь-якого (або яких-небудь) із співробітників;

Для розроблюваної системи алгоритм полягає в поступовому розподілі робіт між викладачами, в залежності від спеціалізації, при умові постійного контролю зайнятості викладача в даний конкретний момент і контролі фінального навантаження на всіх викладачів, з виявлення перевантажених викладачів і викладачів, навантаження яких недостатнє на фоні їх колег по кафедрі.

2.3. Інформаційно-логічна модель створення сайту

2.3.1. Діаграма варіантів використання

Діаграма використання найпростіша - це представлення взаємодії користувача із системою, яка показує взаємозв'язок між користувачем та різними випадками використання, в яких користувач бере участь. Діаграма випадків використання може ідентифікувати різні типи користувачів системи та різні випадки використання, і часто вона супроводжується також іншими типами діаграм. Варіанти використання представлені кругами або еліпсами.

Незважаючи на те, що сам випадок використання може детально вивчити кожну можливість, діаграма прикладів використання може допомогти забезпечити огляд системи на більш високому рівні. Раніше вже було сказано, що "схеми використання - це принципи вашої системи".

Через їх спрощений характер, схеми використання можуть бути хорошим інструментом комунікації для зацікавлених сторін. Креслення намагаються імітувати реальний світ і дають зацікавленій стороні уявлення про те, як буде розроблена система. Сіау та Лі провели дослідження, щоб визначити, чи взагалі існувала дійсна ситуація для схем використання або вони були непотрібними. Було виявлено, що діаграми випадків використання передають намір системи більш спрощеним чином зацікавленим сторонам і що вони "інтерпретуються більш повно, ніж діаграми класів".

Метою діаграми використання є відображення динамічного аспекту системи. Додаткові схеми та документація можуть бути використані для забезпечення повного функціонального та технічного уявлення про систему. Вони забезпечують спрощене та графічне представлення того, що система насправді повинна робити.

Елементи:

- рамки системи (англ. system border) - прямоугольник із назвою у верхніх частинах та еліпсами (прецедентами) всередині. Часто може бути опущено без корисної інформації про полезну інформацію,
- актор (англ. актор) - стилізований людський персонаж, обозначаючий набір ролей користувача (розуміється в широкому змісті: людина, внешняя сутність, клас, інша система), взаємодіючого з деякою сутністю (системною, підсистемою, класом). Актори не можуть бути пов'язані між собою з іншим (за вимкнення відносин щодо обробки / дослідження),
- прецедент - еліпс із надписом, що обозначає виконувану систематичну дію (може включати можливі варіанти), що призводить до спостерігаємих акторами результатів. Надпис може бути ім'ям або описом (з точки зору актора) того, "що" робить система (а не "як"). Ім прецедента зв’язано з неперервним (атомарним) сценарієм - конкретною послідовністю дій, ілюстріруючою повіддю. Під час сценарію актори обмінюються із систематичними повідомленнями. Сценарій може бути приведений на діаграмі прецедентів у відео UML-коментарі. З одним прецедентом може бути пов'язано кілька різних сценаріїв

На рисунку 2.1 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

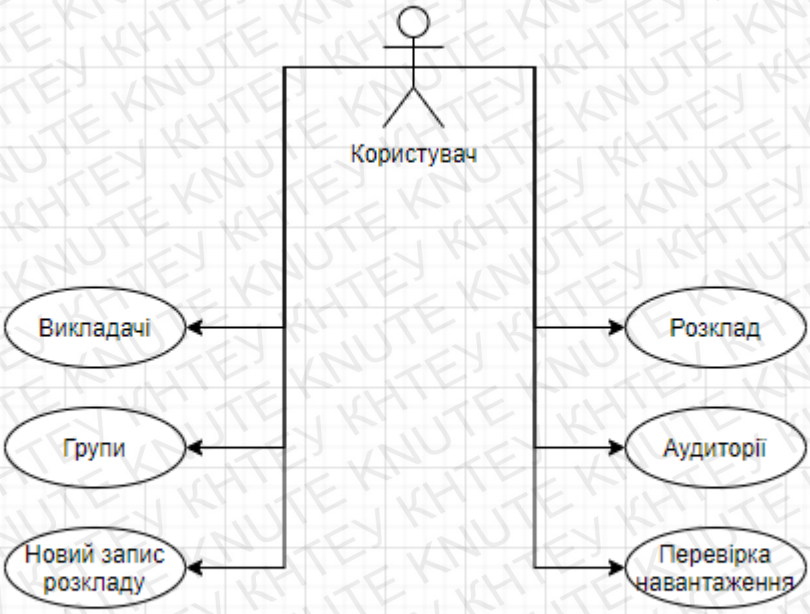

Рис. 2.1 — Діаграма варіантів використання

2.3.2. Діаграма класів

У програмній інженерії діаграма класів в Уніфікованій мові моделювання (UML) - це тип статичної структурної діаграми, що описує структуру системи, показуючи класи системи, їх атрибути, операції (або методи) та взаємозв'язки між об'єктами.

Діаграма класів є основним будівельним елементом об’єктно-орієнтованого моделювання. Він використовується для загального концептуального моделювання структури програми та для детального моделювання переведення моделей у програмовий код. Діаграми класів також можуть бути використані для моделювання даних. Класи на діаграмі класів представляють як основні елементи, взаємодії в програмі, так і класи, що програмуються.

На схемі класи представлені вікнами, які містять три відділення:

- У верхньому відділенні міститься назва класу. Надруковано жирним шрифтом і відцентровано, а перша літера написана великими літерами.
- Середній відсік містить атрибути класу. Вони вирівняні за лівим краєм, а перша буква мала.
- У нижньому відділенні містяться операції, які може виконувати клас. Вони також вирівняні за лівим краєм, а перша буква - мала.

При проектуванні системи ряд класів ідентифікується та згруповується у схему класів, яка допомагає визначити статичні відносини між ними. При детальному моделюванні класи концептуального проекту часто поділяються на ряд підкласів.

Залежність - це семантичний зв’язок між залежними та незалежними елементами моделі. Він існує між двома елементами, якщо зміни у визначенні одного елемента (сервера або цілі) можуть спричинити зміни для іншого (клієнта або джерела). Ця асоціація є односпрямованою. Залежність відображається у вигляді штрихової лінії з відкритою стрілкою, яка вказує від клієнта до постачальника.

Для подальшого опису поведінки систем ці діаграми класів можуть бути доповнені діаграмою стану або машиною стану UML.

Асоціація представляє родину посилань. Двійкова асоціація (з двома кінцями) зазвичай представляється у вигляді рядка. Асоціація може пов’язувати будь-яку кількість класів. Асоціація з трьома ланками називається потрійною асоціацією. Асоціацію можна назвати, а кінці асоціації можна прикрасити іменами ролей, показниками власності, кратністю, видимістю та іншими властивостями.

Існує чотири різні типи асоціацій: двонаправлена, односпрямована, агрегаційна (включає агрегацію композиції) та рефлексивна. Двонаправлені та односпрямовані асоціації є найбільш поширеними.

Наприклад, клас польоту асоціюється з класом літака двонаправлено. Асоціація представляє статичне відношення, яке ділиться між об'єктами двох класів.

Агрегація є варіантом взаємозв'язку "має"; агрегація є більш конкретною, ніж асоціація. Це асоціація, яка представляє частково цілі або часткові стосунки. Як показано на зображенні, професор "має" клас для викладання. Як тип асоціації, агрегація може бути названа та мати ті самі прикраси, що і асоціація. Однак агрегація не може включати більше двох класів; це має бути бінарна асоціація. Крім того, навряд чи існує різниця між агрегаціями та асоціаціями під час реалізації, і діаграма може взагалі пропустити відносини агрегування. [7]

Агрегація може відбуватися, коли клас є колекцією або контейнером інших класів, але вміщені класи не мають сильної залежності життєвого циклу від контейнера. Вміст контейнера все ще існує, коли контейнер знищений.

В UML він графічно представлений у вигляді порожнистої форми ромба на вміщуючому класі одним рядком, що зв’язує його із вміщеним класом. Сукупність - це семантично розширений об’єкт, який у багатьох операціях трактується як одиниця, хоча фізично він складається з декількох менших об’єктів.

Приклад: Бібліотека та студенти. Тут студент може існувати без бібліотеки, зв'язок між студентом і бібліотекою є агрегацією.

Це вказує на те, що один із двох пов’язаних класів (підклас) вважається спеціалізованою формою іншого (супер тип), а суперклас - узагальненням підкласу. На практиці це означає, що будь-який екземпляр підтипу є також екземпляром суперкласу. Зразкове дерево узагальнень цієї форми зустрічається

в біологічній класифікації: людина - це підклас маймуни, який є підкласом ссавців тощо. Зв’язок найлегше зрозуміти за допомогою фрази „А - це В” (людина - це ссавець, ссавець - тварина).

Графічне представлення UML узагальнення - це форма порожнистого трикутника на кінці суперкласу рядка (або дерева рядків), що зв’язує його з одним або кількома підтипами.

Відносини узагальнення також відомі як спадщина або відносини "є".

Суперклас (базовий клас) у відносинах узагальнення також відомий як "батьківський", суперклас, базовий клас або базовий тип.

Підтип у відносинах спеціалізації також відомий як "дочірній", підклас, похідний клас, похідний тип, клас успадкування або тип успадкування.

Зверніть увагу, що ці стосунки нічим не схожі на біологічні стосунки батьків та дітей: використання цих термінів надзвичайно поширене, але може ввести в оману.

А - це тип В

Наприклад, "дуб - це тип дерева", "автомобіль - це тип транспортного засобу"

Узагальнення може бути показано лише на діаграмах класів та на діаграмах використання.

При моделюванні UML взаємозв'язок реалізації - це взаємозв'язок між двома елементами моделі, в яких один елемент моделі (клієнт) реалізує (реалізує або виконує) поведінку, яку вказує інший елемент моделі (постачальник).

Графічне представлення UML реалізації - це порожниста форма трикутника на кінці інтерфейсу штрихової лінії (або дерева рядків), яка з'єднує її з одним або кількома реалізаторами. Проста головка стрілки використовується на кінці інтерфейсу штрихової лінії, що з'єднує її з користувачами. У діаграмах

компонентів використовується графічна умова «м'яч і сокет» (реалізатори виставляють кульку або льодяник, тоді як користувачі показують сокет). Реалізації можна показати лише на діаграмах класів або компонентів. Реалізація - це взаємозв'язок між класами, інтерфейсами, компонентами та пакетами, що з'єднує елемент клієнта з елементом постачальника. Зв'язок реалізації між класами / компонентами та інтерфейсами показує, що клас / компонент реалізує операції, пропоновані інтерфейсом.

Залежність - це слабша форма зв'язку, яка вказує на те, що один клас залежить від іншого, оскільки він використовує його в певний момент часу. Один клас залежить від іншого, якщо незалежний клас є змінною параметра або локальною змінною методу залежного класу. Це відрізняється від асоціації, де атрибут залежного класу є екземпляром незалежного класу. Іноді відносини між двома класами дуже слабкі. Вони взагалі не реалізовані зі змінними-членами. Швидше вони можуть бути реалізовані як аргументи функції-члена.

інший. Ці відносини зазвичай описуються як "А має В" (у матері-кота є кошенята, у кошенят - мати-кішка).

Представлення UML асоціації - це лінія, що з'єднує два пов'язані класи. На кожному кінці рядка є додаткові позначення. Наприклад, ми можемо вказати, використовуючи наконечник стрілки, що загострений кінець видно з хвоста стрілки. Ми можемо вказати власність шляхом розміщення кульки, ролі, яку відіграють елементи цього кінця, вказавши ім'я ролі та множинність екземплярів цієї сутності (діапазон кількості об'єктів, які беруть участь в асоціації з точки зору іншого кінця).

Класи сутності моделюють довгоживучу інформацію, якою обробляє система, а іноді і поведінку, пов'язану з цією інформацією. Їх не слід ідентифікувати як таблиці баз даних чи інших сховищ даних.

Вони намальовані як кола з короткою лінією, прикріпленою до нижньої частини кола. Як варіант, їх можна намалювати як звичайні класи із позначенням стереотипу «сутність» над назвою класу.

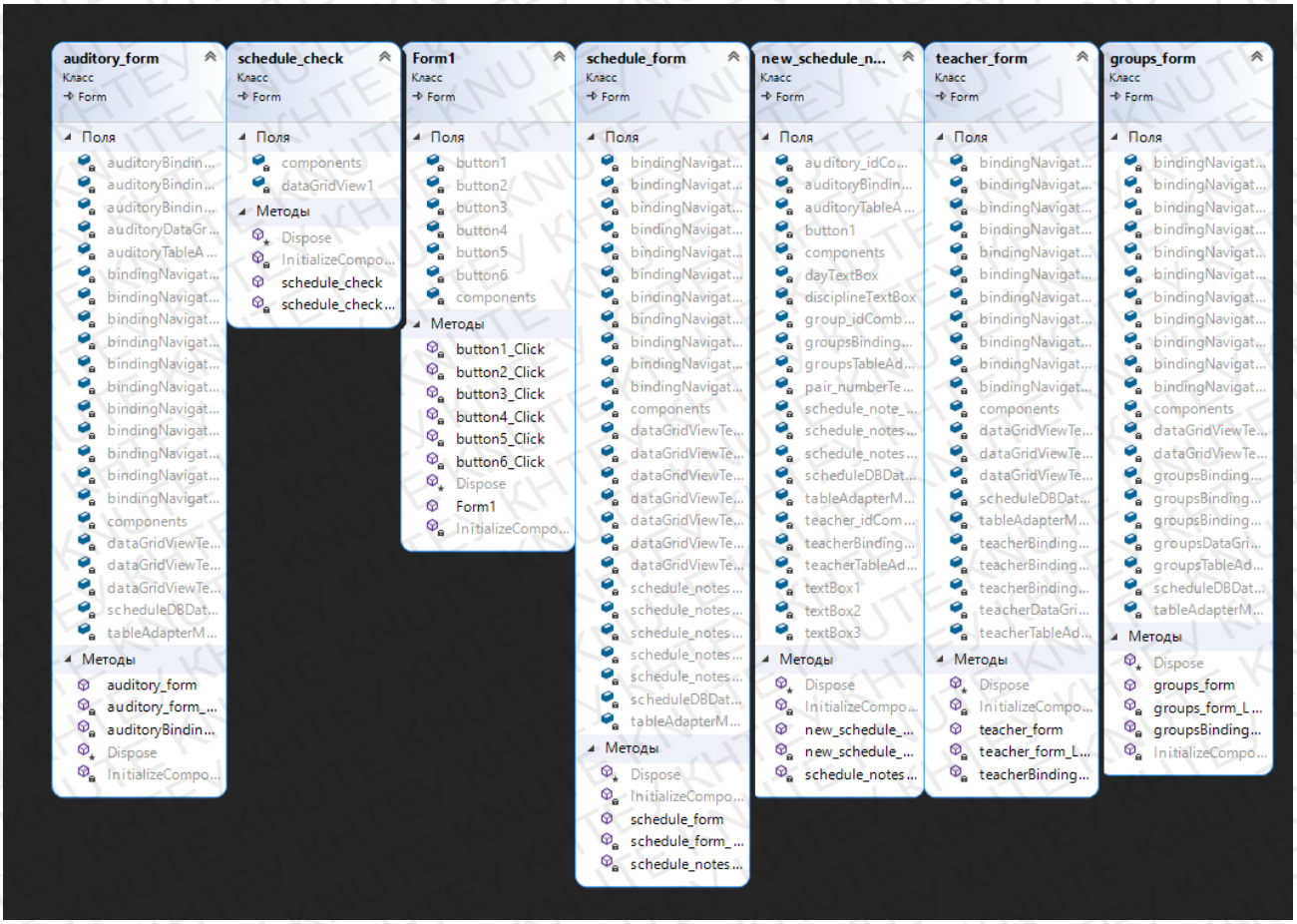

Рис. 2.2 — Діаграма класів

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗПОДІЛУ НАВАНТАЖЕНЯН ВИКЛАДАЧІВ КАФЕДРИ

3.1. Програмно-апаратна реалізація інформаційної моделі додатку для розподілу навантаження викладачів кафедри

3.1.1. Вибір мови програмування

C # (вимовляється як музична нота C♯, але написаний зі знаком числа) - це універсальна багатопарадігмальна мова програмування, що включає строгу типізацію, лексичну область видимості, імператив, декларативний, функціональний, універсальний об'єкт -орієнтовані (на основі класів) та компонентно-орієнтовані дисципліни програмування. Він був розроблений Microsoft приблизно в 2000 році в рамках ініціативи .NET, а потім затверджений в якості міжнародного стандарту Ecma (ECMA-334) і ISO (ISO / IEC 23270: 2018). Mono - це назва безкоштовного проекту з відкритим вихідним кодом для розробки компілятора і середовища виконання для мови. C # є одним з мов програмування, розроблених для Common Language Infrastructure (CLI).

C # був розроблений Андерсом Хейлсбергом, а його командою розробників в даний час керує Мадс Торгерсен. Остання версія - 8.0, випущена в 2019 році разом з Visual Studio 2019 версії 16.3.

Під час розробки .NET Framework бібліотеки класів спочатку були написані з використанням компілятора системи керованого коду під назвою «Simple Managed C» (SMC). У січні 1999 року Андерс Хейлсберг сформував команду для створення нової мови під назвою Cool, який позначав «C-подібна об'єктно-орієнтована мова». Microsoft розглядала збереження назви «Cool» в якості остаточної назви мови, але вирішила не робити цього з причин,

пов'язаних з товарними знаками. До того часу, коли проект .NET був публічно оголошено на конференції професійних розробників в липні 2000 року, мову був перейменований в C #, а бібліотеки класів і середовище виконання ASP.NET були портований на C #.

Хейлсберг - головний дизайнер і головний архітектор C # в Microsoft, раніше займався проектуванням Turbo Pascal, Embarcadero Delphi (раніше CodeGear Delphi, Inprise Delphi і Borland Delphi) і Visual J ++. У своїх інтерв'ю і технічних статтях він заявив, що недоліки в більшості основних мов програмування (наприклад, C ++, Java, Delphi і Smalltalk) призвели до основ Common Language Runtime (CLR), що, в свою чергу, призвело до розробки сама мова C #.

Джеймс Гослінг, який створив мову програмування Java в 1994 році, і Білл Джой, співзасновник Sun Microsystems, творця Java, назвали C # «імітацією» Java; Далі Гослінг сказав, що «C # - це свого роду Java з надійністю, продуктивністю і безпекою видалений». Клаус Крефт і Анджеліка Лангер (автори книги про потоках C ++) заявили у своєму блозі, що «Java і C #» це майже ідентичні мови програмування. Нудне повторення без інновацій », « Навряд чи хто-небудь буде стверджувати, що Java або C # є революційними мовами програмування, які змінили спосіб написання програм », а« C # багато запозичив у Java - і навпаки Тепер, коли C # підтримує упаковки і розпаковування, у нас буде дуже схожа функція в Java. "У липні 2000 року Хейлсберг сказав, що C #" не є клоном Java "і" набагато ближче до C ++ "по своєму дизайну.

З часу випуску C # 2.0 в листопаді 2005 року мови C # і Java розвивалися за все більш і більш розбіжним траєкторіях, ставши двома зовсім різними мовами. Одним з перших основних відхилень стало додавання узагальнень до обох мов з абсолютно різними реалізаціями. C # використовує reification для

надання «першокласних» універсальних об'єктів, які можна використовувати як будь-який інший клас, з генерацією коду, що виконується під час завантаження класу. Крім того, в C #, додано кілька основних функцій для програмування функціонального стилю, кульмінацією яких є розширення LINQ, випущені в C # 3.0, і підтримуюча його структура лямбда-виразів, методів розширення і анонімних типів. Ці функції дозволяють програмістам C # використовувати методи функціонального програмування, такі як замикання, коли це вигідно для їх застосування. Розширення LINQ і функціональний імпорт допомагають розробникам скоротити обсяг стандартного коду, який включається в загальні завдання, такі як запити до бази даних, аналіз файлу XML або пошук в структурі даних, зміщуючи акцент на реальну логіку програми, щоб поліпшити читаність і ремонтопридатність.

По своєму дизайну C # є мовою програмування, який безпосередньо відображає основну інфраструктуру спільної мови (CLI). Більшість його внутрішніх типів відповідають типам значень, реалізованих середовищем CLI. Однак специфікація мови не містить вимог до генерації коду компілятора, тобто не вказує, що компілятор C # має орієнтуватися на середовище виконання спільної мови, або генерувати загальний проміжний мова (CIL), або генерувати будь-який інший конкретний формат. Теоретично, компілятор C # може генерувати машинний код, як традиційні компілятори C ++ та Fortran.

C # підтримує строго типізовані оголошення неявних змінних з ключовим словом var, а також неявно типізовані масиви з ключовим словом new [], за яким слід ініціалізатор колекції.

C # підтримує суворий логічний тип даних, bool. Оператори, які приймають умови, такі як while і if, вимагають вирази типу, що реалізує оператор true, такого як логічний тип. Хоча C ++ також має логічний тип, він може вільно перетворюватися в цілі і з цілих чисел, а вирази, наприклад, якщо

(a) вимагають тільки, щоб a був перетворений в bool, дозволяючи a бути int або покажчиком. C # забороняє цей підхід «цілочисельне значення істина або брехня» на тій підставі, що примус програмістів використовувати вирази, які повертають саме bool, може запобігти певні типи помилок програмування, таких як if (a = b) (використання присвоювання = замість рівності ==).

C # більш безпечний для типів, ніж C ++. Єдиними неявними перетвореннями за замовчуванням є ті, які вважаються безпечними, наприклад, розширення цілих чисел. Це застосовується під час компіляції, під час JIT і, в деяких випадках, під час виконання. Не відбувається неявного перетворення між логічними значеннями і цілими числами, а також між членами перерахування і цілими числами (за винятком літерала 0, який може бути неявно перетворений в будь-який перераховується тип). Будь кероване перетворення повинне бути чітко позначено як явне або неявне, на відміну від конструкторів копіювання C ++ та операторів перетворення, які за замовчуванням неявні.

C # має явну підтримку коваріації і контраваріантності в універсальних типах, на відміну від C ++, який має деяку ступінь підтримки контраваріантності просто завдяки семантиці повертаються типів віртуальних методи.

Учасники перерахування розміщуються у своїй області видимості.

Мова C # не допускає глобальних змінних або функцій. Всі методи і члени повинні бути оголошені всередині класів. Статичні члени відкритих класів можуть замінювати глобальні змінні і функції.

Локальні змінні не можуть приховувати змінні вміщує блоку, на відміну від C та C ++.

C # має підтримку строго типізованих покажчиків на функції через ключове слово делегат. Як і псевдо-C ++ фреймворк Qt, в C # є семантика,

зокрема навколишнє події стилю публікації-підписки, хоча C # використовує для цього делегати.

Керована пам'ять не може бути звільнена; замість цього він автоматично збирається. Збірка сміття вирішує проблему витоків пам'яті, позбавляючи програміста від відповідальності за звільнення пам'яті, яка більше не потрібна.

На відміну від C ++, C # не підтримує множинне успадкування, хоча клас може реалізовувати будь-яку кількість інтерфейсів. Це було дизайнерське рішення провідного архітектора мови, щоб уникнути ускладнення і спростити архітектурні вимоги у всьому CLI. При реалізації декількох інтерфейсів, які містять метод з однаковою сигнатурою, і. тобто два методу з одним і тим же ім'ям, що приймають параметри одного й того ж типу в одному і тому ж порядку, C # дозволяє реалізовувати кожен метод в залежності від того, через який інтерфейс викликається цей метод, або, як Java, дозволяє реалізувати метод один раз і мати один виклик за викликом через будь-який з інтерфейсів класу.

Однак, на відміну від Java, C # підтримує перевантаження операторів. Тільки найбільш часто перевантажені оператори в C ++ можуть бути перевантажені в C #.

C # має можливість використовувати LINQ через .NET Framework. Розробник може запросити будь-який об'єкт IEnumerable <T>, документи XML, набір даних ADO.NET і бази даних SQL. [60] Використання LINQ в C # дає такі переваги, як підтримка Intellisense, потужні можливості фільтрації, безпека типів з можливістю перевірки помилок компіляції і узгодженість даних для запиту з різних джерел. Є кілька різних мовних структур, які можна використовувати з C # з LINQ, і вони є виразами запитів, лямбда-виразами, анонімними типами, неявно типізованими змінними, методами розширення і ініціалізаторами об'єктів

У серпні 2001 року корпорації Microsoft, Hewlett-Packard і Intel Corporation виступили співавторами в уявленні специфікації для C #, а також інфраструктури спільної мови (CLI) в організацію по стандартизації Ecma International. У грудні 2001 року ECMA випустила специфікацію мови ECMA-334 C #. C # став стандартом ISO у 2003 році (ISO / IEC 23270: 2003 Інформаційні технології. Мови програмування - C #). ECMA раніше прийняла еквівалентні специфікації як друге видання C # в грудні 2002 року.

У червні 2005 року ECMA схвалив видання 3 специфікації C # і оновило ECMA-334. Доповнення включали в себе часткові класи, анонімні методи, нульові типи і універсальні шаблони (щось схоже на шаблони C ++).

У липні 2005 року ECMA представила в ICO / МЕК JTC 1 за допомогою прискореного процесу останнього стандарти та відповідні ТЗ. Цей процес зазвичай займає 6-9 місяців.

Визначення мови C # і CLI стандартизовані у відповідності зі стандартами ISO і Ecma, які забезпечують розумну та недискримінаційну ліцензійну захист від патентних претензій.

Microsoft погодилася не пред'являти позов розробникам програмного забезпечення з відкритим вихідним кодом за порушення патентів в некомерційних проектах в частині структури, охоплюваній OSP. Microsoft також погодилася не застосовувати патенти на продукти Novell щодо платять клієнтів Novell, за винятком списку продуктів, в яких явно не згадується C#.NET або реалізація Novell .NET (The Mono Project). Тим не менш, Novell стверджує, що Mono не порушує жодних патентів Microsoft. Microsoft також уклала конкретну угоду про відмову у захисту патентних прав, пов'язаних з плагіном браузера Moonlight, який залежить від Mono, якщо він отриманий через Novell

Microsoft очолює розробку еталонного компілятор C # з відкритим вихідним кодом і набору інструментів, що раніше носили кодова назва "Roslyn". Компілятор, який повністю написаний на керованому коді (C #), був відкритий, а функціональність відображена як API. Це дозволяє розробникам створювати інструменти рефакторінгу та діагностики.

Інші компілятори C # (деякі з яких включають реалізацію інфраструктури спільної мови і бібліотек класів .NET):

• Проект Mono надає компілятор C # з відкритим вихідним кодом, повну реалізацію загальномовної інфраструктури з відкритим вихідним кодом, включаючи необхідні бібліотеки інфраструктури, як вони зазначені в специфікації ECMA, і майже повну реалізацію пропрієтарних бібліотек класів Microsoft .NET до .NET 3.5. Починаючи з Mono 2.6, планів по впровадженню WPF не існує; WF планується до більш пізнього випуску; і є лише часткові реалізації LINQ to SQL і WCF.

• Проект DotGNU (в даний час припинений) також надав компілятор C # з відкритим вихідним кодом, майже повну реалізацію інфраструктури спільної мови, включаючи необхідні бібліотеки інфраструктури, як вони зазначені в специфікації ECMA, і підмножина деяких залишилися пропрієтарних класів Microsoft .NET. бібліотеки .NET 2.0 (не документовані або не включені в специфікацію ECMA, але включені в стандартний дистрибутив Microsoft .NET Framework).• Загальна мовна інфраструктура загального ресурсу Microsoft під кодовою назвою «Rotor» забезпечує реалізацію спільного джерела середовища CLR і компілятора C #, ліцензованого тільки для освітніх і дослідницьких цілей, а також підмножина необхідних бібліотек інфраструктури Common Language Infrastructure в специфікації ECMA (вгору в C # 2.0 і підтримується тільки в Windows XP).

C# була обрана основною мовою розробки даного проекту з огляду на усе вищеописане, а саме — на функціонал, який підтримує дана мова, операційні системи, які підтримуються та простота вивчення.

Загалом — функціонал C# найкраще підходить під дану розробку, за рахунок простої реалізації основних принципів ООП та зрозумілого інтерфейсу.

3.1.2. Вибір середовища розробки

Microsoft Visual Studio - це інтегроване середовище розробки (IDE) від Microsoft. Він використовується для розробки комп'ютерних програм, а також веб-сайтів, веб-додатків, веб-служб та мобільних додатків. Visual Studio використовує платформи Microsoft для розробки програмного забезпечення, такі як API Windows, Windows Forms, Foundation Presentation Foundation, Windows Store та Microsoft Silverlight. Він може створювати як власний код, так і керований код.

Visual Studio включає редактор коду, що підтримує IntelliSense (компонент доповнення коду), а також рефакторинг коду. Інтегрований налагоджувач працює як налагоджувач на рівні джерела, так і налагоджувач на рівні машини. Інші вбудовані інструменти включають кодовий профілер, конструктор для побудови програм GUI, веб-дизайнер, дизайнер класів та дизайнер схем бази даних. Він приймає плагіни, які покращують функціональність майже на кожному рівні - включаючи додавання підтримки для систем керування джерелами (наприклад, Subversion і Git) та додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для мов, що задаються доменом або набори інструментів для інших аспектів розробки програмного забезпечення життєвий цикл (як клієнт Azure DevOps: Team Explorer).

Visual Studio підтримує 36 різних мов програмування та дозволяє редактору коду та налагоджувачу підтримувати (в різній мірі) майже будь-яку мову програмування за умови існування послуги, що залежить від мови. Вбудовані мови включають C, C ++, C ++ / CLI, Visual Basic .NET, C #, F #, JavaScript, TypeScript, XML, XSLT, HTML та CSS. Підтримка інших мов, таких як Python, Ruby, Node.js та M серед інших, доступна через плагіни. Java (і J #) підтримувалися в минулому.

Найбільш основне видання Visual Studio, спільноти, доступне безкоштовно. Гасло для видання Visual Studio Community - "Безкоштовне повнофункціональне IDE для студентів, відкритих джерел та індивідуальних розробників".

На даний момент підтримується версія Visual Studio - 2019 рік.

Visual Studio не підтримує жодної мови програмування, рішення чи інструменту внутрішньо; натомість це дозволяє підключати функціональність, кодовану як VSPackage. Після встановлення функціональність доступна як Сервіс. IDE надає три послуги: SVsSolution, яка надає можливість перераховувати проекти та рішення; SVsUIShell, який забезпечує функцію вікон та інтерфейсу користувача (включаючи вкладки, панелі інструментів та вікна інструментів); та SVsShell, яка займається реєстрацією VSPackages. Крім того, IDE також відповідає за координацію та забезпечення зв'язку між службами. Всі редактори, дизайнери, типи проектів та інші інструменти реалізовані як VSPackages. Visual Studio використовує COM для доступу до VSPackages. SDK Visual Studio також включає в себе керовану рамку пакетів (MPF), яка є набором керованих обгорток навколо COM-інтерфейсів, які дозволяють писати пакети будь-якою мовою, сумісною з CLI. Однак MPF не забезпечує всю функціональність, що піддається інтерфейсам Visual Studio COM. Потім

послуги можуть бути використані для створення інших пакетів, які додають функціональність Visual Studio IDE.

Підтримка мов програмування додається за допомогою спеціального VSPackage, який називається Language Service. Мовна служба визначає різні інтерфейси, які може реалізувати реалізація VSPackage, щоб додати підтримку різних функціональних можливостей. Функції, які можна додати таким чином, включають забарвлення синтаксису, завершення оператора, відповідність дужок, підказки інформації про параметри, списки членів та маркери помилок для складання фону. Якщо інтерфейс буде реалізований, функціонал буде доступний для мови. Мовні послуги реалізуються на мовній основі. Реалізації можуть повторно використовувати код з аналізатора або компілятора для мови. Мовні сервіси можуть бути реалізовані як у рідному коді, так і керованому коді. Для нативного коду можуть бути використані або рідні інтерфейси COM, або Babel Framework (частина Visual Studio SDK). Для керованого коду MPF включає обгортки для написання сервісів керованої мови.

Visual Studio не включає вбудовану підтримку управління джерелами, але вона визначає два альтернативних способи інтеграції систем управління джерелами з IDE. VSPackage управління джерелом може забезпечити власний індивідуальний інтерфейс користувача. Навпаки, плагін управління джерелом за допомогою MSSCCI (Microsoft Source Code Control Interface) надає набір функцій, які використовуються для реалізації різних функцій управління джерелом, зі стандартним інтерфейсом користувача Visual Studio. MSSCCI вперше використовувався для інтеграції Visual SourceSafe з Visual Studio 6.0, але згодом був відкритий через SDK Visual Studio. Visual Studio .NET 2002 використовував MSSCCI 1.1, а Visual Studio .NET 2003 використовували MSSCCI 1.2. Visual Studio 2005, 2008 та 2010 використовує MSSCCI версії 1.3,

яка додає підтримку перейменування та видалення розповсюдження, а також асинхронного відкриття.

Visual Studio підтримує запуск декількох екземплярів середовища (кожен зі своїм набором VSPackages). Екземпляри використовують різні вулики реєстру (див. Визначення MSDN терміна "вулик реєстру" у сенсі, що використовується тут) для зберігання конфігураційного стану та диференціюються їх AppId (ідентифікатор програми). Екземпляри запускаються специфічним для AppId .exe, який вибирає AppId, встановлює кореневий вулик та запускає IDE. VSP-пакети, зареєстровані для одного AppId, інтегруються з іншими VSP-пакетами для цього AppId. Різні випуски продуктів Visual Studio створені за допомогою різних додатків. Продукти випуску Visual Studio Express встановлюються разом із власними AppIds, але продукти Standard, Professional та Team Suite мають однаковий AppId. Отже, видання Express можна встановити поряд з іншими виданнями, на відміну від інших видань, які оновлюють ту саму установку.

Професійне видання включає в себе набір VSPackages у стандартному виданні, а командний набір включає суперсет VSPackages в обох інших виданнях. Система AppId використовується Visual Studio Shell в Visual Studio 2008.

Дане середовище розробки було обрано по декількох критеріях:

- Підтримка мови C#
- Інтуїтивно зрозумілий інтерфейс
- Можливості графічного редактору, підходящі до задач

Так як усі критерії задовільнено — середовищем розробки було обрано саме Visual Studio 2019.

3.1.3. Графічний інтерфейс користувача

Графічний користувальницький інтерфейс - це форма користувальницького інтерфейсу, що дозволяє користувачам взаємодіяти з електронними пристроями за допомогою графічних піктограм та звукового індикатора, таких як як основне позначення, замість текстових користувальницьких інтерфейсів, набраних міток команд або текстової навігації. Графічні інтерфейси були введені у відповідь на сприйману круту криву навчання інтерфейсів командного рядка (CLI), які вимагають набору команд на клавіатурі комп’ютера.

Дії в графічному інтерфейсі зазвичай виконуються за допомогою прямого маніпулювання графічними елементами. Окрім комп'ютерів, графічні інтерфейси використовуються у багатьох портативних мобільних пристроях, таких як MP3-плеєри, портативні медіаплеєри, ігрові пристрої, смартфони та менші побутові, офісні та промислові елементи управління. Термін графічний інтерфейс, як правило, не застосовується до інших типів інтерфейсів із нижчою роздільною здатністю, таких як відеоігри (де переважно відображається дисплей HUD), або не включаючи плоскі екрани, такі як об'ємні дисплеї оскільки цей термін обмежений сферою двовимірних екранів, здатних описувати загальну інформацію, за традицією досліджень інформатики в Дослідницькому центрі Xerox Palo Alto.

Розробка візуальної композиції та часової поведінки графічного інтерфейсу є важливою частиною програмування програмного забезпечення в області взаємодії людина-комп'ютер. Його метою є підвищення ефективності та простоти використання основного логічного проектування збереженої програми, дисципліни проектування, що називається зручністю використання.

Методи дизайну, орієнтованого на користувача, використовуються для того, щоб візуальна мова, введена в дизайн, була добре адаптована до завдань.

Помітні особливості графічного інтерфейсу програми іноді називають хромом або графічним інтерфейсом. Як правило, користувачі взаємодіють з інформацією, маніпулюючи візуальними віджетами, що дозволяють здійснювати взаємодії, відповідні виду даних, який вони мають. Віджети добре продуманого інтерфейсу обрані для підтримки дій, необхідних для досягнення цілей користувачів. Модель-вигляд-контролер забезпечує гнучкі структури, в яких інтерфейс не залежить і опосередковано пов'язаний з функціями програми, тому графічний інтерфейс можна легко налаштувати. Це дозволяє користувачам за бажанням вибирати або розробляти різні обкладинки, а також полегшує роботу дизайнера по зміні інтерфейсу в міру розвитку потреб користувача. Хороший дизайн інтерфейсу стосується користувачів більше, а архітектури системи менше. Великі віджети, такі як вікна, зазвичай містять рамку або контейнер для основного вмісту презентації, наприклад веб-сторінки, електронного повідомлення чи малюнка. Менші, як правило, виступають інструментом введення користувачем.

Графічний інтерфейс може бути розроблений для потреб вертикального ринку як графічний користувальницький інтерфейс, специфічний для додатків. Приклади включають автоматичні касові автомати (банкомати), сенсорні екрани торгових точок (POS) у ресторанах, каси самообслуговування, що використовуються в роздрібному магазині, самостійний квиток і реєстрація авіакомпанії, інформаційні кіоски у громадському просторі, наприклад залізничний вокзал чи музей, а також монітори або екрани управління у вбудованому промисловому додатку, що використовує операційну систему реального часу (RTOS).

До 1980-х років стільникові телефони та портативні ігрові системи також використовували графічні інтерфейси із сенсорним екраном для додатків. Нові автомобілі використовують графічні інтерфейси в своїх навігаційних системах та мультимедійних центрах або комбінаціях навігаційних мультимедійних центрів.

Графічний інтерфейс розробленого додатку складається з:

- Форми головного меню (рис. 3.1)
- Форми викладачів (рис. 3.2)
- Форми груп (рис. 3.3)
- Форми аудиторій (рис. 3.4)
- Форми розкладу (рис. 3.5)
- Форми створення нового запису розкладу (рис. 3.6)
- Форми перевірки навантаження (рис. 3.7)

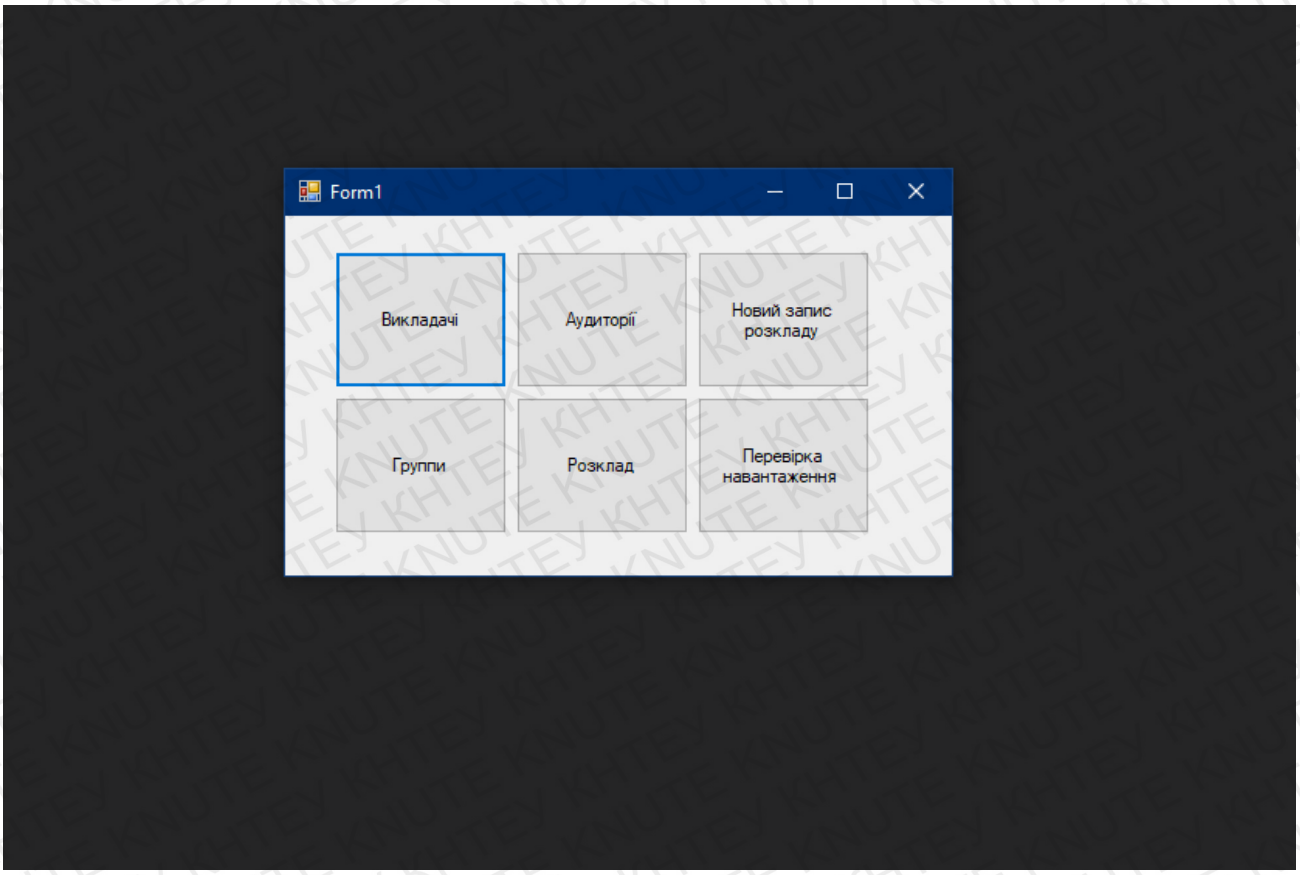

Рис. 3.1 — Форма головного меню

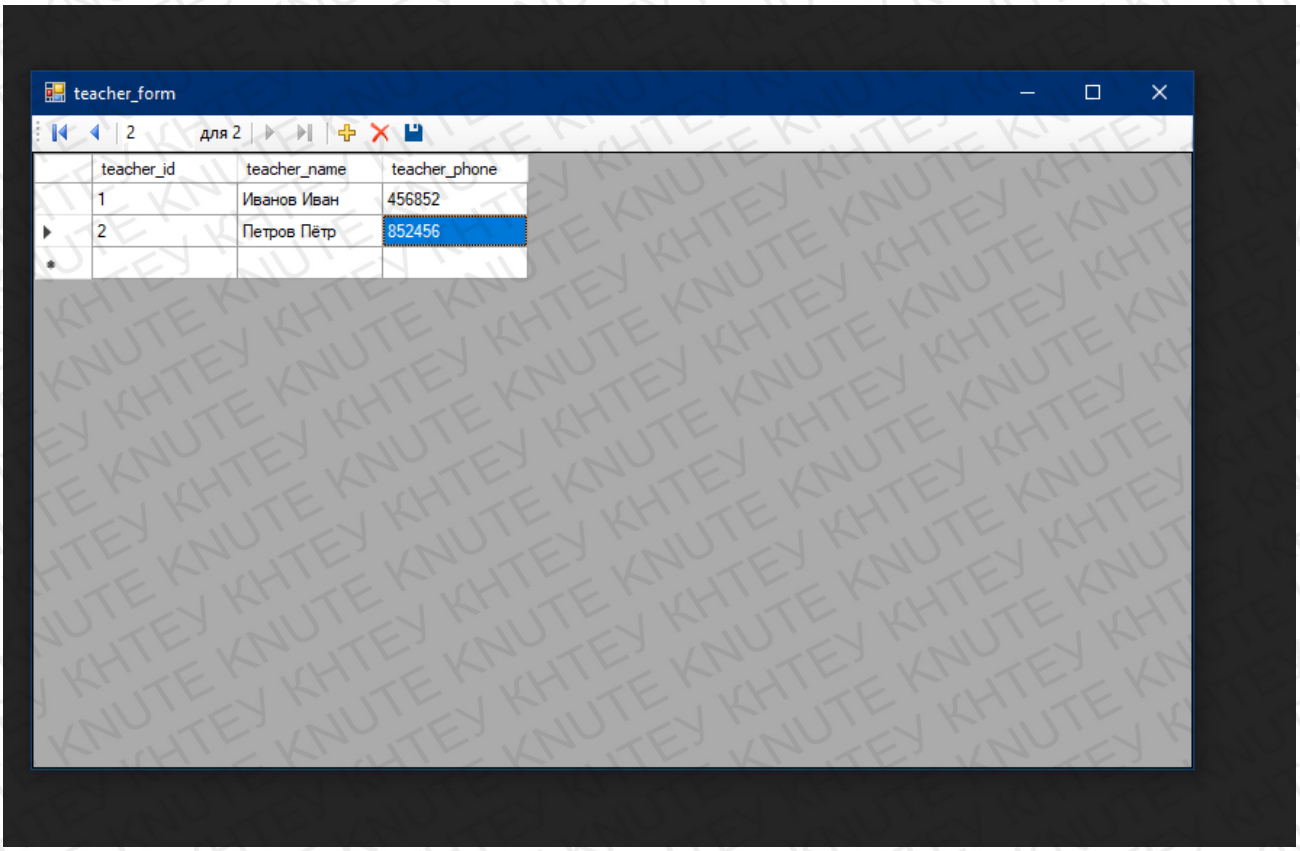

Рис. 3.2 — Форма викладачів

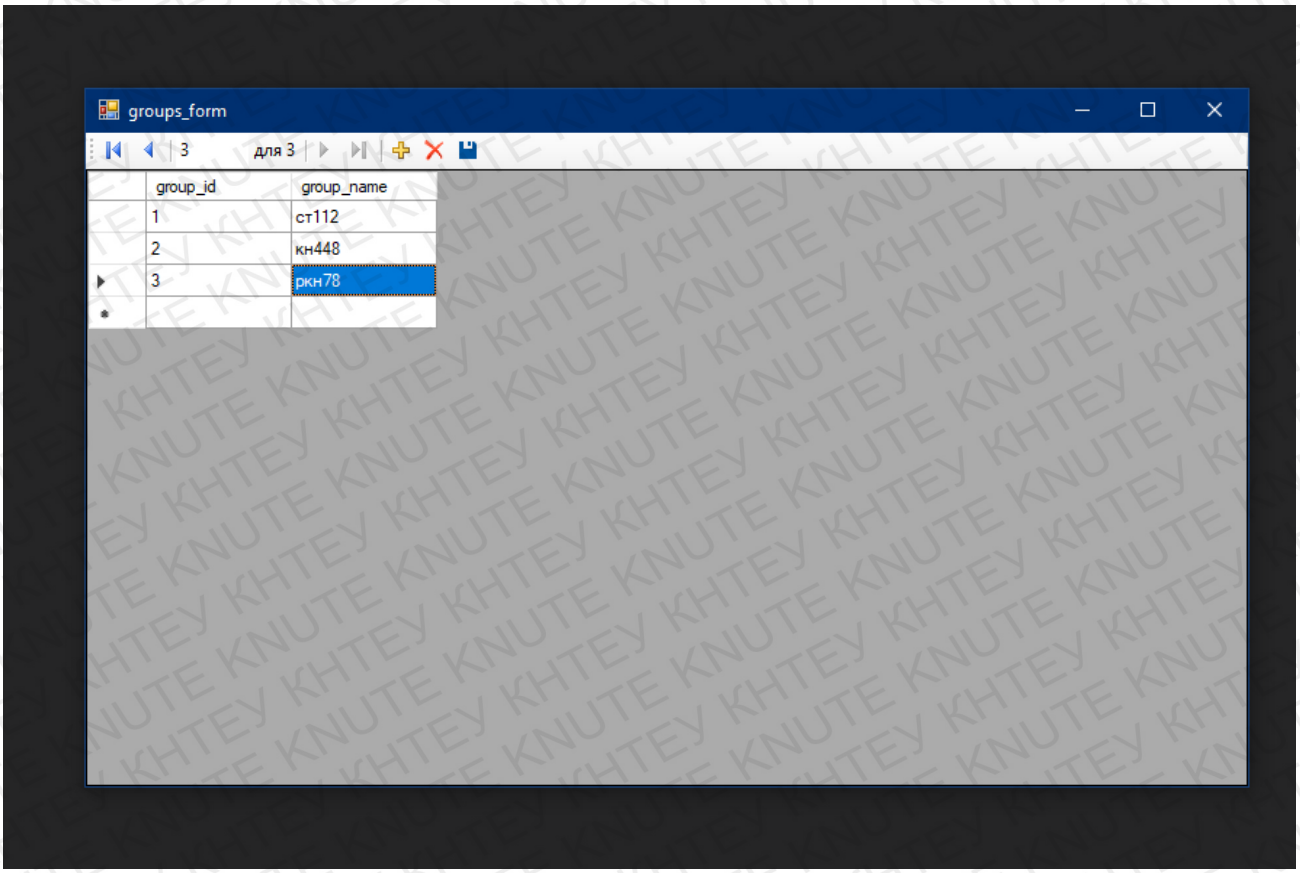

Рис. 3.3 — Форма груп

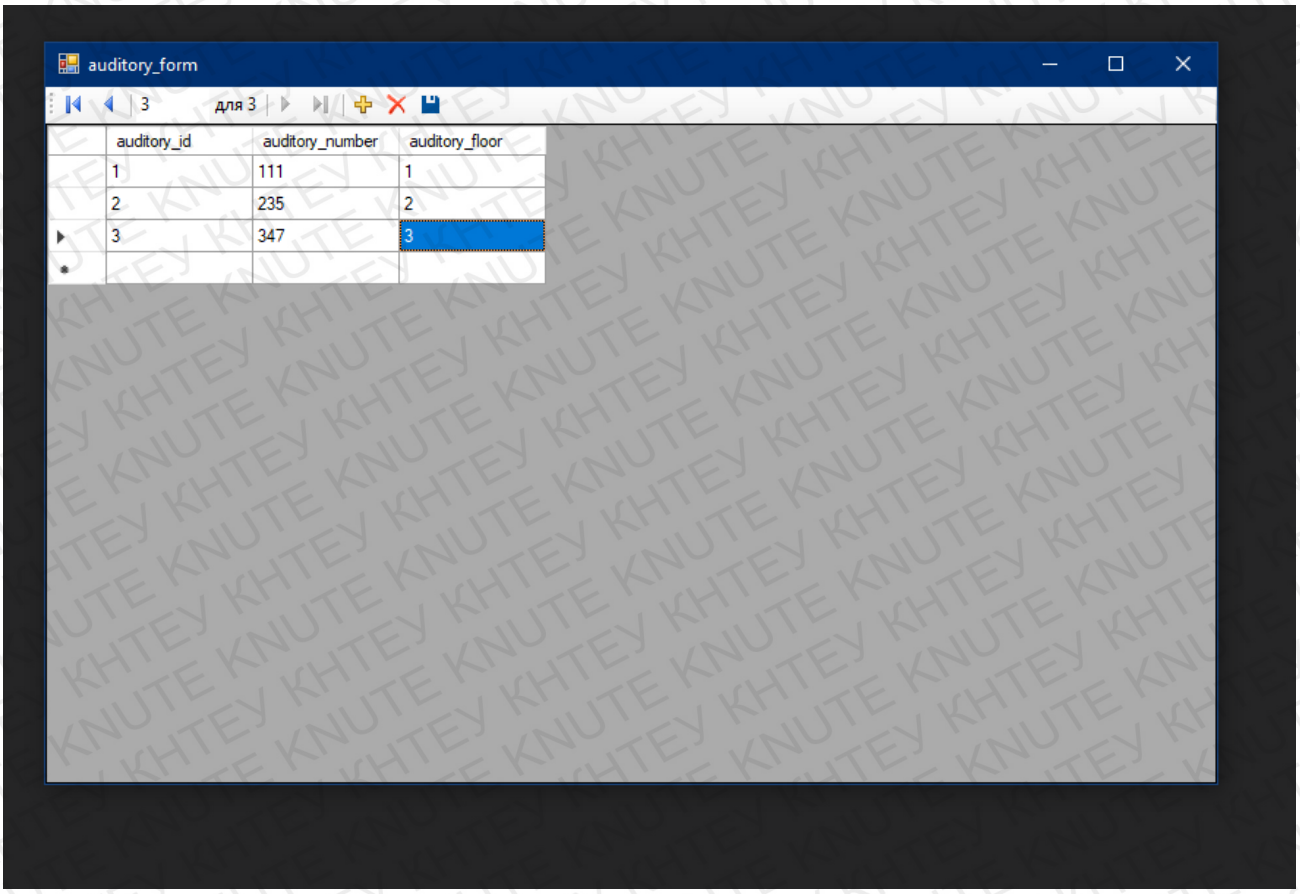

Рис. 3.4 — Форма груп

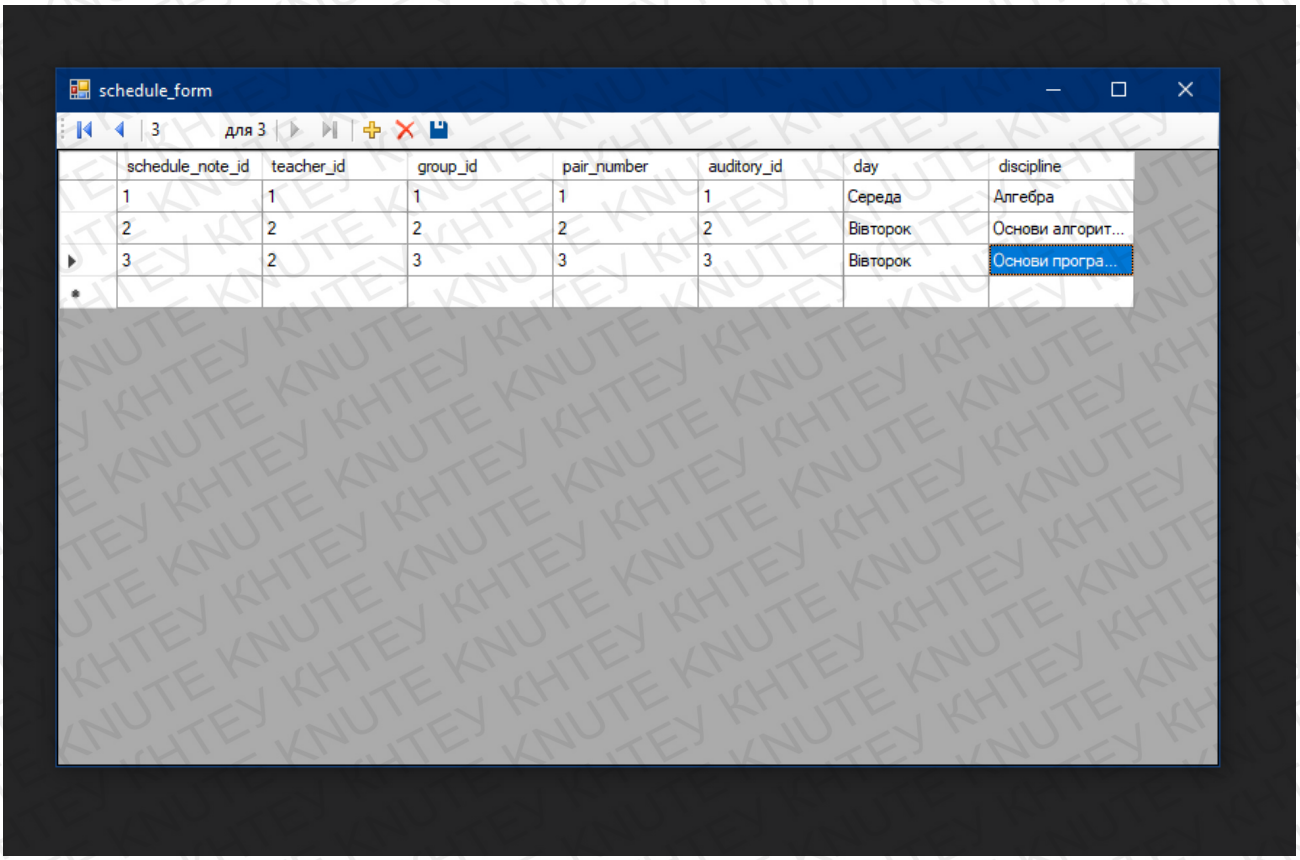

Рис. 3.5 — Форма розкладу

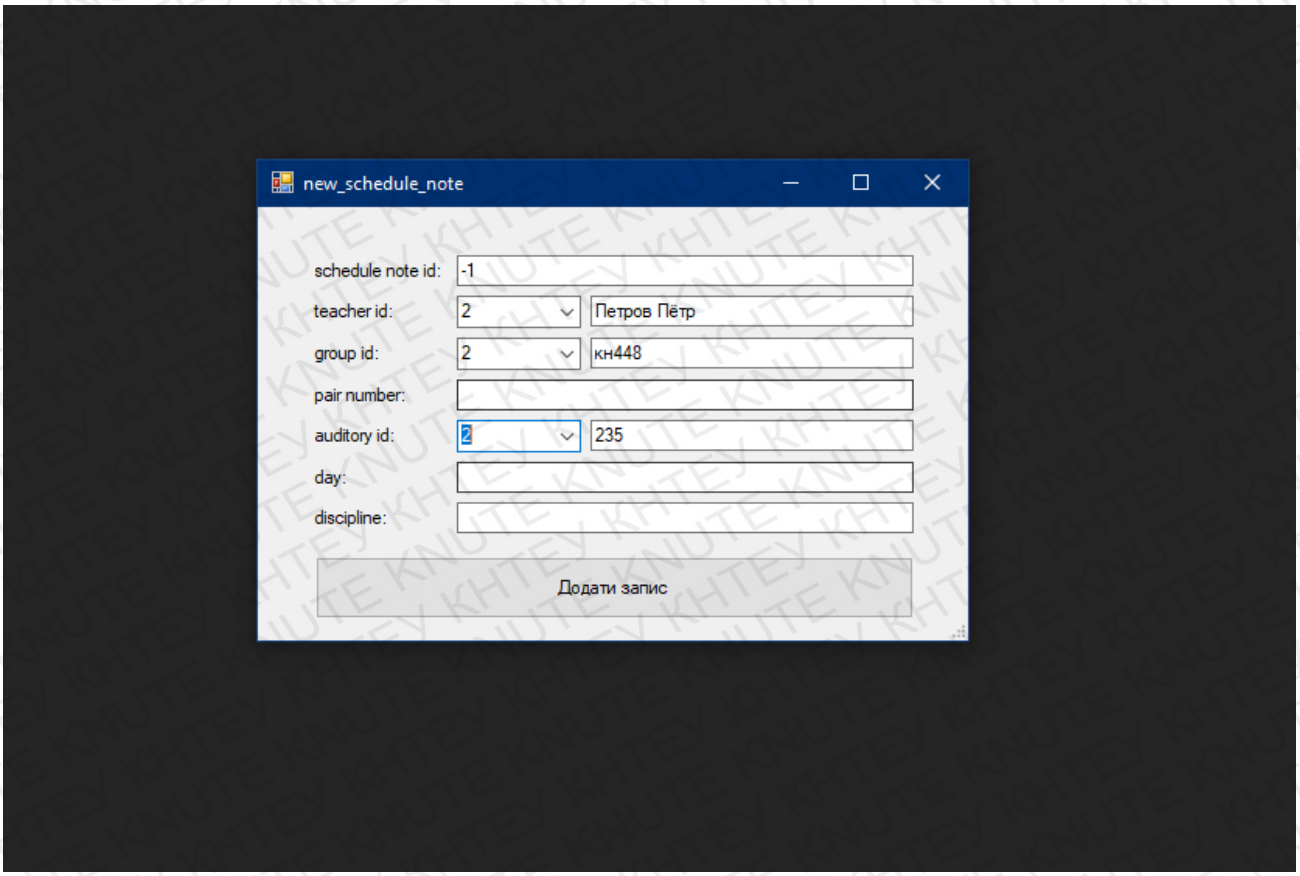

Рис. 3.6 — Форма нового запису розкладу

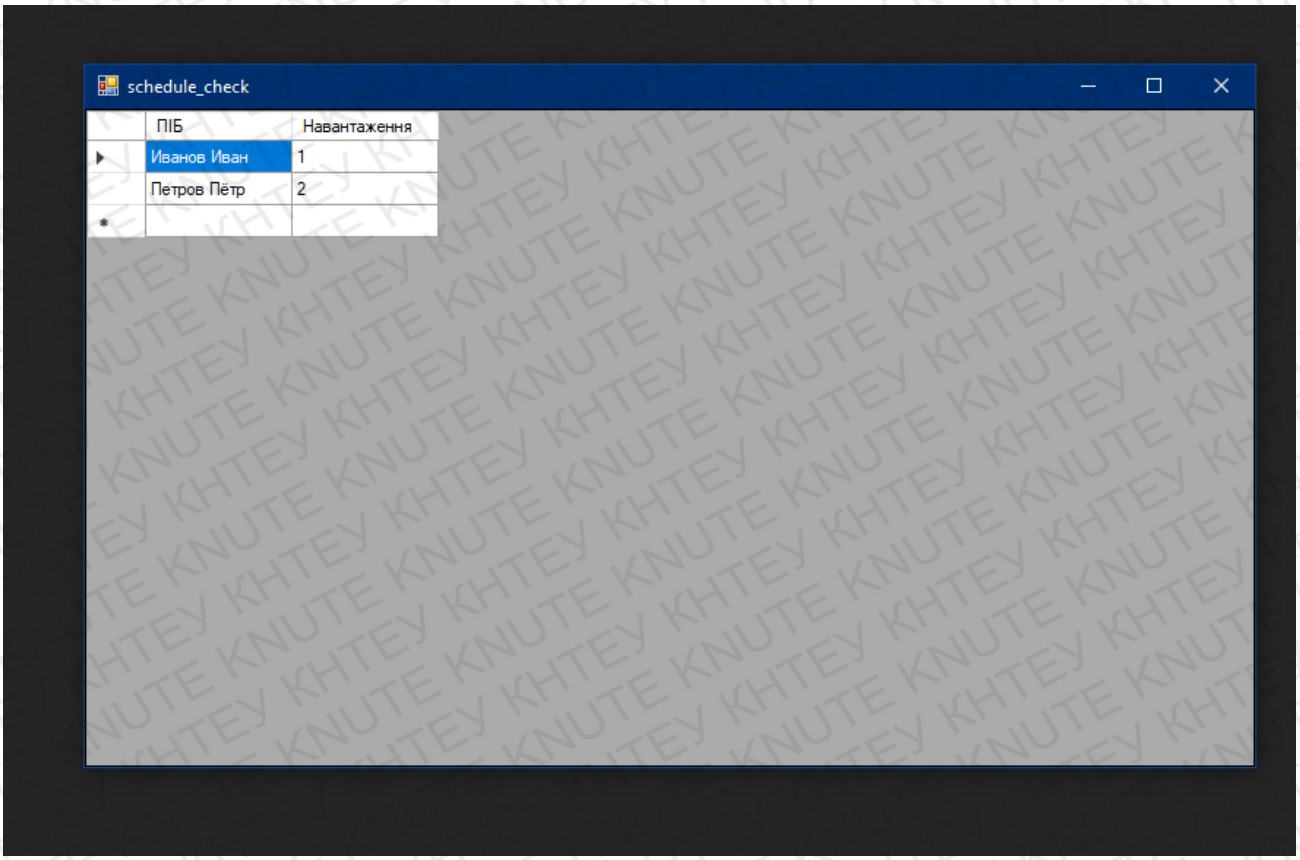

Рис. 3.7 — Форма перевірки навантаження

ВИСНОВКИ

В ході роботи над даним проектом було виконано наступні завдання:

- проаналізувати теоретичні методи створення системи розподілу навантаження викладачів кафедри;
- проаналізувати математичну модель;
- сформувати систему показників;
- обрати інструментарій розробки;
- розробити і описати систему.

В ході аналізу теоретичних методів створення системи розподілу навантаження викладачів кафедри було проведено аналіз поняття розподілення навантаження між співробітниками і виявлено, що розподіл навантаження — це процес побудови оптимального розподілу завдань між співробітниками, спрямований на адекватне завантаження роботою кожного співробітника роботою, відповідно до його спеціалізації, кваліфікації, тощо.

Також було проаналізовано особливості створення системи, описано основний функціонал.

В розділі 2 була проаналізована модель системи, була створена система показників для розробки інформаційної системи та створено математичну модель, яка обґрунтовує необхідність перерозподілу навантаження між викладачами кафедри та механізм перерозподілу.

Третій розділ присвячений проектуванню та розробці інформаційної системи розподілу навантаження викладачів кафедри. В даному розділі було обрано інструментарій розробки, а саме проведено огляд мови програмування C# та середовища розробки Visual Studio. Проаналізовані їх переваги і недоліки, особливості користування даними технологіями.

Окрім вибору інструментарію в третьому розділі було розроблено діаграму варіантів використання і діаграму класів, які повністю описують функціонал програми, доступний користувачеві та внутрішню структуру програмного забезпечення.

Розділ, присвячений розробці, закінчується підрозділом про розроблення графічного інтерфейсу користувача, в якому відображені всі важливі моменти, пов’язані з побудовою інтерфейсу користувача.

Розроблена програма здатна сприяти адекватному розподілу навантаження на викладачів кафедри за рахунок створення нормально розподіленого розкладу, без перекосів в сторону того чи іншого викладача.

Завдяки чіткому виконанню поставлених на початку роботи завдань в результаті роботи було отримано повноцінну функціонуючу систему, яка здатна виконувати функції, закладені в неї при проектуванні і готова до використання в реальних умовах за деяких доробок.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DuBrin, Andrew J. (2009). Essentials of management (8th ed.). Mason, OH: Thomson Business & Economics. ISBN 978-0-324-35389-1. OCLC 227205643.
2. Waring, S.P., 2016. Taylorism transformed: Scientific management theory since 1945. UNC Press Books.
3. Xenophon (1734). "Oikonomikos. Oder Xenophon vom Haus-Wesen, aus der Griechischen- in die Teutsche Sprache übersetzet von Barthold Henrich Brockes, dem jüngern. Mit einer Vorrede S.T. Herrn Jo. Alb. Fabricii ... Nebst den wenigen Stücken, die aus der Lateinischen Uebersetzung Ciceronis noch übrig".
4. "Home : Oxford English Dictionary".
5. SS Gulshan. Management Principles and Practices by Lallan Prasad and SS Gulshan. Excel Books India. pp. 6–. ISBN 978-93-5062-099-1.
6. Ann Viola Ulvin
7. Deslandes G., (2014), “Management in Xenophon's Philosophy : a Retrospective Analysis”, 38th Annual Research Conference, Philosophy of Management, 2014, July 14–16, Chicago
8. Prabbal Frank attempts to make a subtle distinction between management and manipulation: Frank, Prabbal (2007). People Manipulation: A Positive Approach (2 ed.). New Delhi: Sterling Publishers Pvt. Ltd (published 2009). pp. 3–7. ISBN 978-81-207-4352-6. Retrieved 2015-09-05. There is a difference between management and manipulation. The difference is thin [...] If management is handling, then manipulation is skilful handling. In short, manipulation is skilful management. [...] Manipulation is in essence leveraged management. [...] It is an alive thing while management is a dead concept. It requires a proactive approach rather than a reactive approach. [...] People cannot be managed.

9. Powell, Thomas C. (2001). "Competitive advantage: logical and philosophical considerations". Strategic Management Journal. 22 (9): 875–888. doi:10.1002/smj.173. ISSN 1097-0266.
10. Langfred, Claus (2000). "The paradox of self-management: individual and group autonomy in work groups". Journal of Organizational Behavior. 21 (5): 563–585. doi:10.1002/1099-1379(200008)21:5<563::AID-JOB31>3.0.CO;2-H.
11. Wood, Robert; Bandura, Albert (1989). "Social Cognitive Theory of Organizational Management". The Academy of Management Review. 14 (3): 361–384. doi:10.2307/258173. ISSN 0363-7425. JSTOR 258173.
12. Lumineau, Fabrice; Oliveira, Nuno (2017). "A Pluralistic Perspective to Overcome Major Blind Spots in Research on Interorganizational Relationships". Academy of Management Annals. 12 (1): 440–465. doi:10.5465/annals.2016.0033. ISSN 1941-6520. S2CID 148825815.
13. Administration industrielle et générale – prévoyance organization – commandment, coordination – contrôle, Paris : Dunod, 1966
14. Jones, Norman L. (2013-10-02). "Chapter Two: Of Poetry and Politics: The Managerial Culture of Sixteenth-Century England". In Kaufman, Peter Iver (ed.). Leadership and Elizabethan Culture. Jepson Studies in Leadership. Palgrave Macmillan (published 2013). p. 18. ISBN 978-1-137-34029-0. Retrieved 2015-08-29. Mary Parker Follett, the 'prophet of management' reputedly defined management as the 'art of getting things done through people.' [...] Whether or not she said it, Follett describes the attributes of dynamic management as being coactive rather than coercive.
15. Vocational Business: Training, Developing and Motivating People by Richard Barrett – Business & Economics – 2003. p. 51.
16. Compare: Holmes, Leonard (2012-11-28). The Dominance of Management: A Participatory Critique. Voices in Development Management. Ashgate

Publishing, Ltd. (published 2012). p. 20. ISBN 978-1-4094-8866-8. Retrieved 2015-08-29. Lupton's (1983: 17) notion that management is 'what managers do during their working hours', if valid, could only apply to descriptive conceptualizations of management, where 'management' is effectively synonymous with 'managing', and where 'managing' refers to an activity, or set of activities carried out by managers.

17. Harper, Douglas. "management". Online Etymology Dictionary. Retrieved 2015-08-29. – "Meaning 'governing body' (originally of a theater) is from 1739."
18. See for examples Melling, Joseph; McKinlay, Alan, eds. (1996). Management, Labour, and Industrial Politics in Modern Europe: The Quest for Productivity Growth During the Twentieth Century. Edward Elgar. ISBN 978-1-85898-016-4. Retrieved 2015-08-29.
19. Compare: Vasconcelos e Sá, Jorge (2012). There is no leadership: only effective management: Lessons from Lee's Perfect Battle, Xenophon's Cyrus the Great and the practice of the best managers in the world. Porto: Vida Economica Editorial. p. 19. ISBN 9789727886012. Retrieved 2020-01-22. [...] to ask what is leadership about [...] is a false question. The right question is: what is effective management?
20. Board of Directors: Duties & Liabilities Archived 2014-03-24 at the Wayback Machine. Stanford Graduate School of Business.
21. DeMars L. (2006). Heavy Vetting: Boards of directors now want to talk to would-be CFOs — and vice versa. CFO Magazine.
22. 2013 CEO Performance Evaluation Survey. Stanford Graduate School of Business.
23. Kleiman, Lawrence S. "Management and Executive Development."Reference for Business:Encyclopedia of Business(2010): n.p. 25 Mar 2011. [1].
24. "AOM Placement Presentations".

25. "Four Ways to Be A Better Boss". www.randstadusa.com. Randstad USA. Retrieved 18 January 2015.
26. "The Role of HR in Uncertain Times" (PDF). Economist Intelligence Unit. Economist Intelligence Unit. Retrieved 18 January 2015.
27. Pfeffer J, Sutton RI (March 2006). Hard Facts, Dangerous Half-Truths And Total Nonsense: Profiting From Evidence-Based Management (first ed.). Boston, Mass: Harvard Business Review Press. ISBN 978-1-59139-862-2.
28. Spring B (July 2007). "Evidence-based practice in clinical psychology: what it is, why it matters; what you need to know". Journal of Clinical Psychology. 63 (7): 611–31. CiteSeerX 10.1.1.456.9970. doi:10.1002/jclp.20373. PMID 17551934.
29. Lilienfeld SO, Ritschel LA, Lynn SJ, Cautin RL, Latzman RD (November 2013). "Why many clinical psychologists are resistant to evidence-based practice: root causes and constructive remedies". Clinical Psychology Review. 33 (7): 883–900. doi:10.1016/j.cpr.2012.09.008. PMID 23647856.
30. Waring, S.P., 2016, Taylorism transformed: Scientific management theory since 1945. UNC Press Books.