МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЖИТОМИРСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення

**КУРСОВА РОБОТА**

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

з дисципліни: «Бази даних»

на тему:

**«БАЗА ДАНИХДЛЯ СЕРВІСНОГО ЦЕНТРУ З РЕМОНТУ ТА ОБСЛУГОВУВАННЯ ОРГТЕХНІКИ»**

студентки ІІІкурсу групи ПІ-54

спеціальності 121 «Інженерія програмного забезпечення»

Раздобудової Валерії Олексіївни

(прізвище, ім’я та по-батькові)

Керівник старший викладач кафедри КНКоротун О.В. ,

Дата захисту: " \_\_\_ " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 р.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ І.І. Сугоняк \_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.В. Коротун.

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.О. Данильченко\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

Житомир – 2018

РЕФЕРАТ

Пояснювальна записка до курсовоїроботи на тему «Розробка бази даних длясервісного центруз ремонту та обслуговування оргтехніки» складається з вступу, чотирьох розділів, висновків, списку використаної літератури та додатків.

Текстова частина викладена на 35сторінкахдрукованого тексту.

Пояснювальна записка має 59 сторінок додатків. Список використаних джерел містить 12 найменувань і займає 1 сторінку. В роботі наведено 25 рисунків та 15 таблиць. Загальний обсяг роботи – 95 сторінок.

У першому розділі було обґрунтовано створення бази даних длясервісного центру.

У другому розділі було спроектовано БД для сервісного центру.

У третьому розділі проведено проектування і розробка програмного продукту.

У четвертому розділі описано адміністрування БД.

Висновок містить в собі результати виконаної роботи створення бази даних для сервісного центру.

У додатку представлений лістинг розробленого програмного продукту.

Ключові слова: БАЗА ДАНИХ, СЕРВІСНИЙ ЦЕНТР,ТЕХНІКА, ЗАПИСИ, КЛІЄНТИ, РОЗРОБКА БАЗИ ДАНИХ, ФУНКЦІОНАЛ БАЗИ ДАНИХ.

ЗМІСТ

[ВСТУП 4](#_Toc532384928)

[РОЗДІЛ 1 ТЕОРЕТИЧНИЙ АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТОКІВ ТА 5](#_Toc532384929)

[ОСОБЛИВОСТЕЙ БАЗИ ДАНИХ СЕРВІСНОГО ЦЕНТРУ 5](#_Toc532384930)

[1.1 Аналіз інформаційних потреб та визначення бази даних сервісногоцентру 5](#_Toc532384931)

[1.2 Обґрунтування вибору засобів реалізації 6](#_Toc532384932)

[РОЗДІЛ 2 ПРОЕКТУВАННЯ БАЗИ ДАНИХ СЕРВІСНОГОЦЕНТРУ 11](#_Toc532384935)

[2.1 Аналіз інформаційних процесів 11](#_Toc532384936)

[2.2 Проектування структури бази даних сервісногоцентру 12](#_Toc532384937)

[РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПІДСИСТЕМИ ОБРОБКИ ДАНИХ СЕРВІСНОГОЦЕНТРУ 19](#_Toc532384940)

[3.1 Проектування інтерфейсу обробки даних 19](#_Toc532384941)

[3.2 Реалізація операцій обробки даних в БД сервісногоцентру 23](#_Toc532384942)

[3.3 Організація звітності системи 26](#_Toc532384942)

[РОЗДІЛ 4 АДМІНІСТРУВАННЯ БАЗ ДАНИХ 29](#_Toc532384944)

[4.1 Розробка заходів захисту інформації в БД 29](#_Toc532384945)

[4.2 Налаштування параметрів роботи SQL-сервера 31](#_Toc532384946)

[ВИСНОВКИ 34](#_Toc532384948)

[ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 35](#_Toc532384949)

[ДОДАТКИ 36](#_Toc532384950)

# **ВСТУП**

В данійкурсовій роботі наведено процес створення бази даних, яка дає можливість вести облік даних по управлінню сервісним центром.

Сервісний центр - організація, що займається наданням послуг з сервісної підтримки і обслуговування техніки, обладнання та іншої продукції. Діяльність сервісних центрів включає передторговий, гарантійний і післяпродажний ремонт.

Ця тема є актуальною, бо наразі вибірка необхідних даних для звітів та облікроботи персоналу зазвичай проводиться вручну або засобами Excel і на це витрачається значна кількість часу. База даних дасть можливість працювати та обробляти дані легше та швидше. Дана програма забезпечує також можливостями пошуку, фільтрації і сортування даних. Томупрограма є корисною та полегшує роботу працівникамсервісного центрута її клієнтам.

Дуже важливо, щоб база даних виконувала кілька важливих функцій – відображала всю інформацію про сервісний центр, в тому числі і дані про послуги, замовлення, клієнтів, інженерів, надавала потрібну інформацію клієнтам, давала користувачам можливістьдодавання та проведення дій над записами замовлень.

**Метою** роботи є дослідження особливості проектування та реалізації бази даних для сервісного центру.

**Об’єктом** дослідження є методи та засоби проектування баз даних для сервісного центру.

**Предметом** дослідження є розробка бази даних для сервісного центру.

# **РОЗДІЛ 1 ТЕОРЕТИЧНИЙ АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТОКІВ ТА**

# **ОСОБЛИВОСТЕЙ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ**

## **Аналіз інформаційних потреб та визначення предметної області дослідження**

Сучасні сервісні центри не можуть існувати без використання баз даних. База даних сервісного центру призначена для її працівників і забезпечує доступ до інформації про занесених до бази замовлень, клієнтів, типів послуг, працівників, техніки. Можна скористатися системою захисту для конфіденційності інформації. Ефективність програми полягає в скороченні часу на обробку, пошуку необхідної інформації. Ніяких спеціальних знань для роботи з програмою в області програмування не потрібно.

Базою даних (БД) називають пойменовану сукупність структурованих даних, які стосуються певної предметної області. Предметна область - це частина реального світу, що підлягає вивченню з метою створення бази даних для автоматизації процесу управління.

Основною задачею бази даних є нести в собі інформацію, максимально доступно й зручно подавати її користувачеві, забезпечувати актуальність і корисність цієї інформації.

Основною причиною закономірного виникнення концепції баз даних є прагнення підвищити гнучкість автоматизованих інформаційних систем, тобто зробити їх менш залежними від змін вимог до автоматизованої інформаційної системи (АІС) з обробки інформації та більш придатними для розвитку і подальшої модифікації.

Визначимо основні ідеї, що лежать в основі концепції бази даних:

1. Ізолювати будь-яку прикладну програму від впливу змін, внесених в інші програми, через загальні дані шляхом розмежування логічних записів, які використовуються прикладними програмами, від записів, які фізично запам'ятовуються на магнітних носіях.

2. Усунути надлишкове дублювання даних.

3. Централізувати управління даними.

База даних для сервісногоцентру є необхідною, бо працюючи з багатьма даними персонал має великі труднощі в систематизації та обробціінформації.

Паперова робота потребує більшої пильності та забирає більше часу працівників студії звукозапису.

Дуже важливим є те, щоб дані відповідали деяким пунктам, були:

1. Взаємопов'язаними (має бути забезпечена повна й узгоджена інформація про предметну область).

2. Інтегрованими (за умови мінімальної надмірності, необхідної для забезпечення взаємопов'язаності файлів).

3. Незалежними (логічно та фізично від програм, у яких вони використовуються, і від процесів, у яких вони підтримуються).

4. Мали єдину централізовану програму управління, що забезпечує логічну незалежність програм від даних, що знаходяться у файлах.

## **Обґрунтування вибору засобів реалізації**

СУБД - комплекс програмних засобів, призначених для створення структури нової бази, наповнення її вмістом, редагування вмісту і візуалізації інформації. Під *візуалізацією інформації* бази треба розуміти відбір даних, що відображується відповідно до заданого критерію, їхнє упорядкування, оформлення і наступна видача на пристрій виводу або передача по каналах зв'язку.

Існує багато систем управління базами даних. Вони можуть по-різному працювати з різними об'єктами і надають користувачу різні функції і засоби. Більшість СУБД спираються на єдиний сталий комплекс основних понять.

Всі існуючі сучасні БД і СУБД задовольняють, наступним основним вимогам:

- можливість подання структур даних, що є адекватними реальній предметній області (побудова адекватної інформаційної моделі предметної області);

- простота і малі витрати ресурсів на розвиток системи (швидка і дешева модифікація старих та розробка нових програмних додатків в рамках АІС);

- простота та оперативність доступу до даних, можливість пошуку інформації різними методами;

- можливість ефективного одночасного обслуговування великої кількості користувачів;

- забезпечення представлення даних користувачам у зручному вигляді для їх подальшого застосування;

- забезпечення захисту інформації в БД від збоїв і відмов у роботі технічних засобів і помилок користувачів.

Основні переваги застосування БД та СУБД під час реалізації на їх основі автоматизованих пошуково-інформаційних систем:

- скорочення надмірності даних, які зберігаються. Дані, що використовуються кількома програмами, інтегруються і зберігаються в одному місці;

- усувається суперечливість даних, яка може виникати, якщо одні й ті ж дані, що використовуються різними програмами, подаються кілька разів або у випадку необхідності їх зміни, не всі копії відновлені;

- дані, що зберігаються, використовуються спільно. Це надає можливість розробляти нові програмні додатки для наявної базою даних з мінімальними витратами;

- забезпечується більш швидкий та дешевший розвиток АІС за рахунок забезпечення логічної взаємної незалежності програм і даних у БД;

- спрощується підтримка цілісності даних (адекватності та узгодженості);

- забезпечується можливість швидкого надання даних на нестандартні (заздалегідь непередбачені) запити користувачів без додаткової розробки прикладних програм;

- у разі централізованого управління базою даних спрощується стандартизація та уніфікація представлення даних у АІС.

Коли необхідно вибрати СУБД, головне питання зазвичай полягає у виборі реляційної (SQL) або нереляційної (NoSQL) структури. У обох варіантів є свої переваги, а також кілька ключових особливостей, які варто мати на увазі при виборі.

Для вірного вибору СУБД проведемо порівняльний аналіз за декількома параметрами.

Для початку порівняємо за критеріями масштабу підприємства, моделі даних та за потрібним рівнем кваліфікованості персоналу (табл.1).

Таблиця 1

**Характеристики СУБД**

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметри**  **СУБД** | **Масштаб застосування** | **Модель даних** | **Необхідний рівень кваліфікації персоналу** |
| MS SQL Server | Підприємства малих, середніх та великих розмірів | Клієнт – серверна | Середній та низький |
| MySQL | Підприємства малих та середніх розмірів | Клієнт – серверна | Середній та низький |
| Oracle Database | Від персонального користування до великого підприємства | Клієнт – серверна | Високий та середній |
| Microsoft Access | Підприємства малих та середніх розмірів | Файл – серверна | Низький |
| SQLite | Підприємства малих та середніх розмірів | Спроможна до вбудовування | Середній |

База даних розробляється для cервісного центру і оператор повинен мати зручність та простоту у використанні.

З попереднього порівняння можемо зробити висновки, що необхідний рівень кваліфікованості оператора в середньому дорівнює середньому рівню.

В більшості модель даних є клієнт – серверною.

В загальному, масштаб застосування є для малих та середніх підприємств.

Наступне порівняння проведемо за критеріями розробників, доступності та діалекту мови SQL (табл.2).

Таблиця 2

**Порівняння характеристик СУБД**

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметри**  **СУБД** | **Розробники ПЗ** | **Доступність** | **Діалект мови SQL** |
| MS SQL Server | Sybase, Ashton-Tate, Microsoft | Платно, але є безкоштовна полегшена версія | Transact – SQL |
| MySQL | MySQL AB, Sun Microsystems, Oracle | Безкоштовно, є комерційна версія | Частково відповідає стандарту SQL :2003 |
| Oracle Database | Oracle | Платно, але є безкоштовна полегшена версія | PL/SQL |
| Microsoft Access | Microsoft | У вільному доступі | Jet SQL |
| SQLite | Hwaci | Безкоштовно | Частково відповідає стандарту SQL–92 |

Виходячи з порівняльних характеристик СУБД та потрібних вимог до бази даних, що розробляється, обираємо MSSQLServer.

MSSQLServer добре адаптується до графічної реалізацій функцій розроблених в БД.

Для розробки бази даних та її графічного інтерфейсу використовують безліч засобів, але в даному курсовому проекті використані наступні:

1. Microsoft SQL Server — комерційна система керування базами даних, що розповсюджується корпорацією Microsoft. Мова, що використовується для запитів — Transact-SQL, створена спільно Microsoft та Sybase. Transact-SQL є реалізацією стандарту ANSI / ISO щодо структурованої мови запитів SQL із розширеннями. Використовується як для невеликих і середніх за розміром баз даних, так і для великих баз даних масштабу підприємства. Багато років вдало конкурує з іншими системами керування базами даних.
2. С# – мова програмування високого рівня з підтримкою кількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної.
3. Microsoft Visual Studio – продукт, який дозволяє розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби для адміністрування бази даних MsSQL.
4. Transact-SQL (T-SQL)- процедурне розширення мови SQL, створене компанією Microsoft.
5. HTML (англ. HyperText Markup Language, укр. Мова розмітки гіпертекстових документів) — стандартна мова розмітки веб-сторінок в Інтернеті.
6. CSS – каскадні таблиці стилів, спеціальна мова, що використовується для відображення сторінок, написаних мовами розмітки даних. Найбільш часто CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML.

Програма буде реалізовуватися у середовищі програмуванняSQLServer 2017 ManagementStudio та Visual Studio 2017.

# **РОЗДІЛ 2 ПРОЕКТУВАННЯ БАЗИ ДАНИХ ЗА НАПРЯМКОМ КУРСОВОЇ РОБОТИ**

## **2.1 Аналіз інформаційних процесів**

На даному етапі розробки розглянемо структуру АІС сервісного центру.Загальна схема програми приведена на рисунку 2.1.

АІС сервісного центру

Автентифікація

Реєстрація

Перегляд послуг

Клієнтська частина

Адміністративна частина

Заповнення заявки

Опції

Перегляд звітності

Редагування бази даних

Поповнення бази даних

- резервне копіювання бази  
- відновлення бази  
- експорт в Excel

- типи обладнання

- постачальники обладнання  
- обладнання на складі  
- типи послуг

- послуги

- клієнти

- працівники

- записи

- обробка заявок

Рис. 2.1Структура АІС сервісного центру

Адміністративна частина дозволяє оператору сервісного центру(після входження в систему адміністрування) здійснювати редагування бази даних та внесення додаткових елементів (постачальників обладнання, типів обладнання,обладнання, типів послуг, послуг, працівників, клієнтів, записів, обробку заявок) в базу даних. Клієнтська частина даної можливості не має, тобто доступ до ресурсів центру закритий для загального огляду. Крім того, в адміністративній частині доступний перегляд звітності та такі додаткові опції як резервне копіювання бази, відновлення бази та експорт сервісів в Excel файл.

Клієнтська частина доступна всім користувачам сервісного центру. Дозволяє клієнтам здійснювати пошук та перегляд доступних спослуг. Авторизований користувач має змогу залишити заявку, заповнивши необхідну форму.

Кінцевим результатом роботи інформаційної системи є узагальнення введених даних і формування відповідних звітів.

## **2.2 Проектування структури бази даних за напрямом курсової роботи**

При проектуванні до бази даних були включені наступні таблиці:

* dbo.TypesNTCequipment;
* dbo.NTCequipmentProviders;
* dbo.NTCequipmentInStock;
* dbo.TypesNTCServices;
* dbo.NTCServices;
* dbo.NTCClients;
* dbo.NTCWorkers;
* dbo.NTCJournal;
* dbo.Application;
* dbo.AspNetUsers;
* dbo.AspNetRoles;
* dbo.AspNetUserRoles;

Схема БД представлена на рисунку 2.2.

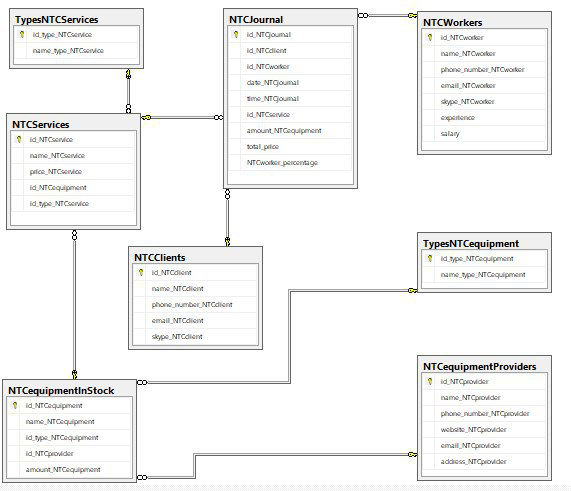


Рис. 2.2 Схема бази данихсервісного центру

Для збереження основних даних про послуги використовується таблиця «NTCServices». В дану таблицю заносяться всі послуги, які доступні в сервісномуцентрі. Структура таблиці наведена нижче.

Таблиця 3

**Структура таблиці «NTCServices»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| id\_NTCservice | int AI | + | - | Ідентифікатор |
| name\_NTCservice | varchar(50) | - | - | Назва |
| price\_NTCservice | int | - | - | Ціна, грн |
| id\_type\_NTCservice | int | - | + | Ідентифікатор типу послуг |
| id\_NTCprovider | int | - | + | Ідентифікатор постачальника техніки |

Для збереження данихтипівпослугвикористовується таблиця «TypesNTCServices». В дану таблицю заносяться всі типи послуг, які доступні в сервісному центрі. Структура таблиці наведена нижче.

Таблиця 4

**Структура таблиці «TypesNTCServices»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| id\_type\_NTCservice | int AI | + | - | Ідентифікатор |
| name\_type\_NTCservice | varchar(50) | - | - | Назва |

Для збереження даних про клієнтів використовується таблиця «NTCClients». В дану таблицю заносяться всі клієнти, які обслуговуються в сервісному центрі. Структура таблиці наведена нижче.

Таблиця 5

**Структура таблиці «NTCClients»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| id\_NTCclient | Int AI | + | - | Ідентифікатор |
| name\_NTCclient | varchar(50) | - | - | ПІБ |
| phone\_number\_NTCclient | varchar(30) | - | - | Номер телефону |
| email\_NTCclient | varchar(50) | - | - | Email |
| skype\_NTCclient | varchar(50) | - | - | Skype ID |

Для збереження даних про працівниківвикористовується таблиця «NTCWorkers». В дану таблицю заносяться всі працівники сервісного центру.

Таблиця 6

**Структура таблиці «NTCWorkers»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| id\_NTCworker | int AI | + | - | Ідентифікатор |
| name\_NTCworker | varchar(50) | - | - | ПІБ |
| experience | varchar(30) | - | - | Досвід |
| phone\_number\_NTCworker | varchar(30) | - | - | Номер |
| email\_NTCworker | int | - | - | Email |
| skype\_NTCworker | int | - | - | Skype ID |
| salary | int | - | - | Заробітня плата, грн |

Для збереження даних про записи використовується таблиця «NTCJournal». В дану таблицю заносяться всізаписи клієнтів сервісногоцентру. Структура таблиці наведена нижче.

Таблиця 7

**Структура таблиці «NTCJournal»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| id\_NTCjournal | int AI | + | - | Ідентифікатор |
| id\_NTCclient | int | - | + | Ідентифікатор клієнта |
| id\_NTCworker | date | - | + | Ідентифікатор інженера |
| date\_NTCjournal | time | - | - | Дата запису |
| time\_NTCjournal | int | - | - | Дата запису |
| id\_NTCservice | int | - | + | Ідентифікатор послуги |
| amount\_NTCequipment | int | - | - | К-сть обладнання |
| total\_price | int | - | - | Загальна сума |

Для збереження обладнання сервісного центру використовується таблиця «NTCequipmentInStock». В дану таблицю заносяться всеобладнання. Структура таблиці наведена нижче.

Таблиця 8

**Структура таблиці «NTCequipmentInStock»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| id\_NTCequipment | int AI | + | - | Ідентифікатор |
| name\_NTCequipment | varchar(50) | - | - | Назва обладнання |
| id\_type\_NTCequipment | int | - | + | Тип обладнання |
| id\_NTCprovider | int | - | + | Ідентифікатор постачальника |
| amount\_NTCequipment | int | - | - | Кількість товару, шт |

Для збереження даних про типиобладнаннявикористовується таблиця «TypesNTCequipment». В дану таблицю заносяться всі доступні типи обладнання. Структура таблиці наведена нижче.

Таблиця 9

**Структура таблиці «TypesNTCequipment»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| id\_type\_NTCequipment | int AI | + | - | Ідентифікатор |
| name\_type\_NTCequipment | varchar(50) | - | - | Назва |

Для збереження даних про постачальників обладнання використовується таблиця «NTCequipmentProviders». В дану таблицю заносяться всі постачальникиобладнаннясервісному центрі. Структура таблиці наведена нижче.

Таблиця 10

**Структура таблиці «NTCequipmentProviders»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| id\_NTCprovider | int AI | + | - | Ідентифікатор |
| name\_NTCprovider | varchar(50) | - | - | Назва |
| phone\_number\_NTCprovider | varchar(50) | - | - | Тип обладнання |
| email\_NTCprovider | varchar(50) | - | - | Пошта |
| website\_NTCprovider | varchar(50) | - | - | Сайт |
| address\_NTCprovider | varchar(50) | - | - | Адреса |

Для збереження заявок користувачіввикористовується таблиця «Application». В дану таблицю заносяться номери користувачів для зв’язку з ними. Структура таблиці наведена нижче.

Таблиця 11

**Структура таблиці «Application»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| id\_application | int AI | + | - | Ідентифікатор |
| phone\_numb | varchar(30) | - | - | Номер |
| mail | varchar(50) | - | - | Пошта |

Для збереження ролей користувачів сайту використовується таблиця «AspNetRoles». В дану таблицю заносяться всі ролі, які можливі в сервісному центрі. Структура таблиці наведена нижче.

Таблиця 12

**Структура таблиці «AspNetRoles»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| id | varchar(128) | + | - | Ідентифікатор |
| name | varchar(256) | - | - | Назва |

Для збереження основних даних про користувачів сайту використовується таблиця «AspNetUsers». В дану таблицю заносяться всі користувачі, які реєструються на сайті. Структура таблиці наведена нижче.

Таблиця 13

**Структура таблиці «AspNetUsers»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| id | varchar(128) | + | - | Ідентифікатор |
| email | varchar(256) | - | - | Електронна пошта |
| passwordHash | varchar(max) | - | - | Хеш паролю |
| UserName | varchar(256) | - | - | Ім’я користувача |

Для збереження відповідностей користувачів до ролей використовується таблиця «AspNetUserRoles». В дану таблицю заносяться записи, які складаються з ідентифікатора користувача та ідентифікатора ролі. Структура таблиці наведена нижче.

Таблиця 14

**Структура таблиці «AspNetUserRoles»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва** | **Тип даних** | **ПК** | **ЗК** | **Опис поля** |
| UserId | varchar(128) | + | - | Ідентифікаторкористувача |
| RoleId | varchar(128) | - | + | Ідентифікатор ролі |

# **РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПІДСИСТЕМИ ОБРОБКИ ДАНИХ СЕРВІСНОГО ЦЕНТРУ**

## **3.1 Проектування інтерфейсу обробки даних**

Для написання програмного забезпечення інтерфейсу користувача було використано MSVisualStudio 2017. Дане середовище надає широкий вибір інструментів для створення робочого місця для користувачів інформаційної системи.

На даному етапі виконання курсової роботи вже реалізовано програмне забезпечення, тому необхідно пояснити роботу з додатком, а також протестувати його роботу.

Опис роботи з додатком включає в себе розробку текстової та ілюстраційної частин, які пояснюють принципи роботи з додатком та опис елементів інтерфейсу для його користувачів. Створений сайт складається з одної головної сторінки та інших сторінок, що забезпечують зручне користування сайтом.При запуску програми відразу відкривається головна сторінка.

На рисунку 3.1 наведено зображення головного вікна та його опис у вигляді виносок.

На рисунку 3.2 наведено зображення вікна про персонал та його опис у вигляді виносок.За замовчуванням користувач є гостем.

Кнопки для входу та реєстрації

Меню

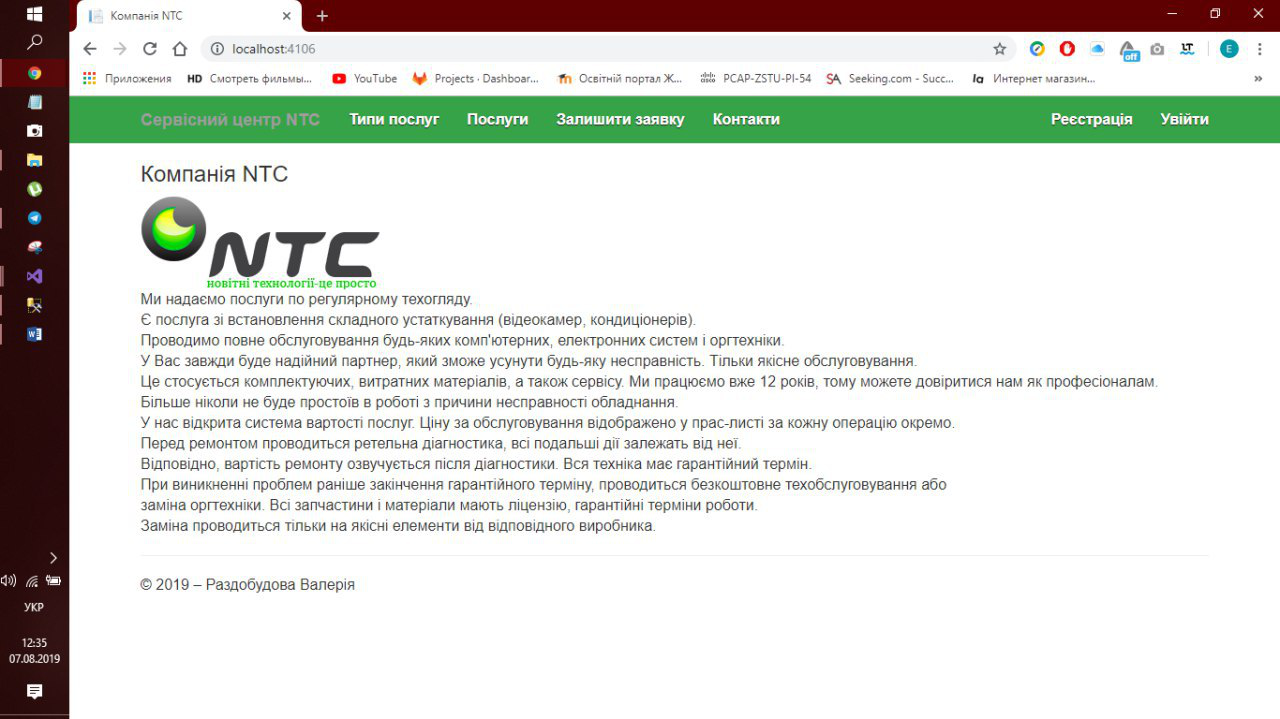


Рис. 3.1 Вигляд головної сторінки сайту з поясненнями

Інформація про доступні послуги

Полe для пошуку

Посилання на оформлення зaявки

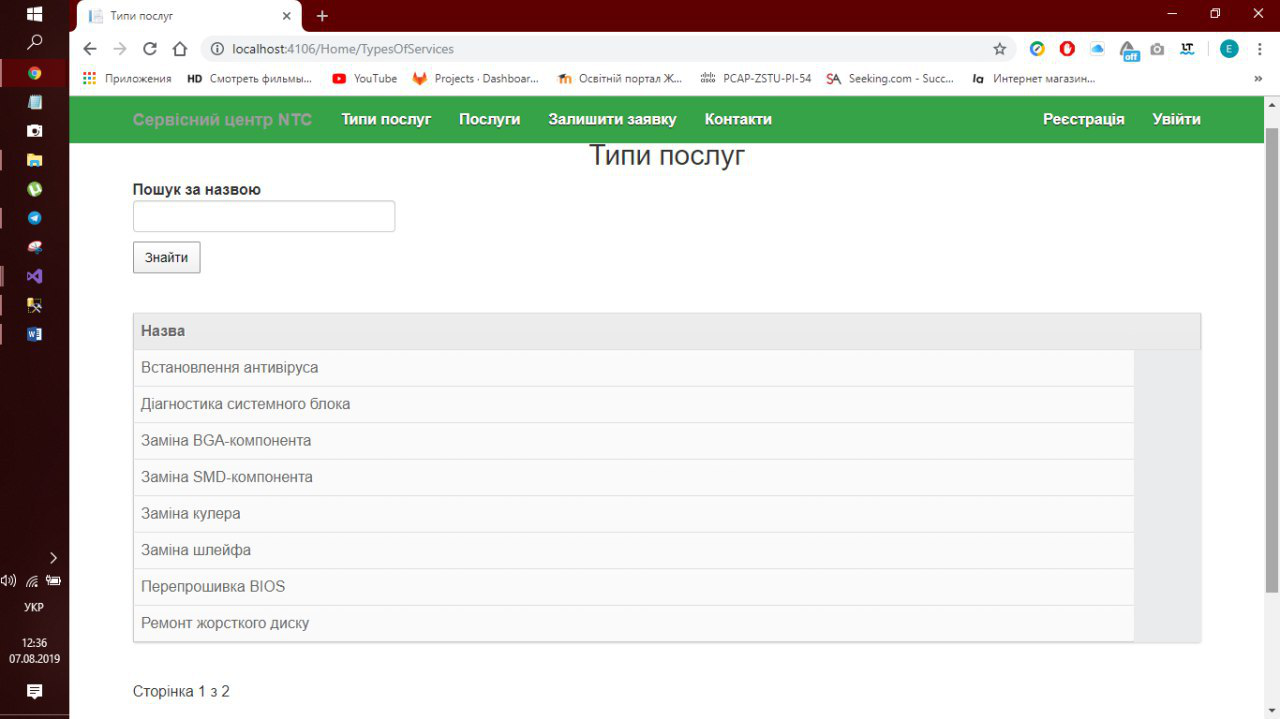


Рис. 3.2 Вигляд сторінки про доступні послуги з поясненнями

Сторінка «Контакти» містить контактну інформацію про сервісний ценр.

Контактні дані

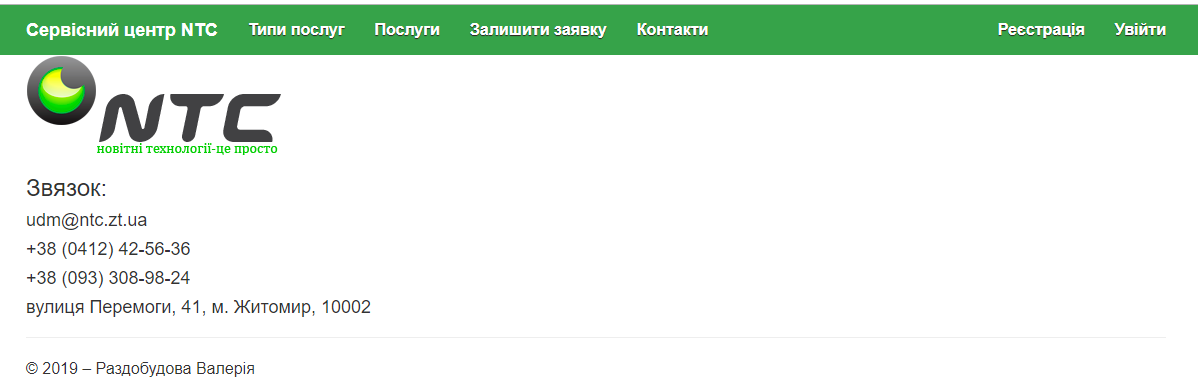


Рис. 3.5 Вигляд сторінки «Контакти» з поясненнями

Після входу користувача, в залежності від ролі, йому будуть надані різні функції та різні сторінки сайту. Наприклад, якщо користувач є гостем то йому є доступними наступні сторінки:

* «Головна»;
* «Типи послуг»
* «Послуги»
* «Контакти»;
* «Вхід»;
* «Реєстрація»

Для користувача, який здійснив вхід додається ще одна функція сайту – «Оформлення заявки. На цій сторінці, користувач зможе заповнити форму, в якій вкаже необхідні дані.

На рисунку 3.3 наведено зображення вікна оформлення замовлення та його опис у вигляді виносок.

Форма оформлення заявки

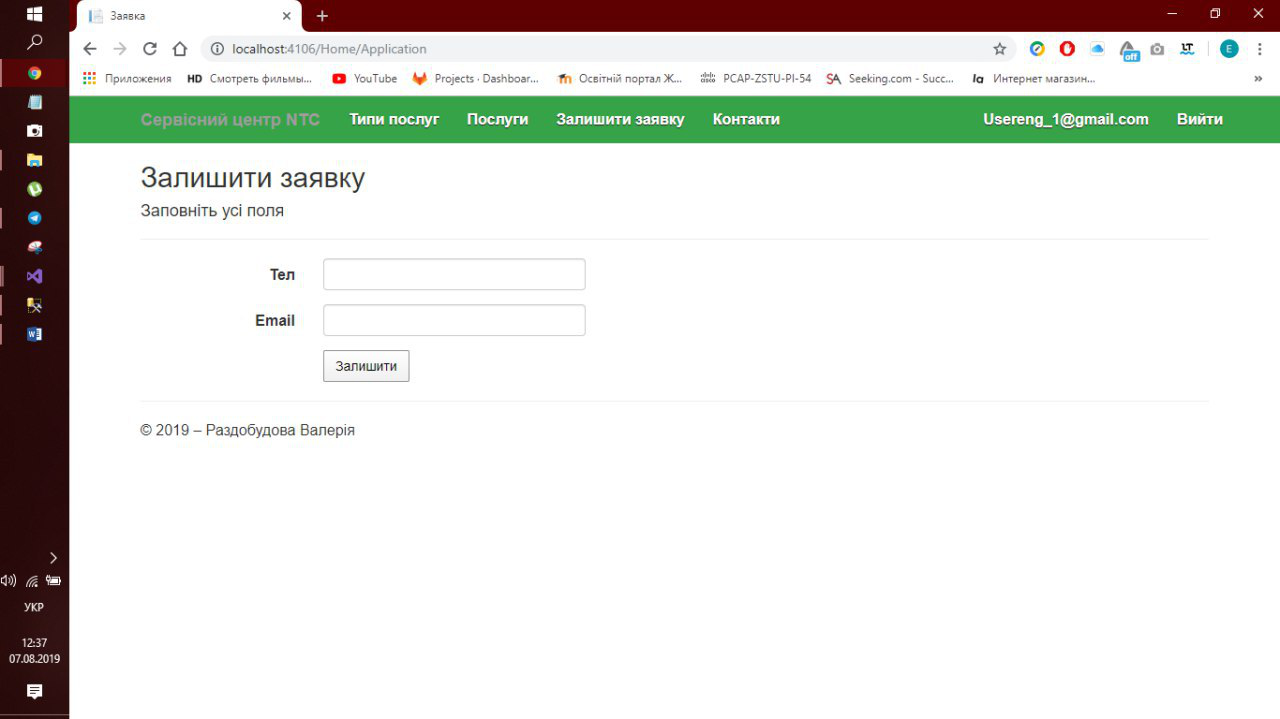


Рис. 3.3Вигляд сторінкиоформлення заявкиз поясненнями

Якщо користувач увійшов під вхідними даними адміністратора сайту, йому надаються усі права адміністрування.

Для адміністратора сервіс центру доступна панель адміністрування, за допомогою якої він може переглядати дані з усіх таблиць і виконувати всі операції над цими даними. Крім того є можливість переглядати звітність та користуватися додатковими опціями.

На рисунку 3.4наведено зображення панелі адміністрування.

На сторінці «Працівники» адміністратор може переглянути всіх працівників сервіс центру,дізнатися над чим вони працюють та деяку іншу інформацію про них.На рисунку 3.5 наведено зображення сторінки працівників.

Таблиці з даними

Додаткові опції адміністрування

Перегляд звітності

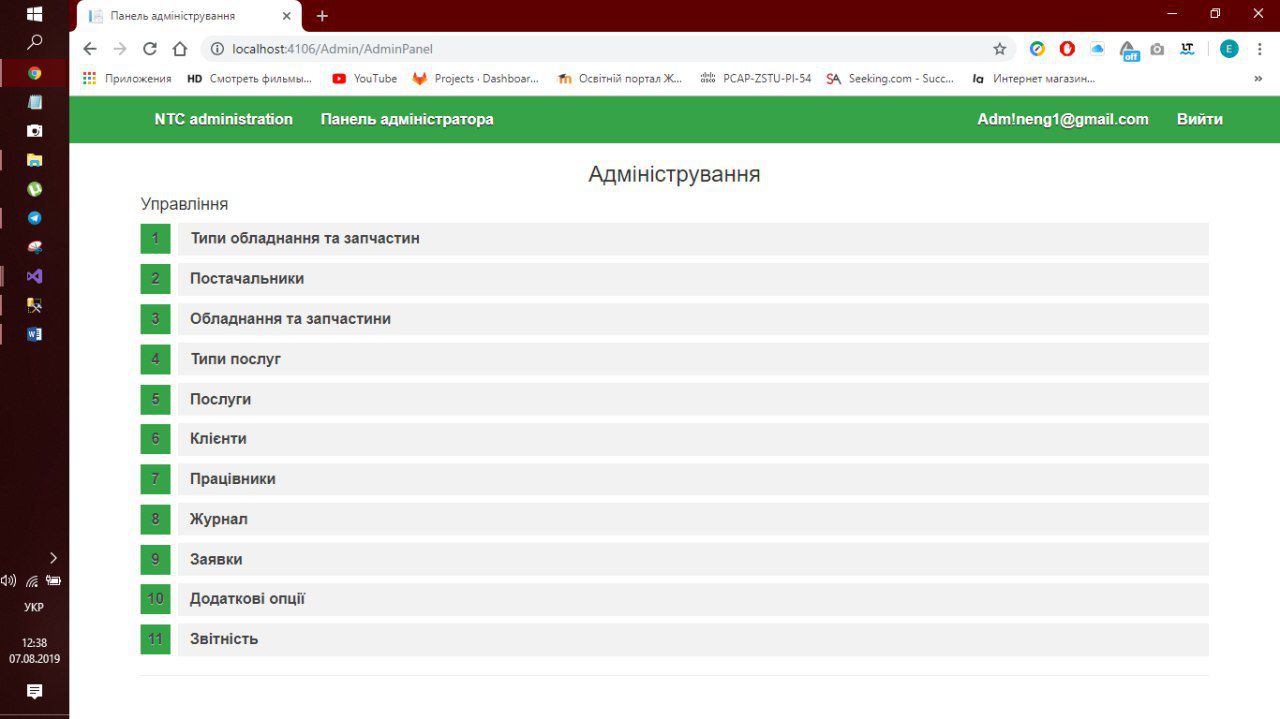


Рис. 3.4Вигляд сторінки «Панель адміністрування» з поясненнями

Додавання працівника

Поля для пошуку

Інформація про працівника

Операції над даними

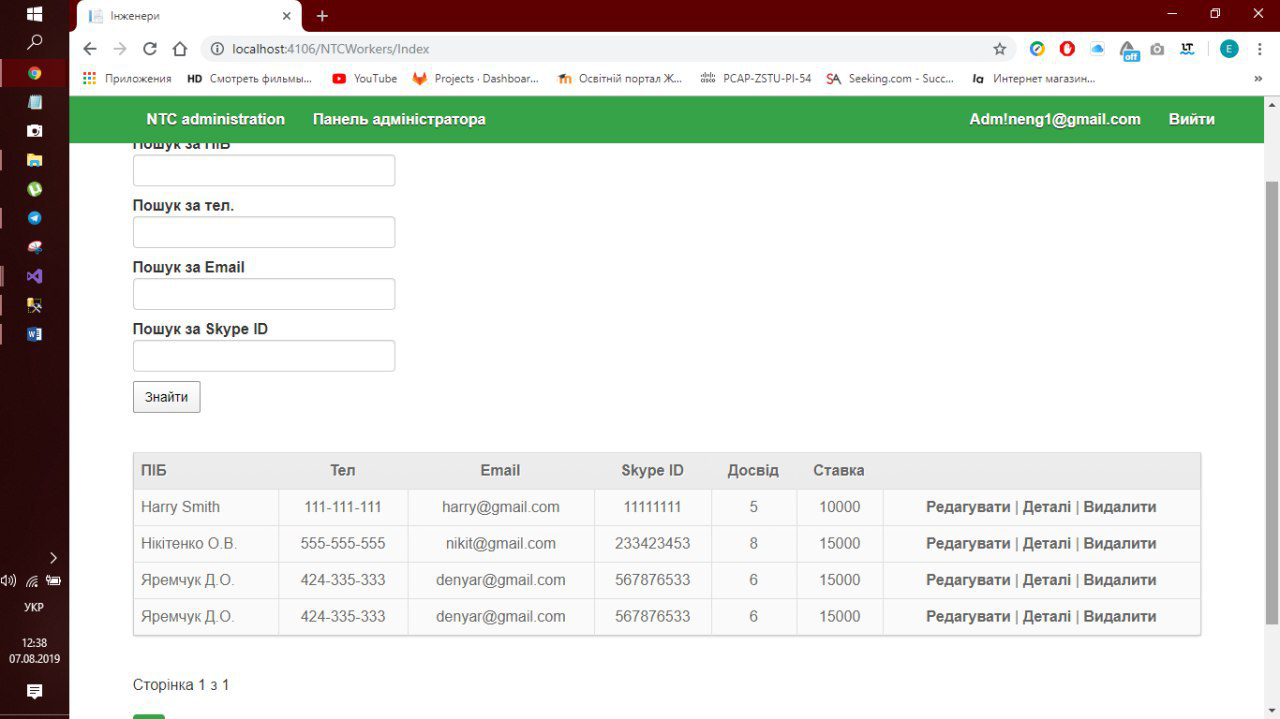


Рис. 3.5Вигляд сторінки «Працівники» з поясненнями

Отже інтерфейс сторінок має всі необхідні елементи.Такий інтерфейс буде дуже простим у користуванні і користувачу не будуть необхідні якісь додаткові пояснення для того щоб вільно користуватись програмою.

## **3.2 Реалізація операцій обробки даних в БД сервісногоцентру**

Для реалізації бази даних використано СУБД MSSQL. Реалізація розмітки сторінок виконано за допомогою мови розмітки HTML5. Для реалізації зовнішньоговигляду - CSS 3.0 та Bootstrap. Для реалізації потрібних функцій використаємо C#.

Для роботи з БД було використано технологію CodeFirst.

Проект реалізований з використанням LINQзапитів. У базі даних реалізовано 7 збережених процедур та 3 триггери.

Триггер, який надає клієнту VIP статус, якщо він обслуговується в сервіс центрі 3 і більше разів:

create trigger SetVipClient

on NTCJournal

after insert

as

begin

declare @id int

select @id = id\_NTCclient from inserted

if (select count(\*) from NTCJournal where id\_NTCclient = @id) >= 5

begin

update NTCClients set name\_NTCclient = concat(name\_NTCclient,' (Постійнийклієнт)') where NTCClients.id\_NTCclient = @id

end

end

Триггер, який встановлює поточну дату та час запису, якщо адміністратор не вказав її:

create trigger SetCurrentRegistrationDate

on NTCJournal

after insert

as

declare @currDate date = convert(date, getdate())

declare @currTime time = convert(time(0), getDate())

if @@ROWCOUNT >= 1

begin

if (select date\_NTCjournal from inserted) is null

begin

update NTCJournal set date\_NTCjournal = @currDate where id\_NTCjournal = (select id\_NTCjournal from inserted)

end

if (select time\_NTCjournal from inserted) is null

begin

update NTCJournal set time\_NTCjournal = @currTime where id\_NTCjournal = (select id\_NTCjournal from inserted)

end

end;

Триггер, який переносить запис на настуний день, якщо послуга та час однаковий:

create trigger ChangeRegistrationDate

on NTCJournal

after insert

as

begin

declare @id int, @date date, @time time

select @id = id\_NTCservice, @date = date\_NTCjournal, @time = time\_NTCjournal from inserted

if (select count(\*) from NTCJournal where date\_NTCjournal = @date and time\_NTCjournal =

@time and id\_NTCservice = @id) > 1

begin

update NTCJournal set date\_NTCjournal = dateadd(day, 1, @date)

where NTCJournal.id\_NTCjournal = (select id\_NTCjournal from inserted)

end

end

Процедура, яка виводить загальний прибуток сервісного центру:

create procedure General\_profit

create procedure General\_profit

as

begin

select sum(NTCJournal.total\_price) as General\_profit from NTCJournal

end

exec General\_profit

Процедура, яка виводить прибуток сервісного центруза місяць:

create procedure Profit\_per\_month

as

begin

select sum(NTCJournal.total\_price) as Profit\_per\_month from NTCJournal

--inner join NTCJournal on NTCequipmentInStock.id\_type\_NTCequipment = NTCJournal.id\_NTCequipment

where date\_NTCjournal > dateadd(month,-1,getdate())

end

exec Profit\_per\_monthПроцедура, яка виводить загальну кількість обладнання на сервісному центрі:

create procedure General\_amount\_equipment

as

begin

select sum(NTCequipmentInStock.amount\_NTCequipment) as General\_amount\_equipment from NTCequipmentInStock

end

exec General\_amount\_equipment

**3.3 Організація звітності системи**

На сайті є сторінка звітності сервісного центру, на якій можна переглянути різну статистичну інформацію (рис.3.13). Статистика реалізована за домогою процедур.

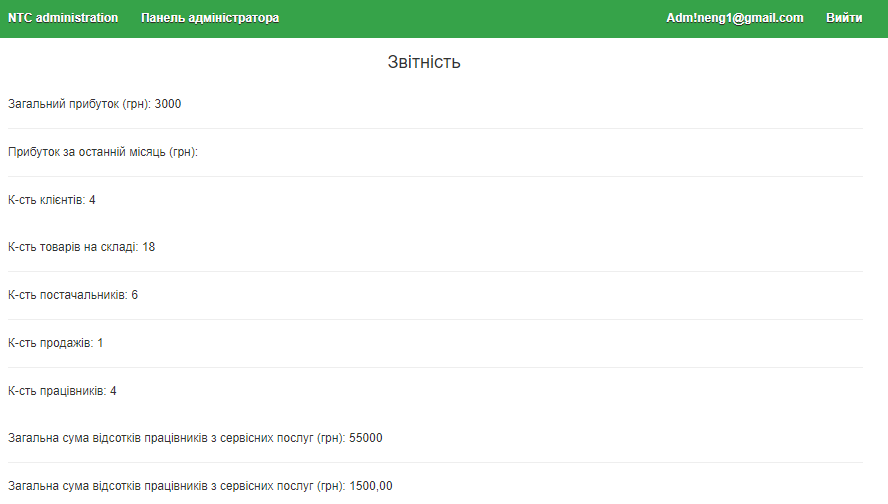


Рис. 3.13 Вигляд сторінки «Звітність»

Також на сайті передбачений пошук шляхом введення значень у поля і натиску на кнопку «Знайти» (рис. 3.10).

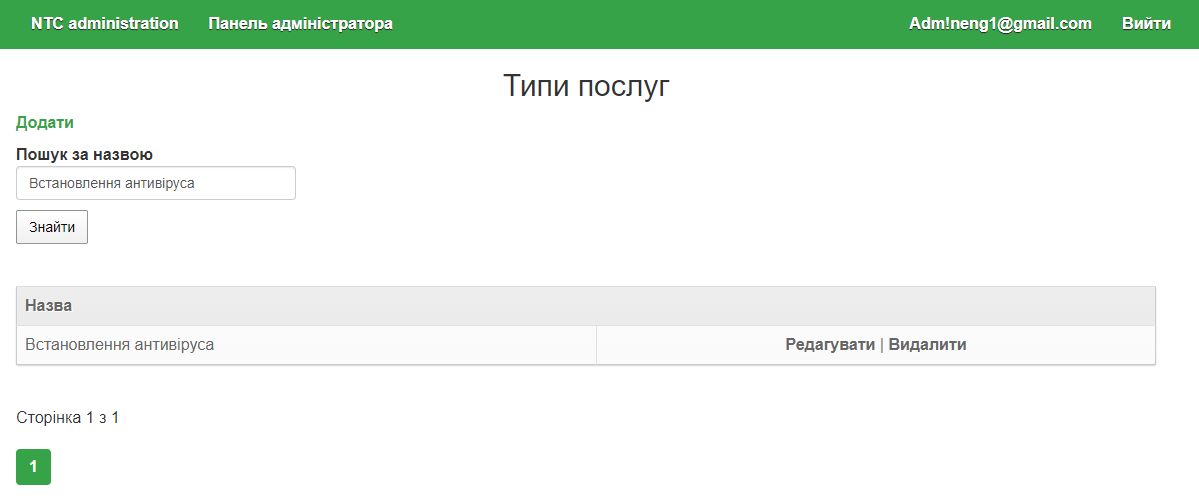


Рис. 3.10 Пошук

Є можливість переглядати конкретні дані. (рис.3.11).

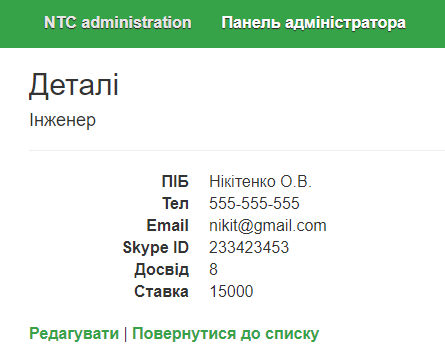


Рис. 3.11Перегляд деталей працівника

На сайті передбачене сортування шляхом натиску на назву відповідного стовпчика (рис. 3.12).

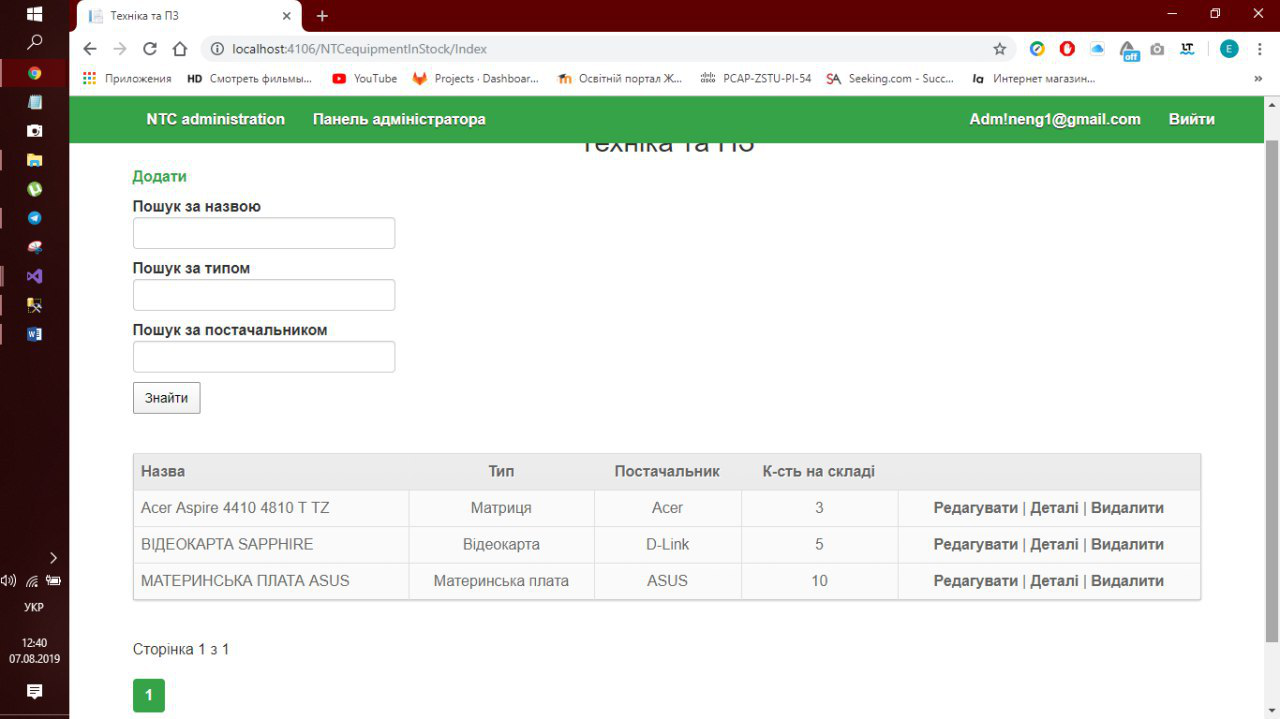


Рис. 3.12 Приклад сортування за кількістю обладнання на сервісному центрі

# **РОЗДІЛ 4 АДМІНІСТРУВАННЯ БАЗ ДАНИХ**

## **4.1 Розробка заходів захисту інформації в БД**

Методи захисту БД виконуються на рівнях додатку. Для підключення до БД на рівні додатку використовується один користувач, дані якого вказуються як дані адміністратора. Перелік об’єктів БД, доступ до яких надано користувачам БД наведено в табл. 1.1. На перетині рядків і стовпців зазначено дії, які може виконувати користувач даного типу ролі: 0 – немає доступу, 1 – перегляд даних, 2 – редагування, 3 – видалення, 4 – вставка даних, 5 - повний доступ(виконання процедури).

Таблиця 14

**Таблиця доступу до даних**

|  |  |  |
| --- | --- | --- |
| Роль  Об’єкт | Звичайний користувач | Адміністратор |
| dbo.TypesNTCServices | 1 | 1, 2, 3, 4 |
| dbo. NTCClients | 0 | 5 |
| dbo.NTCWorkers | 0 | 5 |
| dbo.NTCJournal | 0 | 5 |
| dbo.NTCServices | 1 | 5 |
| dbo.TypesNTCequipment | 0 | 1, 2, 3, 4 |
| dbo.NTCequipmentProviders | 0 | 5 |
| dbo.NTCequipmentInStock | 0 | 5 |
| dbo. Application | 1, 4 | 1, 2, 3 |

Також можна виділити зареєстрованого користувача, бо для нього додається можливість оформлення замовлення, шляхом заповнення форми.

Роути в додатку захищені та фільтрують доступ згідно з дозволеними ролями, тобтокористувач не зможе відвідати роут, до якого він не має доступу.

Крім того, для адміністратора сайту передачені додаткові опції адміністрування, а саме резервне копіювання бази даних, відновлення бази даних та експорт даних в Excel (рис. 3.14, рис. 3.15, рис. 3.16)

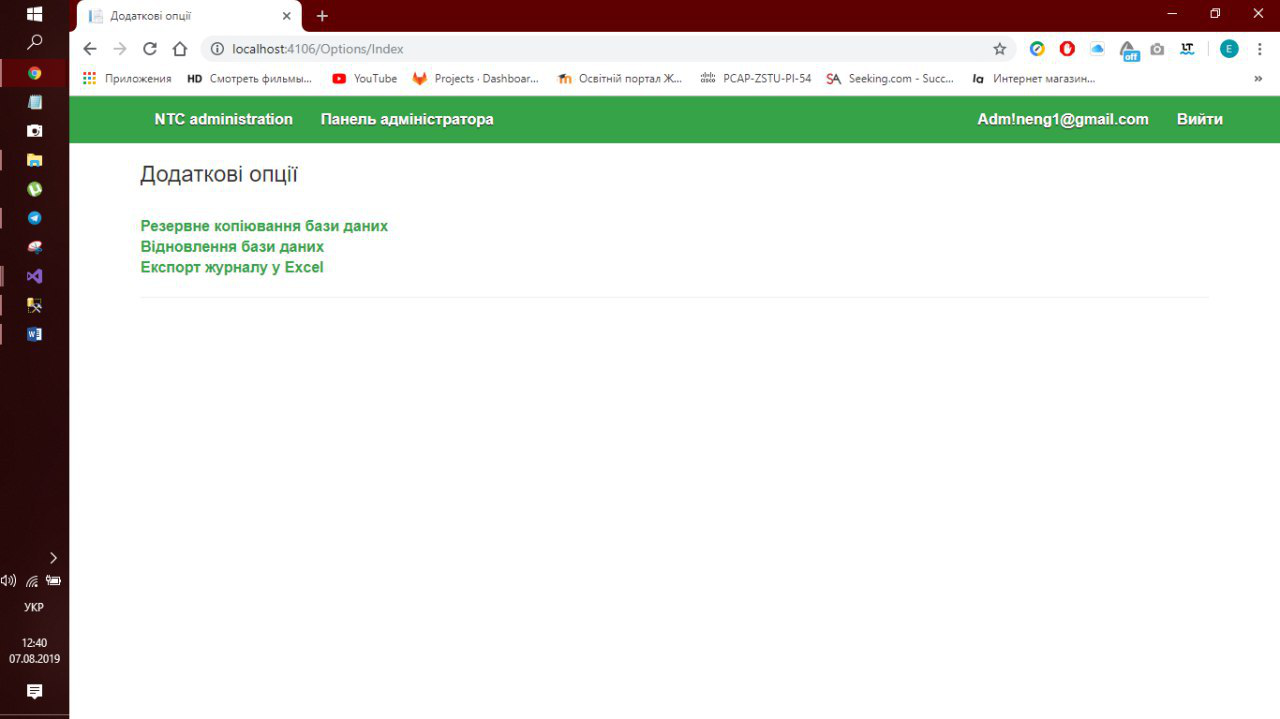


Рис. 3.14 Вигляд сторінки додаткових опцій

Рис. 3.15Повідомлення по успішне виконання операції резервного копіювання

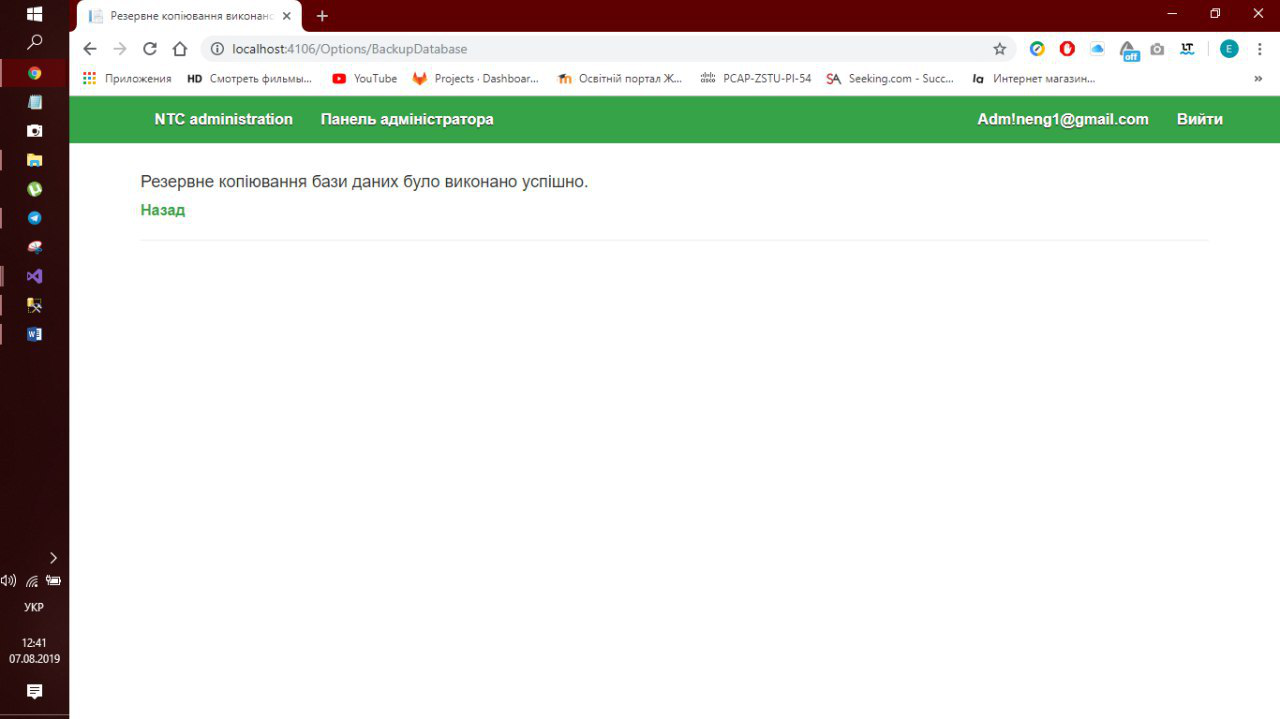


Рис. 3.16Результат виконання операції резервного копіювання

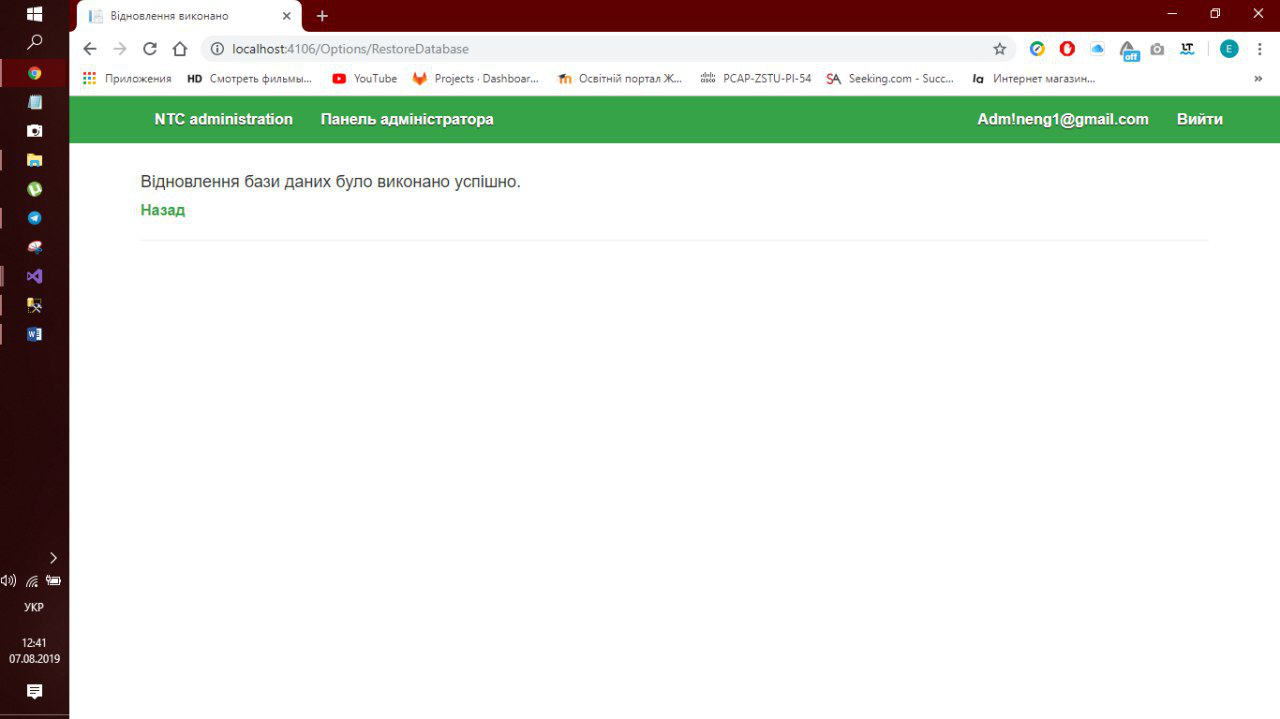


Рис. 3.15Повідомлення пpо успішне виконання операції відновлення

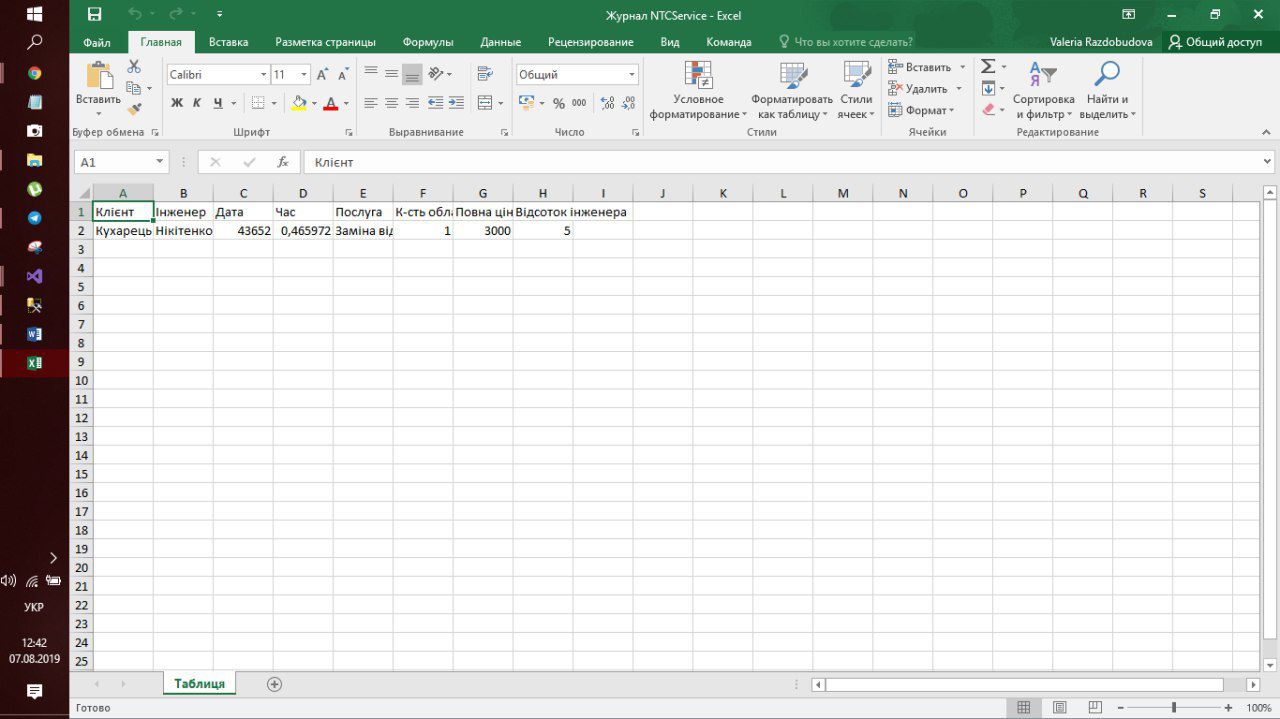


Рис. 3.16Результат виконання операції еспорту даних в Excel

## **4.2 Налаштування параметрів роботи SQL-сервера**

Для того, щоб наша БД для сервісного центрузмогла працювати на іншому комп’ютері потрібно встановити Microsoft SQL Server.

Після встановлення Microsoft SQL Server є декілька способів перенесення БД на інший комп’ютер. Деякі з цих варіантів приведені нижче.

Як варіант, ми можемо виконати такі дії:

1. На вихідному комп'ютері збережіть файли бази даних;
2. На вихідному комп'ютері від'єднайте базу від екземпляра SQL Server вихідного комп'ютера;
3. Перемістіть файли бази даних у відповідні каталоги на цільовому комп'ютері;
4. На екземплярі SQL Server цільового комп'ютера приєднайте базу даних;

Іншим варіантом є створити резервну копію бази даних на вихідному комп'ютері, а потім відновити її у примірнику SQL Server на цільовому комп'ютері.

База даних може бути використана, коли резервна копія працює. Якщо користувачі виконують оператори INSERT, UPDATE або DELETE в базі даних після завершення резервної копії, резервна копія не містить цих змін. Якщо ви повинні перенести всі зміни, ви можете перенести ці зміни з мінімальним простотою, якщо ви виконуєте як резервну копію журналу транзакцій, так і повну резервну копію бази даних.

Відновіть повне резервне копіювання бази даних на цільовому сервері та вкажіть параметр WITH NORECOVERY.

Щоб запобігти додатковим змінам бази даних, прямі користувачі повинні припинити діяльність бази даних на вихідному сервері.

Ще одним способом є використання збережених процедур sp\_detach\_db та sp\_attach\_db.

Щоб використовувати збережені процедури sp\_detach\_db та sp\_attach\_db, виконайте такі дії:

Від'єднати базу даних на вихідному сервері, використовуючи збережену процедуру sp\_detach\_db. Ми повинні скопіювати файли .mdf, .ndf та .ldf, пов'язані з базою даних, до цільового сервера. Ознайомтеся з цією таблицею для опису типів файлів:

Таблиця 15

**Опис типів файлів**

|  |  |
| --- | --- |
| **Розширення назви файлу** | **Опис** |
| .mdf | Основний файл даних |
| .ndf | Додатковий файл даних |
| .ldf | Файл журналу транзакцій |

Потрібно прикріпити базу даних на цільовому сервері, використовуючи збережену процедуру sp\_attach\_db, і вказати файли, які ми скопіювали на цільовий сервер на попередньому кроці.

ВИСНОВКИ

В першому розділі курсової роботи було сформовано уявлення про БД, обгрунтовано причину розробки БД для cервісногоцентру. Після порівняння декількох СУБД, було обрано СУБД MSSQL. Ця СУБД передбачає, що необхідний рівень кваліфікованості персоналу дорівнює середньому рівню. Модель даних є клієнт – серверною.В загальному, масштаб застосування цієї СУБД є для малих та середніх підприємств

В другому розділі курсової роботи було проведено проектування БД. При проектуванні БД було визначено, що для повного функціонування БД потрібно 11таблиць, які містять всю інформацію про клієнтів та персонал cервісного центру.

В третьому розділі курсової роботи було спроектовано та реалізовано інтерфейс обробки даних. Приведено приклади процедур, триггерів та LINQзапитів. Також було організовано звітність системи. В кінці розділу описано роботу зі створеною програмою та проведено тестування створеного додатку.

В четвертому розділі курсової роботи було приведено таблицю доступу до даних, додаткові опції адміністрування, такі як резервне копіювання бази даних, відновлення бази даних і експорт даних в Excelта приведені декілька варіантів, за допомогою яких БД для cервісного центру зможе функціонувати на іншому комп’ютері.

В результаті виконання курсової роботиотримано БД та ПЗ, які по функціональності повністю відповідають поставленим на початку проектування вимогам. Додаток є простим у користуванні з інтуїтивно зрозумілим україномовним інтерфейсом.

Програмне забезпечення представляє собою графічнийдодаток для зручного адміністрування та користування БД для cервісного центру. При запусканні якого з’являлось головне вікно сайту, в якому необхідно було лише обирати свої подальші дії.

# ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Атре Ш. Структурный подход к организации баз данных. - М.: Финансы и статистика, 2005. - 320 с.
2. Бураков П.В. Введение в системы баз данных. Учебное пособие / П.В. Бураков., В.Ю. Петров. - СПбГУ ИТМО, 2010. 129 с.
3. Горнаков С. Г., Осваиваем популярные системы управления сайтом (CMS). – М.: ДМК Пресс, 2009. – 336 с.
4. Дейт К. Введение в системы баз даных /К.Дейт// – К.; М.; СПб.: Изд.дом “Вильямс”. – 2000. – С. 560.
5. Квентин Зервас. Web 2.0: создание приложений на С# = Practical Web 2.0 Applications with C#. —М.:«Вильямс», 2009. — С. 544.
6. Кузнецов Максим, Симдянов Игорь. PHP на примерах. — 2-е изд. перераб. и доп. — Спб.: «БХВ-Петербург», 2011. — С. 400.
7. Кузнєцов М., Симдянов І. Голишев С. PHP 5. Практика створення Web-сайтів. – К., 2007.
8. Купер Алан. Алан Купер об интерфейсе. Основы проектирования взаимодействия. – «Символ-Плюс», 2017. – С.688
9. Куправа Т. А. Создание и программирование баз данных средствами СУБД. - М.: Мир, 1999. - 109 с.
10. Павловская Т.А. C#. Программирование на языке высокого уровня: Учебник для вузов / Т.А. Павловская. - Питер, 2009. - 432 с.
11. Петлюшкин А.В., HTML в Web-дизайне. – СПб.: БВХ-Петербург, 2004. – С. 400.
12. Севостьянов И.О., Поисковая оптимизация. Практическое руководство по продвижению сайта в Интернете / И.О. Севостьянов. – СПб.: Питер, 2010. – С. 240.

# ДОДАТКИ

ДОДАТКИ

Додаток А

**Лістинг програмного коду «AccountController.cs»**

using System;

using System.Globalization;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using System.Web;

using System.Web.Mvc;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin.Security;

using NTCService.Models;

namespace NTCService.Controllers

{

[Authorize]

publicclassAccountController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

public AccountController()

{

}

public AccountController(ApplicationUserManager userManager, ApplicationSignInManager signInManager)

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

return \_signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();

}

privateset

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

privateset

{

\_userManager = value;

}

}

//

// GET: /Account/Login

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Login(LoginViewModel model, string returnUrl)

{

if (!ModelState.IsValid)

{

return View(model);

}

// Сбои при входе не приводят к блокированию учетной записи

// Чтобы ошибки при вводе пароля инициировали блокирование учетной записи, замените на shouldLockout: true

var result = await SignInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, shouldLockout: false);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(returnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.RequiresVerification:

return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = model.RememberMe });

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Неудачнаяпопыткавхода.");

return View(model);

}

}

//

// GET: /Account/VerifyCode

[AllowAnonymous]

publicasync Task<ActionResult> VerifyCode(string provider, string returnUrl, bool rememberMe)

{

// Требовать предварительный вход пользователя с помощью имени пользователя и пароля или внешнего имени входа

if (!await SignInManager.HasBeenVerifiedAsync())

{

return View("Error");

}

return View(new VerifyCodeViewModel { Provider = provider, ReturnUrl = returnUrl, RememberMe = rememberMe });

}

//

// POST: /Account/VerifyCode

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> VerifyCode(VerifyCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

// Приведенный ниже код защищает от атак методом подбора, направленных на двухфакторные коды.

// Если пользователь введет неправильные коды за указанное время, его учетная запись

// будет заблокирована на заданный период.

// Параметры блокирования учетных записей можно настроить в IdentityConfig

var result = await SignInManager.TwoFactorSignInAsync(model.Provider, model.Code, isPersistent: model.RememberMe, rememberBrowser: model.RememberBrowser);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(model.ReturnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Неправильныйкод.");

return View(model);

}

}

//

// GET: /Account/Register

[AllowAnonymous]

public ActionResult Register()

{

return View();

}

//

// POST: /Account/Register

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

var result = await UserManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

// Дополнительные сведения о включении подтверждения учетной записи и сброса пароля см. на странице https://go.microsoft.com/fwlink/?LinkID=320771.

// Отправка сообщения электронной почты с этой ссылкой

// string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);

// var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Подтверждениеучетнойзаписи", "Подтвердитевашуучетнуюзапись, щелкнув<a href=\"" + callbackUrl + "\">здесь</a>");

return RedirectToAction("Index", "Home");

}

AddErrors(result);

}

// Появление этого сообщения означает наличие ошибки; повторное отображение формы

return View(model);

}

//

// GET: /Account/ConfirmEmail

[AllowAnonymous]

publicasync Task<ActionResult> ConfirmEmail(string userId, string code)

{

if (userId == null || code == null)

{

return View("Error");

}

var result = await UserManager.ConfirmEmailAsync(userId, code);

return View(result.Succeeded ? "ConfirmEmail" : "Error");

}

//

// GET: /Account/ForgotPassword

[AllowAnonymous]

public ActionResult ForgotPassword()

{

return View();

}

//

// POST: /Account/ForgotPassword

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> ForgotPassword(ForgotPasswordViewModel model)

{

if (ModelState.IsValid)

{

var user = await UserManager.FindByNameAsync(model.Email);

if (user == null || !(await UserManager.IsEmailConfirmedAsync(user.Id)))

{

// Не показывать, что пользователь не существует или не подтвержден

return View("ForgotPasswordConfirmation");

}

// Дополнительные сведения о включении подтверждения учетной записи и сброса пароля см. на странице https://go.microsoft.com/fwlink/?LinkID=320771.

// Отправка сообщения электронной почты с этой ссылкой

// string code = await UserManager.GeneratePasswordResetTokenAsync(user.Id);

// var callbackUrl = Url.Action("ResetPassword", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Сброспароля", "Сбросьтевашпароль, щелкнув<a href=\"" + callbackUrl + "\">здесь</a>");

// return RedirectToAction("ForgotPasswordConfirmation", "Account");

}

// Появлениеэтогосообщенияозначаетналичиеошибки; повторноеотображениеформы

return View(model);

}

//

// GET: /Account/ForgotPasswordConfirmation

[AllowAnonymous]

public ActionResult ForgotPasswordConfirmation()

{

return View();

}

//

// GET: /Account/ResetPassword

[AllowAnonymous]

public ActionResult ResetPassword(string code)

{

return code == null ? View("Error") : View();

}

//

// POST: /Account/ResetPassword

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> ResetPassword(ResetPasswordViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var user = await UserManager.FindByNameAsync(model.Email);

if (user == null)

{

// Не показывать, что пользователь не существует

return RedirectToAction("ResetPasswordConfirmation", "Account");

}

var result = await UserManager.ResetPasswordAsync(user.Id, model.Code, model.Password);

if (result.Succeeded)

{

return RedirectToAction("ResetPasswordConfirmation", "Account");

}

AddErrors(result);

return View();

}

//

// GET: /Account/ResetPasswordConfirmation

[AllowAnonymous]

public ActionResult ResetPasswordConfirmation()

{

return View();

}

//

// POST: /Account/ExternalLogin

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLogin(string provider, string returnUrl)

{

// Запрос перенаправления к внешнему поставщику входа

returnnew ChallengeResult(provider, Url.Action("ExternalLoginCallback", "Account", new { ReturnUrl = returnUrl }));

}

//

// GET: /Account/SendCode

[AllowAnonymous]

publicasync Task<ActionResult> SendCode(string returnUrl, bool rememberMe)

{

var userId = await SignInManager.GetVerifiedUserIdAsync();

if (userId == null)

{

return View("Error");

}

var userFactors = await UserManager.GetValidTwoFactorProvidersAsync(userId);

var factorOptions = userFactors.Select(purpose =>new SelectListItem { Text = purpose, Value = purpose }).ToList();

return View(new SendCodeViewModel { Providers = factorOptions, ReturnUrl = returnUrl, RememberMe = rememberMe });

}

//

// POST: /Account/SendCode

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> SendCode(SendCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View();

}

// Созданиеиотправкамаркера

if (!await SignInManager.SendTwoFactorCodeAsync(model.SelectedProvider))

{

return View("Error");

}

return RedirectToAction("VerifyCode", new { Provider = model.SelectedProvider, ReturnUrl = model.ReturnUrl, RememberMe = model.RememberMe });

}

//

// GET: /Account/ExternalLoginCallback

[AllowAnonymous]

publicasync Task<ActionResult> ExternalLoginCallback(string returnUrl)

{

var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync();

if (loginInfo == null)

{

return RedirectToAction("Login");

}

// Выполнение входа пользователя посредством данного внешнего поставщика входа, если у пользователя уже есть имя входа

var result = await SignInManager.ExternalSignInAsync(loginInfo, isPersistent: false);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(returnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.RequiresVerification:

return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = false });

case SignInStatus.Failure:

default:

// Если у пользователя нет учетной записи, то ему предлагается создать ее

ViewBag.ReturnUrl = returnUrl;

ViewBag.LoginProvider = loginInfo.Login.LoginProvider;

return View("ExternalLoginConfirmation", new ExternalLoginConfirmationViewModel { Email = loginInfo.Email });

}

}

//

// POST: /Account/ExternalLoginConfirmation

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> ExternalLoginConfirmation(ExternalLoginConfirmationViewModel model, string returnUrl)

{

if (User.Identity.IsAuthenticated)

{

return RedirectToAction("Index", "Manage");

}

if (ModelState.IsValid)

{

// Получение сведений о пользователе от внешнего поставщика входа

var info = await AuthenticationManager.GetExternalLoginInfoAsync();

if (info == null)

{

return View("ExternalLoginFailure");

}

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

var result = await UserManager.CreateAsync(user);

if (result.Succeeded)

{

result = await UserManager.AddLoginAsync(user.Id, info.Login);

if (result.Succeeded)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

return RedirectToLocal(returnUrl);

}

}

AddErrors(result);

}

ViewBag.ReturnUrl = returnUrl;

return View(model);

}

//

// POST: /Account/LogOff

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);

return RedirectToAction("Index", "Home");

}

//

// GET: /Account/ExternalLoginFailure

[AllowAnonymous]

public ActionResult ExternalLoginFailure()

{

return View();

}

protectedoverridevoid Dispose(bool disposing)

{

if (disposing)

{

if (\_userManager != null)

{

\_userManager.Dispose();

\_userManager = null;

}

if (\_signInManager != null)

{

\_signInManager.Dispose();

\_signInManager = null;

}

}

base.Dispose(disposing);

}

#region Вспомогательные приложения

// Используется для защиты от XSRF-атак при добавлении внешних имен входа

privateconststring XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager

{

get

{

return HttpContext.GetOwinContext().Authentication;

}

}

privatevoid AddErrors(IdentityResult result)

{

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error);

}

}

private ActionResult RedirectToLocal(string returnUrl)

{

if (Url.IsLocalUrl(returnUrl))

{

return Redirect(returnUrl);

}

return RedirectToAction("Index", "Home");

}

internalclassChallengeResult : HttpUnauthorizedResult

{

public ChallengeResult(string provider, string redirectUri)

: this(provider, redirectUri, null)

{

}

public ChallengeResult(string provider, string redirectUri, string userId)

{

LoginProvider = provider;

RedirectUri = redirectUri;

UserId = userId;

}

publicstring LoginProvider { get; set; }

publicstring RedirectUri { get; set; }

publicstring UserId { get; set; }

publicoverridevoid ExecuteResult(ControllerContext context)

{

var properties = new AuthenticationProperties { RedirectUri = RedirectUri };

if (UserId != null)

{

properties.Dictionary[XsrfKey] = UserId;

}

context.HttpContext.GetOwinContext().Authentication.Challenge(properties, LoginProvider);

}

}

#endregion

}

}

Додаток Б

**Лістинг програмного коду «HomeController.cs»**

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Web;

using System.Web.Mvc;

using NTCService;

using PagedList;

namespace NTCService.Controllers

{

publicclassHomeController : Controller

{

private ModelDB db = new ModelDB();

public ActionResult Index()

{

return View();

}

public ActionResult TypesOfServices(string sortOrder, string SearchByName, string currentFilter, int? page)

{

ViewBag.CurrentSort = sortOrder;

ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

if (SearchByName != null)

{

page = 1;

}

else

{

SearchByName = currentFilter;

}

ViewBag.CurrentFilter = SearchByName;

var typesServices = from s in db.TypesNTCServices select s;

if (!String.IsNullOrEmpty(SearchByName))

{

typesServices = typesServices.Where(s => s.name\_type\_NTCservice.Contains(SearchByName));

}

switch (sortOrder)

{

case"name\_desc":

typesServices = typesServices.OrderByDescending(s => s.name\_type\_NTCservice);

break;

default:

typesServices = typesServices.OrderBy(s => s.name\_type\_NTCservice);

break;

}

int pageSize = 8;

int pageNumber = (page ?? 1);

return View(typesServices.ToPagedList(pageNumber, pageSize));

}

public ActionResult Services(string sortOrder, string SearchByName, string SearchByTypeNTCService, string currentFilter,

int? page)

{

ViewBag.CurrentSort = sortOrder;

ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

ViewBag.PriceSortParm = sortOrder == "price" ? "price\_desc" : "price";

ViewBag.TypeServiceSortParm = sortOrder == "name\_type" ? "name\_type\_desc" : "name\_type";

ViewBag.EquipmentInStockrSortParm = sortOrder == "eq\_in\_stock" ? "eq\_in\_stock\_desc" : "eq\_in\_stock";

if (SearchByName != null)

{

page = 1;

}

else

{

SearchByName = currentFilter;

}

if (SearchByTypeNTCService != null)

{

page = 1;

}

else

{

SearchByTypeNTCService = currentFilter;

}

ViewBag.CurrentFilter = SearchByName;

ViewBag.CurrentFilter = SearchByTypeNTCService;

var services = db.NTCServices.Include(n => n.NTCequipmentInStock).Include(n => n.TypesNTCServices);

if (!String.IsNullOrEmpty(SearchByName))

{

services = services.Where(s => s.name\_NTCservice.Contains(SearchByName));

}

if (!String.IsNullOrEmpty(SearchByTypeNTCService))

{

services = services.Where(s => s.TypesNTCServices.name\_type\_NTCservice.Contains(SearchByTypeNTCService));

}

switch (sortOrder)

{

case"name\_desc":

services = services.OrderByDescending(s => s.name\_NTCservice);

break;

case"price":

services = services.OrderBy(s => s.price\_NTCservice);

break;

case"price\_desc":

services = services.OrderByDescending(s => s.price\_NTCservice);

break;

case"name\_type":

services = services.OrderBy(s => s.TypesNTCServices.name\_type\_NTCservice);

break;

case"name\_type\_desc":

services = services.OrderByDescending(s => s.TypesNTCServices.name\_type\_NTCservice);

break;

case"eq\_in\_stock":

services = services.OrderBy(s => s.NTCequipmentInStock.name\_NTCequipment);

break;

case"eq\_in\_stock\_desc":

services = services.OrderByDescending(s => s.NTCequipmentInStock.name\_NTCequipment);

break;

default:

services = services.OrderBy(s => s.name\_NTCservice);

break;

}int pageSize = 8;

int pageNumber = (page ?? 1);

return View(services.ToPagedList(pageNumber, pageSize));

}

public ActionResult Contact()

{

ViewBag.Message = "Your contact page.";

return View();

}

[Authorize]

public ActionResult Application()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Application([Bind(Include = "id\_application,phone\_numb,mail")] Application application)

{

if (ModelState.IsValid)

{

db.Application.Add(application);

db.SaveChanges();

return RedirectToAction("Sent", "Home");

}

return View(application);

}

public ActionResult Sent()

{

return View();

}

}

}

Додаток В

**Лістинг програмного коду «ManageController.cs»**

using System;

using System.Linq;

using System.Threading.Tasks;

using System.Web;

using System.Web.Mvc;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin.Security;

using NTCService.Models;

namespace NTCService.Controllers

{

[Authorize]

publicclassManageController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

public ManageController()

{

}

public ManageController(ApplicationUserManager userManager, ApplicationSignInManager signInManager)

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

return \_signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();

}

privateset

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

privateset

{

\_userManager = value;

}

}

//

// GET: /Manage/Index

publicasync Task<ActionResult> Index(ManageMessageId? message)

{

ViewBag.StatusMessage =

message == ManageMessageId.ChangePasswordSuccess ? "Вашпарольизменен."

: message == ManageMessageId.SetPasswordSuccess ? "Парользадан."

: message == ManageMessageId.SetTwoFactorSuccess ? "Настроенпоставщикдвухфакторнойпроверкиподлинности."

: message == ManageMessageId.Error ? "Произошлаошибка."

: message == ManageMessageId.AddPhoneSuccess ? "Вашномертелефонадобавлен."

: message == ManageMessageId.RemovePhoneSuccess ? "Вашномертелефонаудален."

: "";

var userId = User.Identity.GetUserId();

var model = new IndexViewModel

{

HasPassword = HasPassword(),

PhoneNumber = await UserManager.GetPhoneNumberAsync(userId),

TwoFactor = await UserManager.GetTwoFactorEnabledAsync(userId),

Logins = await UserManager.GetLoginsAsync(userId),

BrowserRemembered = await AuthenticationManager.TwoFactorBrowserRememberedAsync(userId)

};

return View(model);

}

//

// POST: /Manage/RemoveLogin

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> RemoveLogin(string loginProvider, string providerKey)

{

ManageMessageId? message;

var result = await UserManager.RemoveLoginAsync(User.Identity.GetUserId(), new UserLoginInfo(loginProvider, providerKey));

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

message = ManageMessageId.RemoveLoginSuccess;

}

else

{

message = ManageMessageId.Error;

}

return RedirectToAction("ManageLogins", new { Message = message });

}

//

// GET: /Manage/AddPhoneNumber

public ActionResult AddPhoneNumber()

{

return View();

}

//

// POST: /Manage/AddPhoneNumber

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> AddPhoneNumber(AddPhoneNumberViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

// Созданиеиотправкамаркера

var code = await UserManager.GenerateChangePhoneNumberTokenAsync(User.Identity.GetUserId(), model.Number);

if (UserManager.SmsService != null)

{

var message = new IdentityMessage

{

Destination = model.Number,

Body = "Вашкодбезопасности: " + code

};

await UserManager.SmsService.SendAsync(message);

}

return RedirectToAction("VerifyPhoneNumber", new { PhoneNumber = model.Number });

}

//

// POST: /Manage/EnableTwoFactorAuthentication

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> EnableTwoFactorAuthentication()

{

await UserManager.SetTwoFactorEnabledAsync(User.Identity.GetUserId(), true);

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", "Manage");

}

//

// POST: /Manage/DisableTwoFactorAuthentication

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> DisableTwoFactorAuthentication()

{

await UserManager.SetTwoFactorEnabledAsync(User.Identity.GetUserId(), false);

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", "Manage");

}

//

// GET: /Manage/VerifyPhoneNumber

publicasync Task<ActionResult> VerifyPhoneNumber(string phoneNumber)

{

var code = await UserManager.GenerateChangePhoneNumberTokenAsync(User.Identity.GetUserId(), phoneNumber);

// Отправка SMS через поставщик SMS для проверки номера телефона

return phoneNumber == null ? View("Error") : View(new VerifyPhoneNumberViewModel { PhoneNumber = phoneNumber });

}

//

// POST: /Manage/VerifyPhoneNumber

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> VerifyPhoneNumber(VerifyPhoneNumberViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var result = await UserManager.ChangePhoneNumberAsync(User.Identity.GetUserId(), model.PhoneNumber, model.Code);

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.AddPhoneSuccess });

}

// Это сообщение означает наличие ошибки; повторное отображение формы

ModelState.AddModelError("", "Не удалось проверить телефон");

return View(model);

}

//

// POST: /Manage/RemovePhoneNumber

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> RemovePhoneNumber()

{

var result = await UserManager.SetPhoneNumberAsync(User.Identity.GetUserId(), null);

if (!result.Succeeded)

{

return RedirectToAction("Index", new { Message = ManageMessageId.Error });

}

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.RemovePhoneSuccess });

}

//

// GET: /Manage/ChangePassword

public ActionResult ChangePassword()

{

return View();

}

//

// POST: /Manage/ChangePassword

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> ChangePassword(ChangePasswordViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var result = await UserManager.ChangePasswordAsync(User.Identity.GetUserId(), model.OldPassword, model.NewPassword);

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.ChangePasswordSuccess });

}

AddErrors(result);

return View(model);

}

//

// GET: /Manage/SetPassword

public ActionResult SetPassword()

{

return View();

}

//

// POST: /Manage/SetPassword

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> SetPassword(SetPasswordViewModel model)

{

if (ModelState.IsValid)

{

var result = await UserManager.AddPasswordAsync(User.Identity.GetUserId(), model.NewPassword);

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.SetPasswordSuccess });

}

AddErrors(result);

}

// Это сообщение означает наличие ошибки; повторное отображение формы

return View(model);

}

//

// GET: /Manage/ManageLogins

publicasync Task<ActionResult> ManageLogins(ManageMessageId? message)

{

ViewBag.StatusMessage =

message == ManageMessageId.RemoveLoginSuccess ? "Внешнееимявходаудалено."

: message == ManageMessageId.Error ? "Произошлаошибка."

: "";

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user == null)

{

return View("Error");

}

var userLogins = await UserManager.GetLoginsAsync(User.Identity.GetUserId());

var otherLogins = AuthenticationManager.GetExternalAuthenticationTypes().Where(auth => userLogins.All(ul => auth.AuthenticationType != ul.LoginProvider)).ToList();

ViewBag.ShowRemoveButton = user.PasswordHash != null || userLogins.Count > 1;

return View(new ManageLoginsViewModel

{

CurrentLogins = userLogins,

OtherLogins = otherLogins

});

}

//

// POST: /Manage/LinkLogin

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LinkLogin(string provider)

{

// Запрос перенаправления к внешнему поставщику входа для связывания имени входа текущего пользователя

returnnew AccountController.ChallengeResult(provider, Url.Action("LinkLoginCallback", "Manage"), User.Identity.GetUserId());

}

//

// GET: /Manage/LinkLoginCallback

publicasync Task<ActionResult> LinkLoginCallback()

{

var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync(XsrfKey, User.Identity.GetUserId());

if (loginInfo == null)

{

return RedirectToAction("ManageLogins", new { Message = ManageMessageId.Error });

}

var result = await UserManager.AddLoginAsync(User.Identity.GetUserId(), loginInfo.Login);

return result.Succeeded ? RedirectToAction("ManageLogins") : RedirectToAction("ManageLogins", new { Message = ManageMessageId.Error });

}

protectedoverridevoid Dispose(bool disposing)

{

if (disposing && \_userManager != null)

{

\_userManager.Dispose();

\_userManager = null;

}

base.Dispose(disposing);

}

#region Вспомогательные приложения

// Используется для защиты от XSRF-атак при добавлении внешних имен входа

privateconststring XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager

{

get

{

return HttpContext.GetOwinContext().Authentication;

}

}

privatevoid AddErrors(IdentityResult result)

{

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error);

}

}

privatebool HasPassword()

{

var user = UserManager.FindById(User.Identity.GetUserId());

if (user != null)

{

return user.PasswordHash != null;

}

returnfalse;

}

privatebool HasPhoneNumber()

{

var user = UserManager.FindById(User.Identity.GetUserId());

if (user != null)

{

return user.PhoneNumber != null;

}

returnfalse;

}

publicenumManageMessageId

{

AddPhoneSuccess,

ChangePasswordSuccess,

SetTwoFactorSuccess,

SetPasswordSuccess,

RemoveLoginSuccess,

RemovePhoneSuccess,

Error

}

#endregion

}

}

Додаток Г

**Лістинг програмного коду «TypesNTCequipmentController.cs»**

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Web;

using System.Web.Mvc;

using NTCService;

using PagedList;

namespace NTCService.Controllers

{

[Authorize(Roles = "Admin")]

publicclassTypesNTCequipmentController : Controller

{

private ModelDB db = new ModelDB();

public ActionResult Index(string sortOrder, string SearchByName, string currentFilter, int? page)

{

ViewBag.CurrentSort = sortOrder;

ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

if (SearchByName != null)

{

page = 1;

}

else

{

SearchByName = currentFilter;

}

ViewBag.CurrentFilter = SearchByName;

var typesEquipment = from s in db.TypesNTCequipment select s;

if (!String.IsNullOrEmpty(SearchByName))

{

typesEquipment = typesEquipment.Where(s => s.name\_type\_NTCequipment.Contains(SearchByName));

}

switch (sortOrder)

{

case"name\_desc":

typesEquipment = typesEquipment.OrderByDescending(s => s.name\_type\_NTCequipment);

break;

default:

typesEquipment = typesEquipment.OrderBy(s => s.name\_type\_NTCequipment);

break;

}

int pageSize = 8;

int pageNumber = (page ?? 1);

return View(typesEquipment.ToPagedList(pageNumber, pageSize));

}

publicasync Task<ActionResult> Details(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

TypesNTCequipment typesNTCequipment = await db.TypesNTCequipment.FindAsync(id);

if (typesNTCequipment == null)

{

return HttpNotFound();

}

return View(typesNTCequipment);

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Create([Bind(Include = "id\_type\_NTCequipment,name\_type\_NTCequipment")]

TypesNTCequipment typesNTCequipment)

{

if (ModelState.IsValid)

{

db.TypesNTCequipment.Add(typesNTCequipment);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(typesNTCequipment);

}

publicasync Task<ActionResult> Edit(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

TypesNTCequipment typesNTCequipment = await db.TypesNTCequipment.FindAsync(id);

if (typesNTCequipment == null)

{

return HttpNotFound();

}

return View(typesNTCequipment);

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Edit([Bind(Include = "id\_type\_NTCequipment,name\_type\_NTCequipment")]

TypesNTCequipment typesNTCequipment)

{

if (ModelState.IsValid)

{

db.Entry(typesNTCequipment).State = EntityState.Modified;

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(typesNTCequipment);

}

publicasync Task<ActionResult> Delete(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

TypesNTCequipment typesNTCequipment = await db.TypesNTCequipment.FindAsync(id);

if (typesNTCequipment == null)

{

return HttpNotFound();

}

return View(typesNTCequipment);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> DeleteConfirmed(int id)

{

TypesNTCequipment typesNTCequipment = await db.TypesNTCequipment.FindAsync(id);

db.TypesNTCequipment.Remove(typesNTCequipment);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

protectedoverridevoid Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Додаток Д

**Лістинг програмного коду «NTCequipmentProvidersController.cs»**

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Web;

using System.Web.Mvc;

using NTCService;

using PagedList;

namespace NTCService.Controllers

{

[Authorize(Roles = "Admin")]

publicclassNTCequipmentProvidersController : Controller

{

private ModelDB db = new ModelDB();

public ActionResult Index(string sortOrder, string SearchByName, string currentFilter, int? page)

{

ViewBag.CurrentSort = sortOrder;

ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

ViewBag.PhoneNumbSortParm = sortOrder == "phone\_numb" ? "phone\_numb\_desc" : "phone\_numb";

ViewBag.WebsiteSortParm = sortOrder == "website" ? "website\_desc" : "website";

ViewBag.EmailSortParm = sortOrder == "email" ? "email\_desc" : "email";

ViewBag.AddressSortParm = sortOrder == "address" ? "address\_desc" : "address";

if (SearchByName != null)

{

page = 1;

}

else

{

SearchByName = currentFilter;

}

ViewBag.CurrentFilter = SearchByName;

var equipmentProviders = from s in db.NTCequipmentProviders select s;

if (!String.IsNullOrEmpty(SearchByName))

{

equipmentProviders = equipmentProviders.Where(s => s.name\_NTCprovider.Contains(SearchByName));

}

switch (sortOrder)

{

case"name\_desc":

equipmentProviders = equipmentProviders.OrderByDescending(s => s.name\_NTCprovider);

break;

case"phone\_numb":

equipmentProviders = equipmentProviders.OrderByDescending(s => s.phone\_number\_NTCprovider);

break;

case"phone\_numb\_desc":

equipmentProviders = equipmentProviders.OrderByDescending(s => s.phone\_number\_NTCprovider);

break;

case"website":

equipmentProviders = equipmentProviders.OrderByDescending(s => s.website\_NTCprovider);

break;

case"website\_desc":

equipmentProviders = equipmentProviders.OrderByDescending(s => s.website\_NTCprovider);

break;

case"email":

equipmentProviders = equipmentProviders.OrderByDescending(s => s.email\_NTCprovider);

break;

case"email\_desc":

equipmentProviders = equipmentProviders.OrderByDescending(s => s.email\_NTCprovider);

break;

case"address":

equipmentProviders = equipmentProviders.OrderByDescending(s => s.address\_NTCprovider);

break;

case"address\_desc":

equipmentProviders = equipmentProviders.OrderByDescending(s => s.address\_NTCprovider);

break;

default:

equipmentProviders = equipmentProviders.OrderBy(s => s.name\_NTCprovider);

break;

}

int pageSize = 8;

int pageNumber = (page ?? 1);

return View(equipmentProviders.ToPagedList(pageNumber, pageSize));

}

publicasync Task<ActionResult> Details(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCequipmentProviders nTCequipmentProviders = await db.NTCequipmentProviders.FindAsync(id);

if (nTCequipmentProviders == null)

{

return HttpNotFound();

}

return View(nTCequipmentProviders);

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Create([Bind(Include = "id\_NTCprovider,name\_NTCprovider,phone\_number\_NTCprovider," +

"website\_NTCprovider,email\_NTCprovider,address\_NTCprovider")] NTCequipmentProviders nTCequipmentProviders)

{

if (ModelState.IsValid)

{

db.NTCequipmentProviders.Add(nTCequipmentProviders);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(nTCequipmentProviders);

}

publicasync Task<ActionResult> Edit(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCequipmentProviders nTCequipmentProviders = await db.NTCequipmentProviders.FindAsync(id);

if (nTCequipmentProviders == null)

{

return HttpNotFound();

}

return View(nTCequipmentProviders);

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Edit([Bind(Include = "id\_NTCprovider,name\_NTCprovider,phone\_number\_NTCprovider," +

"website\_NTCprovider,email\_NTCprovider,address\_NTCprovider")] NTCequipmentProviders nTCequipmentProviders)

{

if (ModelState.IsValid)

{

db.Entry(nTCequipmentProviders).State = EntityState.Modified;

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(nTCequipmentProviders);

}

publicasync Task<ActionResult> Delete(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCequipmentProviders nTCequipmentProviders = await db.NTCequipmentProviders.FindAsync(id);

if (nTCequipmentProviders == null)

{

return HttpNotFound();

}

return View(nTCequipmentProviders);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> DeleteConfirmed(int id)

{

NTCequipmentProviders nTCequipmentProviders = await db.NTCequipmentProviders.FindAsync(id);

db.NTCequipmentProviders.Remove(nTCequipmentProviders);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

protectedoverridevoid Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Додаток Е

**Лістинг програмного коду «NTCequipmentInStockController.cs»**

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Web;

using System.Web.Mvc;

using NTCService;

using PagedList;

namespace NTCService.Controllers

{

[Authorize(Roles = "Admin")]

publicclassNTCequipmentInStockController : Controller

{

private ModelDB db = new ModelDB();

public ActionResult Index(string sortOrder, string SearchByName, string SearchByTypeNTCService,

string SearchByNTCequipmentProviders, string currentFilter, int? page)

{

ViewBag.CurrentSort = sortOrder;

ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

ViewBag.TypeServiceSortParm = sortOrder == "name\_type" ? "name\_type\_desc" : "name\_type";

ViewBag.EquipmentProviderSortParm = sortOrder == "eq\_provider" ? "eq\_provider\_desc" : "eq\_provider";

ViewBag.AmountSortParm = sortOrder == "amount" ? "amount\_desc" : "amount";

if (SearchByName != null)

{

page = 1;

}

else

{

SearchByName = currentFilter;

}

if (SearchByTypeNTCService != null)

{

page = 1;

}

else

{

SearchByTypeNTCService = currentFilter;

}

if (SearchByNTCequipmentProviders != null)

{

page = 1;

}

else

{

SearchByNTCequipmentProviders = currentFilter;

}

ViewBag.CurrentFilter = SearchByName;

ViewBag.CurrentFilter = SearchByTypeNTCService;

ViewBag.CurrentFilter = SearchByNTCequipmentProviders;

var equipmentInStock = db.NTCequipmentInStock.Include(n => n.NTCequipmentProviders).Include(n => n.TypesNTCequipment);

if (!String.IsNullOrEmpty(SearchByName))

{

equipmentInStock = equipmentInStock.Where(s => s.name\_NTCequipment.Contains(SearchByName));

}

if (!String.IsNullOrEmpty(SearchByTypeNTCService))

{

equipmentInStock = equipmentInStock.Where(s => s.TypesNTCequipment.name\_type\_NTCequipment.

Contains(SearchByTypeNTCService));

}

if (!String.IsNullOrEmpty(SearchByNTCequipmentProviders))

{

equipmentInStock = equipmentInStock.Where(s => s.NTCequipmentProviders.name\_NTCprovider.

Contains(SearchByNTCequipmentProviders));

}

switch (sortOrder)

{

case"name\_desc":

equipmentInStock = equipmentInStock.OrderByDescending(s => s.name\_NTCequipment);

break;

case"name\_type":

equipmentInStock = equipmentInStock.OrderBy(s => s.TypesNTCequipment.name\_type\_NTCequipment);

break;

case"name\_type\_desc":

equipmentInStock = equipmentInStock.OrderByDescending(s => s.TypesNTCequipment.name\_type\_NTCequipment);

break;

case"eq\_provider":

equipmentInStock = equipmentInStock.OrderBy(s => s.NTCequipmentProviders.name\_NTCprovider);

break;

case"eq\_provider\_desc":

equipmentInStock = equipmentInStock.OrderByDescending(s => s.NTCequipmentProviders.name\_NTCprovider);

break;

case"amount":

equipmentInStock = equipmentInStock.OrderBy(s => s.amount\_NTCequipment);

break;

case"amount\_desc":

equipmentInStock = equipmentInStock.OrderByDescending(s => s.amount\_NTCequipment);

break;

default:

equipmentInStock = equipmentInStock.OrderBy(s => s.name\_NTCequipment);

break;

}

int pageSize = 8;

int pageNumber = (page ?? 1);

return View(equipmentInStock.ToPagedList(pageNumber, pageSize));

}

publicasync Task<ActionResult> Details(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCequipmentInStock nTCequipmentInStock = await db.NTCequipmentInStock.FindAsync(id);

if (nTCequipmentInStock == null)

{

return HttpNotFound();

}

return View(nTCequipmentInStock);

}

public ActionResult Create()

{

ViewBag.id\_NTCprovider = new SelectList(db.NTCequipmentProviders, "id\_NTCprovider", "name\_NTCprovider");

ViewBag.id\_type\_NTCequipment = new SelectList(db.TypesNTCequipment, "id\_type\_NTCequipment", "name\_type\_NTCequipment");

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Create([Bind(Include = "id\_NTCequipment,name\_NTCequipment,id\_type\_NTCequipment," +

"id\_NTCprovider,amount\_NTCequipment")] NTCequipmentInStock nTCequipmentInStock)

{

if (ModelState.IsValid)

{

db.NTCequipmentInStock.Add(nTCequipmentInStock);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

ViewBag.id\_NTCprovider = new SelectList(db.NTCequipmentProviders, "id\_NTCprovider", "name\_NTCprovider",

nTCequipmentInStock.id\_NTCprovider);

ViewBag.id\_type\_NTCequipment = new SelectList(db.TypesNTCequipment, "id\_type\_NTCequipment", "name\_type\_NTCequipment",

nTCequipmentInStock.id\_type\_NTCequipment);

return View(nTCequipmentInStock);

}

publicasync Task<ActionResult> Edit(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCequipmentInStock nTCequipmentInStock = await db.NTCequipmentInStock.FindAsync(id);

if (nTCequipmentInStock == null)

{

return HttpNotFound();

}

ViewBag.id\_NTCprovider = new SelectList(db.NTCequipmentProviders, "id\_NTCprovider", "name\_NTCprovider",

nTCequipmentInStock.id\_NTCprovider);

ViewBag.id\_type\_NTCequipment = new SelectList(db.TypesNTCequipment, "id\_type\_NTCequipment", "name\_type\_NTCequipment",

nTCequipmentInStock.id\_type\_NTCequipment);

return View(nTCequipmentInStock);

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Edit([Bind(Include = "id\_NTCequipment,name\_NTCequipment,id\_type\_NTCequipment," +

"id\_NTCprovider,amount\_NTCequipment")] NTCequipmentInStock nTCequipmentInStock)

{

if (ModelState.IsValid)

{

db.Entry(nTCequipmentInStock).State = EntityState.Modified;

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

ViewBag.id\_NTCprovider = new SelectList(db.NTCequipmentProviders, "id\_NTCprovider", "name\_NTCprovider",

nTCequipmentInStock.id\_NTCprovider);

ViewBag.id\_type\_NTCequipment = new SelectList(db.TypesNTCequipment, "id\_type\_NTCequipment", "name\_type\_NTCequipment",

nTCequipmentInStock.id\_type\_NTCequipment);

return View(nTCequipmentInStock);

}

publicasync Task<ActionResult> Delete(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCequipmentInStock nTCequipmentInStock = await db.NTCequipmentInStock.FindAsync(id);

if (nTCequipmentInStock == null)

{

return HttpNotFound();

}

return View(nTCequipmentInStock);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> DeleteConfirmed(int id)

{

NTCequipmentInStock nTCequipmentInStock = await db.NTCequipmentInStock.FindAsync(id);

db.NTCequipmentInStock.Remove(nTCequipmentInStock);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

protectedoverridevoid Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Додаток Є

**Лістинг програмного коду «TypesNTCServicesController»**

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Web;

using System.Web.Mvc;

using NTCService;

using PagedList;

namespace NTCService.Controllers

{

[Authorize(Roles = "Admin")]

publicclassTypesNTCServicesController : Controller

{

private ModelDB db = new ModelDB();

public ActionResult Index(string sortOrder, string SearchByName, string currentFilter, int? page)

{

ViewBag.CurrentSort = sortOrder;

ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

if (SearchByName != null)

{

page = 1;

}

else

{

SearchByName = currentFilter;

}

ViewBag.CurrentFilter = SearchByName;

var typesServices = from s in db.TypesNTCServices select s;

if (!String.IsNullOrEmpty(SearchByName))

{

typesServices = typesServices.Where(s => s.name\_type\_NTCservice.Contains(SearchByName));

}

switch (sortOrder)

{

case"name\_desc":

typesServices = typesServices.OrderByDescending(s => s.name\_type\_NTCservice);

break;

default:

typesServices = typesServices.OrderBy(s => s.name\_type\_NTCservice);

break;

}

int pageSize = 8;

int pageNumber = (page ?? 1);

return View( typesServices.ToPagedList(pageNumber, pageSize));

}

publicasync Task<ActionResult> Details(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

TypesNTCServices typesNTCServices = await db.TypesNTCServices.FindAsync(id);

if (typesNTCServices == null)

{

return HttpNotFound();

}

return View(typesNTCServices);

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Create([Bind(Include = "id\_type\_NTCservice,name\_type\_NTCservice")]

TypesNTCServices typesNTCServices)

{

if (ModelState.IsValid)

{

db.TypesNTCServices.Add(typesNTCServices);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(typesNTCServices);

}

publicasync Task<ActionResult> Edit(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

TypesNTCServices typesNTCServices = await db.TypesNTCServices.FindAsync(id);

if (typesNTCServices == null)

{

return HttpNotFound();

}

return View(typesNTCServices);

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Edit([Bind(Include = "id\_type\_NTCservice,name\_type\_NTCservice")]

TypesNTCServices typesNTCServices)

{

if (ModelState.IsValid)

{

db.Entry(typesNTCServices).State = EntityState.Modified;

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(typesNTCServices);

}

publicasync Task<ActionResult> Delete(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

TypesNTCServices typesNTCServices = await db.TypesNTCServices.FindAsync(id);

if (typesNTCServices == null)

{

return HttpNotFound();

}

return View(typesNTCServices);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> DeleteConfirmed(int id)

{

TypesNTCServices typesNTCServices = await db.TypesNTCServices.FindAsync(id);

db.TypesNTCServices.Remove(typesNTCServices);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

protectedoverridevoid Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Додаток Ж

**Лістинг програмного коду «NTCServicesController.cs»**

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Web;

using System.Web.Mvc;

using NTCService;

using PagedList;

namespace NTCService.Controllers

{

[Authorize(Roles = "Admin")]

publicclassNTCServicesController : Controller

{

private ModelDB db = new ModelDB();

public ActionResult Index(string sortOrder, string SearchByName, string SearchByTypeNTCService, string currentFilter, int? page)

{

ViewBag.CurrentSort = sortOrder;

ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

ViewBag.PriceSortParm = sortOrder == "price" ? "price\_desc" : "price";

ViewBag.TypeServiceSortParm = sortOrder == "name\_type" ? "name\_type\_desc" : "name\_type";

ViewBag.EquipmentInStockrSortParm = sortOrder == "eq\_in\_stock" ? "eq\_in\_stock\_desc" : "eq\_in\_stock";

if (SearchByName != null)

{

page = 1;

}

else

{

SearchByName = currentFilter;

}

if (SearchByTypeNTCService != null)

{

page = 1;

}

else

{

SearchByTypeNTCService = currentFilter;

}

ViewBag.CurrentFilter = SearchByName;

ViewBag.CurrentFilter = SearchByTypeNTCService;

var services = db.NTCServices.Include(n => n.NTCequipmentInStock).Include(n => n.TypesNTCServices);

if (!String.IsNullOrEmpty(SearchByName))

{

services = services.Where(s => s.name\_NTCservice.Contains(SearchByName));

}

if (!String.IsNullOrEmpty(SearchByTypeNTCService))

{

services = services.Where(s => s.TypesNTCServices.name\_type\_NTCservice.Contains(SearchByTypeNTCService));

}

switch (sortOrder)

{

case"name\_desc":

services = services.OrderByDescending(s => s.name\_NTCservice);

break;

case"price":

services = services.OrderBy(s => s.price\_NTCservice);

break;

case"price\_desc":

services = services.OrderByDescending(s => s.price\_NTCservice);

break;

case"name\_type":

services = services.OrderBy(s => s.TypesNTCServices.name\_type\_NTCservice);

break;

case"name\_type\_desc":

services = services.OrderByDescending(s => s.TypesNTCServices.name\_type\_NTCservice);

break;

case"eq\_in\_stock":

services = services.OrderBy(s => s.NTCequipmentInStock.name\_NTCequipment);

break;

case"eq\_in\_stock\_desc":

services = services.OrderByDescending(s => s.NTCequipmentInStock.name\_NTCequipment);

break;

default:

services = services.OrderBy(s => s.name\_NTCservice);

break;

}

int pageSize = 8;

int pageNumber = (page ?? 1);

return View(services.ToPagedList(pageNumber, pageSize));

}

publicasync Task<ActionResult> Details(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCServices nTCServices = await db.NTCServices.FindAsync(id);

if (nTCServices == null)

{

return HttpNotFound();

}

return View(nTCServices);

}

public ActionResult Create()

{

ViewBag.id\_NTCequipment = new SelectList(db.NTCequipmentInStock, "id\_NTCequipment", "name\_NTCequipment");

ViewBag.id\_type\_NTCservice = new SelectList(db.TypesNTCServices, "id\_type\_NTCservice", "name\_type\_NTCservice");

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Create([Bind(Include = "id\_NTCservice,name\_NTCservice,price\_NTCservice,id\_NTCequipment," +

"id\_type\_NTCservice")] NTCServices nTCServices)

{

if (ModelState.IsValid)

{

db.NTCServices.Add(nTCServices);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

ViewBag.id\_NTCequipment = new SelectList(db.NTCequipmentInStock, "id\_NTCequipment", "name\_NTCequipment",

nTCServices.id\_NTCequipment);

ViewBag.id\_type\_NTCservice = new SelectList(db.TypesNTCServices, "id\_type\_NTCservice", "name\_type\_NTCservice",

nTCServices.id\_type\_NTCservice);

return View(nTCServices);

}

publicasync Task<ActionResult> Edit(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCServices nTCServices = await db.NTCServices.FindAsync(id);

if (nTCServices == null)

{

return HttpNotFound();

}

ViewBag.id\_NTCequipment = new SelectList(db.NTCequipmentInStock, "id\_NTCequipment", "name\_NTCequipment",

nTCServices.id\_NTCequipment);

ViewBag.id\_type\_NTCservice = new SelectList(db.TypesNTCServices, "id\_type\_NTCservice", "name\_type\_NTCservice",

nTCServices.id\_type\_NTCservice);

return View(nTCServices);

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Edit([Bind(Include = "id\_NTCservice,name\_NTCservice,price\_NTCservice,id\_NTCequipment," +

"id\_type\_NTCservice")] NTCServices nTCServices)

{

if (ModelState.IsValid)

{

db.Entry(nTCServices).State = EntityState.Modified;

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

ViewBag.id\_NTCequipment = new SelectList(db.NTCequipmentInStock, "id\_NTCequipment", "name\_NTCequipment",

nTCServices.id\_NTCequipment);

ViewBag.id\_type\_NTCservice = new SelectList(db.TypesNTCServices, "id\_type\_NTCservice", "name\_type\_NTCservice",

nTCServices.id\_type\_NTCservice);

return View(nTCServices);

}

publicasync Task<ActionResult> Delete(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCServices nTCServices = await db.NTCServices.FindAsync(id);

if (nTCServices == null)

{

return HttpNotFound();

}

return View(nTCServices);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> DeleteConfirmed(int id)

{

NTCServices nTCServices = await db.NTCServices.FindAsync(id);

db.NTCServices.Remove(nTCServices);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

protectedoverridevoid Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Додаток З

**Лістинг програмного коду «NTCClientsController.cs»**

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Web;

using System.Web.Mvc;

using NTCService;

using PagedList;

namespace NTCService.Controllers

{

[Authorize(Roles = "Admin")]

publicclassNTCClientsController : Controller

{

private ModelDB db = new ModelDB();

public ActionResult Index(string sortOrder, string SearchByName, string SearchByNumb, string SearchByMail, string SearchBySkype,

string currentFilter, int? page)

{

ViewBag.CurrentSort = sortOrder;

ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

ViewBag.PhoneNumbSortParm = sortOrder == "phone\_numb" ? "phone\_numb\_desc" : "phone\_numb";

ViewBag.EmailSortParm = sortOrder == "email" ? "email\_desc" : "email";

ViewBag.SkypeSortParm = sortOrder == "skype" ? "skype\_desc" : "skype";

if (SearchByName != null)

{

page = 1;

}

else

{

SearchByName = currentFilter;

}

if (SearchByNumb != null)

{

page = 1;

}

else

{

SearchByNumb = currentFilter;

}

if (SearchByMail != null)

{

page = 1;

}

else

{

SearchByMail = currentFilter;

}

if (SearchBySkype != null)

{

page = 1;

}

else

{

SearchBySkype = currentFilter;

}

ViewBag.CurrentFilter = SearchByName;

ViewBag.CurrentFilter = SearchByNumb;

ViewBag.CurrentFilter = SearchByMail;

ViewBag.CurrentFilter = SearchBySkype;

var clients = from s in db.NTCClients select s;

if (!String.IsNullOrEmpty(SearchByName))

{

clients = clients.Where(s => s.name\_NTCclient.Contains(SearchByName));

}

if (!String.IsNullOrEmpty(SearchByNumb))

{

clients = clients.Where(s => s.phone\_number\_NTCclient.Contains(SearchByNumb));

}

if (!String.IsNullOrEmpty(SearchByMail))

{

clients = clients.Where(s => s.email\_NTCclient.Contains(SearchByMail));

}

if (!String.IsNullOrEmpty(SearchBySkype))

{

clients = clients.Where(s => s.skype\_NTCclient.Contains(SearchBySkype));

}

switch (sortOrder)

{

case"name\_desc":

clients = clients.OrderByDescending(s => s.name\_NTCclient);

break;

case"phone\_numb":

clients = clients.OrderBy(s => s.phone\_number\_NTCclient);

break;

case"phone\_numb\_desc":

clients = clients.OrderByDescending(s => s.phone\_number\_NTCclient);

break;

case"email":

clients = clients.OrderBy(s => s.email\_NTCclient);

break;

case"email\_desc":

clients = clients.OrderByDescending(s => s.email\_NTCclient);

break;

case"skype":

clients = clients.OrderBy(s => s.skype\_NTCclient);

break;

case"skype\_desc":

clients = clients.OrderByDescending(s => s.skype\_NTCclient);

break;

default:

clients = clients.OrderBy(s => s.name\_NTCclient);

break;

}

int pageSize = 8;

int pageNumber = (page ?? 1);

return View(clients.ToPagedList(pageNumber, pageSize));

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Create([Bind(Include = "id\_NTCclient,name\_NTCclient,phone\_number\_NTCclient," +

"email\_NTCclient,skype\_NTCclient")] NTCClients nTCClients)

{

if (ModelState.IsValid)

{

db.NTCClients.Add(nTCClients);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(nTCClients);

}

publicasync Task<ActionResult> Edit(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCClients nTCClients = await db.NTCClients.FindAsync(id);

if (nTCClients == null)

{

return HttpNotFound();

}

return View(nTCClients);

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Edit([Bind(Include = "id\_NTCclient,name\_NTCclient,phone\_number\_NTCclient," +

"email\_NTCclient,skype\_NTCclient")] NTCClients nTCClients)

{

if (ModelState.IsValid)

{

db.Entry(nTCClients).State = EntityState.Modified;

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(nTCClients);

}

publicasync Task<ActionResult> Delete(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCClients nTCClients = await db.NTCClients.FindAsync(id);

if (nTCClients == null)

{

return HttpNotFound();

}

return View(nTCClients);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> DeleteConfirmed(int id)

{

NTCClients nTCClients = await db.NTCClients.FindAsync(id);

db.NTCClients.Remove(nTCClients);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

protectedoverridevoid Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Додаток И

**Лістинг програмного коду «NTCWorkersController.cs»**

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Web;

using System.Web.Mvc;

using NTCService;

using PagedList;

namespace NTCService.Controllers

{

[Authorize(Roles = "Admin")]

publicclassNTCWorkersController : Controller

{

private ModelDB db = new ModelDB();

public ActionResult Index(string sortOrder, string SearchByName, string SearchByNumb, string SearchByMail, string SearchBySkype,

string currentFilter, int? page)

{

ViewBag.CurrentSort = sortOrder;

ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

ViewBag.PhoneNumbSortParm = sortOrder == "phone\_numb" ? "phone\_numb\_desc" : "phone\_numb";

ViewBag.EmailSortParm = sortOrder == "email" ? "email\_desc" : "email";

ViewBag.SkypeSortParm = sortOrder == "skype" ? "skype\_desc" : "skype";

ViewBag.ExperienceSortParm = sortOrder == "experience" ? "experience\_desc" : "experience";

ViewBag.SalarySortParm = sortOrder == "salary" ? "salary\_desc" : "salary";

if (SearchByName != null)

{

page = 1;

}

else

{

SearchByName = currentFilter;

}

if (SearchByNumb != null)

{

page = 1;

}

else

{

SearchByNumb = currentFilter;

}

if (SearchByMail != null)

{

page = 1;

}

else

{

SearchByMail = currentFilter;

}

if (SearchBySkype != null)

{

page = 1;

}

else

{

SearchBySkype = currentFilter;

}

ViewBag.CurrentFilter = SearchByName;

ViewBag.CurrentFilter = SearchByNumb;

ViewBag.CurrentFilter = SearchByMail;

ViewBag.CurrentFilter = SearchBySkype;

var workers = from s in db.NTCWorkers select s;

if (!String.IsNullOrEmpty(SearchByName))

{

workers = workers.Where(s => s.name\_NTCworker.Contains(SearchByName));

}

if (!String.IsNullOrEmpty(SearchByNumb))

{

workers = workers.Where(s => s.phone\_number\_NTCworker.Contains(SearchByNumb));

}

if (!String.IsNullOrEmpty(SearchByMail))

{

workers = workers.Where(s => s.email\_NTCworker.Contains(SearchByMail));

}

if (!String.IsNullOrEmpty(SearchBySkype))

{

workers = workers.Where(s => s.skype\_NTCworker.Contains(SearchBySkype));

}

switch (sortOrder)

{

case"name\_desc":

workers = workers.OrderByDescending(s => s.name\_NTCworker);

break;

case"phone\_numb":

workers = workers.OrderBy(s => s.phone\_number\_NTCworker);

break;

case"phone\_numb\_desc":

workers = workers.OrderByDescending(s => s.phone\_number\_NTCworker);

break;

case"email":

workers = workers.OrderBy(s => s.email\_NTCworker);

break;

case"email\_desc":

workers = workers.OrderByDescending(s => s.email\_NTCworker);

break;

case"skype":

workers = workers.OrderBy(s => s.skype\_NTCworker);

break;

case"skype\_desc":

workers = workers.OrderByDescending(s => s.skype\_NTCworker);

break;

case"experience":

workers = workers.OrderBy(s => s.experience);

break;

case"experience\_desc":

workers = workers.OrderByDescending(s => s.experience);

break;

case"salary":

workers = workers.OrderBy(s => s.salary);

break;

case"salary\_desc":

workers = workers.OrderByDescending(s => s.salary);

break;

default:

workers = workers.OrderBy(s => s.name\_NTCworker);

break;

}

int pageSize = 8;

int pageNumber = (page ?? 1);

return View(workers.ToPagedList(pageNumber, pageSize));

}

publicasync Task<ActionResult> Details(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCWorkers nTCWorkers = await db.NTCWorkers.FindAsync(id);

if (nTCWorkers == null)

{

return HttpNotFound();

}

return View(nTCWorkers);

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Create([Bind(Include = "id\_NTCworker,name\_NTCworker,phone\_number\_NTCworker," +

"email\_NTCworker,skype\_NTCworker,experience,salary")] NTCWorkers nTCWorkers)

{

if (ModelState.IsValid)

{

db.NTCWorkers.Add(nTCWorkers);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(nTCWorkers);

}

publicasync Task<ActionResult> Edit(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCWorkers nTCWorkers = await db.NTCWorkers.FindAsync(id);

if (nTCWorkers == null)

{

return HttpNotFound();

}

return View(nTCWorkers);

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Edit([Bind(Include = "id\_NTCworker,name\_NTCworker,phone\_number\_NTCworker," +

"email\_NTCworker,skype\_NTCworker,experience,salary")] NTCWorkers nTCWorkers)

{

if (ModelState.IsValid)

{

db.Entry(nTCWorkers).State = EntityState.Modified;

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(nTCWorkers);

}

publicasync Task<ActionResult> Delete(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCWorkers nTCWorkers = await db.NTCWorkers.FindAsync(id);

if (nTCWorkers == null)

{

return HttpNotFound();

}

return View(nTCWorkers);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> DeleteConfirmed(int id)

{

NTCWorkers nTCWorkers = await db.NTCWorkers.FindAsync(id);

db.NTCWorkers.Remove(nTCWorkers);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

protectedoverridevoid Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Додаток І

**Лістинг програмного коду «NTCJournalController.cs»**

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Web;

using System.Web.Mvc;

using NTCService;

using PagedList;

namespace NTCService.Controllers

{

[Authorize(Roles = "Admin")]

publicclassNTCJournalController : Controller

{

private ModelDB db = new ModelDB();

public ActionResult Index(string sortOrder, string SearchByClient, string SearchByWorker, string SearchByService,

string currentFilter, int? page)

{

ViewBag.CurrentSort = sortOrder;

ViewBag.DateSortParm = String.IsNullOrEmpty(sortOrder) ? "date\_recording\_desc" : "";

ViewBag.ClientSortParm = sortOrder == "name\_client" ? "name\_client\_desc" : "name\_client";

ViewBag.WorkerSortParm = sortOrder == "name\_worker" ? "name\_worker\_desc" : "name\_worker";

ViewBag.TimeSortParm = sortOrder == "time\_recording" ? "time\_recording\_desc" : "time\_recording";

ViewBag.ServicestSortParm = sortOrder == "name\_service" ? "name\_service\_desc" : "name\_service";

ViewBag.AmountSortParm = sortOrder == "amount" ? "amount\_desc" : "amount";

ViewBag.PriceSortParm = sortOrder == "total\_price" ? "total\_price\_desc" : "total\_price";

ViewBag.WorkerPersentSortParm = sortOrder == "worker\_percentage" ? "worker\_percentage\_desc" : "worker\_percentage";

if (SearchByClient != null)

{

page = 1;

}

else

{

SearchByClient = currentFilter;

}

if (SearchByWorker != null)

{

page = 1;

}

else

{

SearchByWorker = currentFilter;

}

if (SearchByService != null)

{

page = 1;

}

else

{

SearchByService = currentFilter;

}

ViewBag.CurrentFilter = SearchByClient;

ViewBag.CurrentFilter = SearchByWorker;

ViewBag.CurrentFilter = SearchByService;

var journal = db.NTCJournal.Include(n => n.NTCClients).Include(n => n.NTCServices).Include(n => n.NTCWorkers);

if (!String.IsNullOrEmpty(SearchByClient))

{

journal = journal.Where(s => s.NTCClients.name\_NTCclient.Contains(SearchByClient));

}

if (!String.IsNullOrEmpty(SearchByWorker))

{

journal = journal.Where(s => s.NTCWorkers.name\_NTCworker.Contains(SearchByWorker));

}

if (!String.IsNullOrEmpty(SearchByService))

{

journal = journal.Where(s => s.NTCServices.name\_NTCservice.Contains(SearchByService));

}

switch (sortOrder)

{

case"date\_recording\_desc":

journal = journal.OrderByDescending(s => s.date\_NTCjournal);

break;

case"name\_client":

journal = journal.OrderBy(s => s.NTCClients.name\_NTCclient);

break;

case"name\_client\_desc":

journal = journal.OrderByDescending(s => s.NTCClients.name\_NTCclient);

break;

case"name\_worker":

journal = journal.OrderBy(s => s.NTCWorkers.name\_NTCworker);

break;

case"name\_worker\_desc":

journal = journal.OrderByDescending(s => s.NTCWorkers.name\_NTCworker);

break;

case"time\_recording":

journal = journal.OrderBy(s => s.time\_NTCjournal);

break;

case"time\_recording\_desc":

journal = journal.OrderByDescending(s => s.time\_NTCjournal);

break;

case"name\_service":

journal = journal.OrderBy(s => s.NTCServices.name\_NTCservice);

break;

case"name\_service\_desc":

journal = journal.OrderByDescending(s => s.NTCServices.name\_NTCservice);

break;

case"amount":

journal = journal.OrderBy(s => s.amount\_NTCequipment);

break;

case"amount\_desc":

journal = journal.OrderByDescending(s => s.amount\_NTCequipment);

break;

case"total\_price":

journal = journal.OrderBy(s => s.total\_price);

break;

case"total\_price\_desc":

journal = journal.OrderByDescending(s => s.total\_price);

break;

case"worker\_percentage":

journal = journal.OrderBy(s => s.NTCworker\_percentage);

break;

case"worker\_percentage\_desc":

journal = journal.OrderByDescending(s => s.NTCworker\_percentage);

break;

default:

journal = journal.OrderBy(s => s.date\_NTCjournal);

break;

}

int pageSize = 8;

int pageNumber = (page ?? 1);

return View(journal.ToPagedList(pageNumber, pageSize));

}

publicasync Task<ActionResult> Details(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCJournal nTCJournal = await db.NTCJournal.FindAsync(id);

if (nTCJournal == null)

{

return HttpNotFound();

}

return View(nTCJournal);

}

public ActionResult Create()

{

ViewBag.id\_NTCclient = new SelectList(db.NTCClients, "id\_NTCclient", "name\_NTCclient");

ViewBag.id\_NTCservice = new SelectList(db.NTCServices, "id\_NTCservice", "name\_NTCservice");

ViewBag.id\_NTCworker = new SelectList(db.NTCWorkers, "id\_NTCworker", "name\_NTCworker");

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Create([Bind(Include = "id\_NTCjournal,id\_NTCclient,id\_NTCworker,date\_NTCjournal," +

"time\_NTCjournal,id\_NTCservice,amount\_NTCequipment,total\_price,NTCworker\_percentage")] NTCJournal nTCJournal)

{

if (ModelState.IsValid)

{

db.NTCJournal.Add(nTCJournal);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

ViewBag.id\_NTCclient = new SelectList(db.NTCClients, "id\_NTCclient", "name\_NTCclient", nTCJournal.id\_NTCclient);

ViewBag.id\_NTCservice = new SelectList(db.NTCServices, "id\_NTCservice", "name\_NTCservice", nTCJournal.id\_NTCservice);

ViewBag.id\_NTCworker = new SelectList(db.NTCWorkers, "id\_NTCworker", "name\_NTCworker", nTCJournal.id\_NTCworker);

return View(nTCJournal);

}

publicasync Task<ActionResult> Edit(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCJournal nTCJournal = await db.NTCJournal.FindAsync(id);

if (nTCJournal == null)

{

return HttpNotFound();

}

ViewBag.id\_NTCclient = new SelectList(db.NTCClients, "id\_NTCclient", "name\_NTCclient", nTCJournal.id\_NTCclient);

ViewBag.id\_NTCservice = new SelectList(db.NTCServices, "id\_NTCservice", "name\_NTCservice", nTCJournal.id\_NTCservice);

ViewBag.id\_NTCworker = new SelectList(db.NTCWorkers, "id\_NTCworker", "name\_NTCworker", nTCJournal.id\_NTCworker);

return View(nTCJournal);

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Edit([Bind(Include = "id\_NTCjournal,id\_NTCclient,id\_NTCworker,date\_NTCjournal," +

"time\_NTCjournal,id\_NTCservice,amount\_NTCequipment,total\_price,NTCworker\_percentage")] NTCJournal nTCJournal)

{

if (ModelState.IsValid)

{

db.Entry(nTCJournal).State = EntityState.Modified;

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

ViewBag.id\_NTCclient = new SelectList(db.NTCClients, "id\_NTCclient", "name\_NTCclient", nTCJournal.id\_NTCclient);

ViewBag.id\_NTCservice = new SelectList(db.NTCServices, "id\_NTCservice", "name\_NTCservice", nTCJournal.id\_NTCservice);

ViewBag.id\_NTCworker = new SelectList(db.NTCWorkers, "id\_NTCworker", "name\_NTCworker", nTCJournal.id\_NTCworker);

return View(nTCJournal);

}

publicasync Task<ActionResult> Delete(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

NTCJournal nTCJournal = await db.NTCJournal.FindAsync(id);

if (nTCJournal == null)

{

return HttpNotFound();

}

return View(nTCJournal);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> DeleteConfirmed(int id)

{

NTCJournal nTCJournal = await db.NTCJournal.FindAsync(id);

db.NTCJournal.Remove(nTCJournal);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

protectedoverridevoid Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Додаток Ї

**Лістинг програмного коду «ApplicationController.cs»**

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Web;

using System.Web.Mvc;

using NTCService;

using PagedList;

namespace NTCService.Controllers

{

[Authorize(Roles = "Admin")]

publicclassApplicationController : Controller

{

private ModelDB db = new ModelDB();

public ActionResult Index( int? page)

{

int pageSize = 8;

int pageNumber = (page ?? 1);

return View(db.Application.ToPagedList(pageNumber, pageSize));

}

publicasync Task<ActionResult> Details(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Application application = await db.Application.FindAsync(id);

if (application == null)

{

return HttpNotFound();

}

return View(application);

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Create([Bind(Include = "id\_application,phone\_number\_NTCworker,email\_NTCworker")]

Application application)

{

if (ModelState.IsValid)

{

db.Application.Add(application);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(application);

}

publicasync Task<ActionResult> Edit(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Application application = await db.Application.FindAsync(id);

if (application == null)

{

return HttpNotFound();

}

return View(application);

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> Edit([Bind(Include = "id\_application,phone\_number\_NTCworker,email\_NTCworker")]

Application application)

{

if (ModelState.IsValid)

{

db.Entry(application).State = EntityState.Modified;

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(application);

}

publicasync Task<ActionResult> Delete(int? id)

{

if (id == null)

{

returnnew HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Application application = await db.Application.FindAsync(id);

if (application == null)

{

return HttpNotFound();

}

return View(application);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<ActionResult> DeleteConfirmed(int id)

{

Application application = await db.Application.FindAsync(id);

db.Application.Remove(application);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

protectedoverridevoid Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}}

Додаток Й

**Лістинг програмного коду «OptionsController.cs»**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Data.Entity;

using OfficeOpenXml;

namespace NTCService.Controllers

{

[Authorize(Roles = "Admin")]

publicclassOptionsController : Controller

{

private ModelDB db = new ModelDB();

public ActionResult Index()

{

return View();

}

public ActionResult BackupDatabase()

{

using (var db = new ModelDB())

{

var cmd = String.Format

("BACKUP DATABASE ntcservice TO DISK='D:\\юнітісрака\\MSSQL12.SQLEXPRESS\\MSSQL\\Backup\\" +

"ntcservice.bak' WITH FORMAT, MEDIANAME='Full Db backup', MEDIADESCRIPTION='Full Database Backup';");

db.Database.ExecuteSqlCommand(TransactionalBehavior.DoNotEnsureTransaction, cmd);

}

return View();

}

public ActionResult RestoreDatabase()

{

using (var db = new ModelDB())

{

var cmd = String.Format

("USE master restore DATABASE ntcservice from DISK='D:\\юнітісрака\\MSSQL12.SQLEXPRESS\\MSSQL\\Backup\\" +

"ntcservice.bak' WITH REPLACE;");

db.Database.ExecuteSqlCommand(TransactionalBehavior.DoNotEnsureTransaction, cmd);

}

return View();

}

public FileContentResult Export()

{

var fileDownloadName = String.Format("Журнал NTCService.xlsx");

conststring contentType = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";

ExcelPackage package = GenerateExcelFile(db.NTCJournal.ToList());

var fsr = new FileContentResult(package.GetAsByteArray(), contentType);

fsr.FileDownloadName = fileDownloadName;

return fsr;

}

privatestatic ExcelPackage GenerateExcelFile(IEnumerable<NTCJournal> datasource)

{

ExcelPackage pck = new ExcelPackage();

ExcelWorksheet ws = pck.Workbook.Worksheets.Add("Таблиця");

ws.Cells[1, 1].Value = "Клієнт ";

ws.Cells[1, 2].Value = "Інженер";

ws.Cells[1, 3].Value = "Дата";

ws.Cells[1, 4].Value = "Час";

ws.Cells[1, 5].Value = "Послуга";

ws.Cells[1, 6].Value = "К-стьобладнання";

ws.Cells[1, 7].Value = "Повнаціна";

ws.Cells[1, 8].Value = "Відсотокінженера";

for (int i = 0; i < datasource.Count(); i++)

{

ws.Cells[i + 2, 1].Value = datasource.ElementAt(i).NTCClients.name\_NTCclient;

ws.Cells[i + 2, 2].Value = datasource.ElementAt(i).NTCWorkers.name\_NTCworker;

ws.Cells[i + 2, 3].Value = datasource.ElementAt(i).date\_NTCjournal;

ws.Cells[i + 2, 4].Value = datasource.ElementAt(i).time\_NTCjournal;

ws.Cells[i + 2, 5].Value = datasource.ElementAt(i).NTCServices.name\_NTCservice;

ws.Cells[i + 2, 6].Value = datasource.ElementAt(i).amount\_NTCequipment;

ws.Cells[i + 2, 7].Value = datasource.ElementAt(i).total\_price;

ws.Cells[i + 2, 8].Value = datasource.ElementAt(i).NTCworker\_percentage;

}

return pck;

}

}

}

Додаток К

**Лістинг програмного коду «ReportingController.cs»**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace NTCService.Controllers

{

[Authorize(Roles = "Admin")]

publicclassReportingController : Controller

{

public ActionResult Index()

{

return View();

}

}

}

Додаток Л

**Технічне завдання**

1. **Загальне положення**
   1. **Найменування програмного засобу**

Повне найменування програмної системи: "База даних длясервісного центру з ремонту та обслуговування оргтехніки" (надалі "програма"). Коротка назва програмної системи - "Сервісний центр"

* 1. **Призначення розробки та область застосування**

. База даних сервісного центру призначена для її працівників і забезпечує доступ до інформації про занесених до бази замовлень, клієнтів, типів послуг, працівників, обладнання. В програмі передбачено використання механізму відбору по параметрам даних.

Система може бути впроваджена у роботу сервісного центрузнеобмеженою кількістю співробітників та клієнтів.

**1.3.Мета**

Програмна система ""База даних длясервісного центру з ремонту та обслуговування оргтехніки" дозволить підвищити ефективність, повноту та швидкість роботи операторів сервісного центру щодо пошуку і відбору даних та забезпечить керівництвосвоєчасною та повною інформацією про актуальний стан сервісного центру.

**1.4. Найменування розробника та замовника.**

Розробник даного продукту - студентка групи ПІ-54 РаздобудоваВалерія(надалі "розробник").

Замовник програмного продукту – кафедра інженерії програмного забезпечення Житомирського державного технологічного університету в межах виконання курсової з дисципліни «Бази даних» (надалі замовник).

1. **Підстава для розробки** 
   1. **Документ на підставі якого ведеться розробка**

Робота ведеться на підставі навчального плану за напрямом 121 «Інженерія програмного забезпечення» наказу про закріплення тем курсових робіт за студентами.

1. **Вимоги до програми** 
   1. **Вимоги до функціональних характеристик.**

**3.1.1. Загальні вимоги**

Програмна система має забезпечувати:

* можливість дистанційної роботи з робочих станцій локальної та глобальної мережі підприємства;
* інтерфейс користувача, що не залежить від операційної системи;
* постійний доступ користувачів до БД;
* оптимальне збереження даних (за обсягом та структурою);
* аутентифікацію користувачів та захист інформації від несанкціонованого доступу;
* можливість модернізації системи через зміну функціональних потреб користувача або модернізацію обладнання;

**3.1.1.Складвиконуванихфункцій**

Розробити базу данихсервісногоцентру

, що підтримує виконання наступних транзакцій:

1. Реєстрацію користувачів;
2. Реєстрацію даних клієнтів у відповідності до типової форми;
3. Реєстрацію працівників у відповідності до типової форми;
4. Реєстрацію записів насервісні послуги;
5. Оформлення заявки на сервісні послуги;
6. Ведення обліку обладнання та його постачальників;
7. Відображення звітності по сервісному центру;
8. Пошук та відбір даних по мультикритеріями (ПІБ, тип послуг, дата та час запису, тощо);
9. Резервне копіювання та відновлення бази даних, еспорт даних у Excel;

**3.1.2. Організація вхідних і вихідних даних**

Вхідними данимиєданіпро послуги, працівників, клієнтів, записи, обладнаннята його постачальників.

Організація вхідних і вихідних даних повинна відповідати інформаційній структурі виконуваних з операцій, вхідним та вихідним паперовим документами.

Введення оперативних даних повинно виконуватися з використанням діалогових екранних форм, побудованих на основі візуальних компонентів. Введення даних виконується на основі затверджених форм документів.

**3.1.3. Часові характеристики і розмір пам'яті, необхідної для роботи програми.**

Час реакції програми на дії користувача (маніпуляції з пристроями введення даних) не повинен перевищувати 0,25 с.

Час виконання команд меню не більше 1 с.

Відображення масивів даних за запитами не більше 3 хвилин.

Доступність БД – 90% цілодобово.

Операції з’єднання з БД не більше 1 хвилини.

Обсяг оперативної пам'яті, необхідний для роботи програми не менше 1Гб.

Дисковій простір, необхідний для збереження програми і файлів даних не більше 300 Мбат для робочої станції та 20 ГБайт..

Інсталяційний пакет програми, що містить у складі БД не повинні перевищувати 100 Мбайт.

**3.2. Вимоги до надійності.**

**3.2.1. Вимоги до надійного функціонування**

Програма повинна нормально функціонувати при безперебійній роботі ПК. Доступність БД 90% при одночасному доступі 30 користувачів.

При апаратних збоях, відновлення нормальної роботи програми повинне виконуватися після:

а) апаратні збої сервера - перезавантаження ОС сервера, запуск сервера БД (запуску

резервного сервера, використання технологій RAID для збереження даних);

б)апаратні збої робочої станції – перезавантаження ОС ПК, запуск виконуваного файлу

програми.

в) БД повинна повертатись в найближчий несуперечний стан – передбачити точки

відновлення.

При збоях програмного забезпечення:

а) система повинна забезпечувати можливість відновлення даних та фіксацію і «відкат»

транзакцій.

б) в системі має бути реалізована коректна обробка виняткових ситуацій.

**3.2.2.Контроль вхідної і вихідної інформації**

Для контролю коректності вхідної інформації та захисту від помилок оператора:

* Перевірка відповідності даних доменам інформаційних атрибутів;
* Використанням механізму авто заповнення та вибору за переліком для зв’язаних даних;
* Захист від помилок оператора (залипань, випадкових символів тощо).

Визначені некоректні дії повинні супроводжуватись повідомленнями про помилку і блокуванням операцій оновлення даних. В системі має бути передбачений захист від загального блокування.

Для вихідної інформації передбачити:

* відсутність блокування даних через багато користувальницький доступ;
* постійне оновлення даних у відображених на екрані звітних формах.
* повідомлення про використання режиму монопольного доступу супервайзерами БД.

**3.2.3. Час відновлення після відмови.**

Час відновлення після відмови, не пов'язаною з роботою програми, повинен складатися із:

* часу перезапуску операційної системи; часу запуску сервера БД (підключення до сервера )
* запуску виконуваного файлу, часу повторного введення або зчитування даних.

**3.3. Умови експлуатації і збереження**

Програма використовується у багато користувальницькому середовищі. Регламенті операції:

* створення резервних копій даних;
* збереження резервної копії журналу транзакцій;
* архівування даних;
* обслуговуючі функції (прибирання “сміття”, дефрагментація, тощо);
* оновлення системи ідентифікації та аутентифікації користувачів.

Збереження архівних копій - на визначених носіях. Збереження резервних копій – на визначеному диску без перезапису. Знищення копій за регламентом.

**3.4. Вимоги до інформаційної і програмної сумісності.**

* + 1. **Вимоги до інформаційних структур на вході і виході**

Формат відображення даних має дозволяти надсилання потрібних даних на друк.

* + 1. **Вимоги до методів рішення і мов програмування**

Вибір методів рішення здійснюється розробникам без узгодження з замовником. СУБД обирається у відповідності до характеристик визначених в п.3. З замовником погоджується вибір варіанту за вартісною ознакою.

**3.4.3. Вимоги до системи програмних засобів.**

Вимоги до програмного забезпечення сервера:

ОС – серверна версія ОС Windows не нижче Windows 2008 Server, IIS (або інший сервер),

СУБД визначається встановлюється та налаштовується розробником (ліцензування СУБД виконується замовником).

Вимоги до програмного забезпечення робочої станції: ОС - родина Windows не нижче Windows 2008.

Драйвери периферійних пристроїв - введення/виводу визначаються та встановлюються при встановленні ОС в залежності від конфігурації робочої станції.

Робоча станція є клієнтом мереж Microsoft.

**3.5. Вимоги до складу і параметрів технічних засобів.**

Вимоги до складу технічних засобів:

1. Сервер:

* сервер у базовій конфігурації із підтримкою RAID;
* з’ємний запам’ятовуючий пристрій для архівування даних;
* принтер для друку;
* засоби для під’єднання до локальної мережі.

2. Робоча станція:

* ПК на базі процесорів Intel, AMD у стандартній комплектації;
* периферійні пристрої друку;
* засоби для під’єднання до локальної мережі.

**4. Вимоги до програмної документації**

Програмна документація повинна включати наступні відомості:

1. "Інструкція по інсталяції (встановленню) програми", складається з опису інсталяційного пакету, переліку етапів інсталяції та їх послідовності, опису встановлених програмних компонентів та режимів їх роботи після інсталяції. Під час оформлення пояснювальної записки до курсової роботи дані відомості містяться у 4 розділі.
2. Керівництво користувача" складається з опису послідовності завантаження програми, основних режимів роботи, опису основних екранних форм, переліку виняткових ситуацій та реакції користувача на них, порядку виконання завдань в системі. Під час оформлення пояснювальної записки до курсової роботи дані відомості містяться в 3 розділі курсової роботи.
3. "Керівництво адміністратора баз даних", складається з опису складу таблиць бази даних та правил доступів до них, опису та послідовності робіт по обслуговуванню бази даних (архівування, резервне копіювання, з вказівкою періодичності виконання та засобів, що для цього використовуються), порядку налаштування серверу та клієнтських додатків. Під час оформлення пояснювальної записки до курсової роботи дані відомості містяться в 2,3 та 4 розділах курсової роботи згідно плану.