**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ**

**Інститут (факультет )**_Автоматизації і комп'ютерних систем_____
**Кафедра** ___Інформаційних систем_____

| **«До захисту в ЕК»** | | **«До захисту допущено»** | |
|---|---|---|---|
| Директор інституту(декан факультету) | | Завідувач кафедри | |
| _____ | __Андрій Форсюк__ | _____ | __Сергій Чумаченко |
| (підпис) | (ім'я та прізвище) | (підпис) | (ім'я та прізвище) |
| «___» _____ 2022р. | | «___» _____ 2022р. | |

# КВАЛІФІКАЦІЙНА РОБОТА
# НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

зі спеціальності__122 «Комп'ютерні науки»_____
(код та назва спеціальності)
освітньо-професійної програми_Інформаційні управляючі системи та технології
_____
на тему:_Дослідження та розроблення інформаційно-аналітичної системи підтримки управління підприємством лісового господарства (на прикладі ТОВ «ПОЛІССЯЛІССЕРВІС)

Виконав: здобувач_2_ курсу, групи ІС-2-4М

_____Бережний Микола Вікторович_____
(прізвище, ім'я, по батькові повністю) (підпис)

Керівник ___Литвинов Валерій Андроникович_____
(прізвище , ім'я та по батькові повністю) (підпис)

Консультанти_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

Рецензент _Лідія Власенко____ _____
(ім'я та прізвище) (підпис)

Я як здобувач(ка) Національного університету харчових технологій розумію і підтримую політику університету з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволеної допомоги під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач_____
(підпис)

Київ - 2022р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) __Автоматизації і комп'ютерних систем_____
Кафедра __Інформаційних систем_____
Освітній ступінь__магістр _____
**Спеціальність**__122 «Комп'ютерні науки»_____
**(код і назва)**
**Освітньо-професійна програма** _____Інформаційні управляючі системи та технології_____
**(назва)**

**ЗАТВЕРДЖУЮ**
**Завідувач**
**кафедри** Інформаційних систем

_____
**"____" _____2021року**

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

**_____Бережний Микола Вікторович__**
(прізвище, ім'я, по батькові)

1. Тема роботи: Дослідження та розроблення інформаційно-аналітичної системи підтримки управління підприємством лісового господарства (на прикладі ТОВ «ПОЛІССЯЛІССЕРВІС»)_____
керівник роботи: професор Литвинов Валерій Андроникович_____,
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від "11"листопада 2021року № 884-кс
2. Строк подання здобувачем роботи_____
3. Вихідні дані до роботи ___ В даній магістерській роботі було досліджено організаційну структуру, та рівень автоматизації лісничого відділу ТОВ «ПОЛІССЯЛІССЕРВІС» також проведений аналіз існуючих систем-аналогів для підтримки . Використовуючи отриману інформацію, мною було прийнято рішення про створення нової інформаційної системи, що буде задовольняти всі вимоги даного відділу._____
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)____
Системний аналіз підприємства та системи збуту, постановка задачі на проектування_____Розробка логічної моделі даних _____
Постановка задачі _____
Розроблення інтерфейсу користувача в середовищі MS Visual Studio_____ Аналіз системи управління лісового господарства_____ Формування висновків_____

5. Перелік графічного матеріалу

_ Організаційна структура підприємства, логічна модель даних інтерфейсу користувача в середовищі MS Visual Studio, інструкція системного адміністратора та користувача_____

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1 | Литвинов В.А. доктор технічних наук, професор | | |
| 2 | Литвинов В.А. доктор технічних наук, професор | | |
| 3 | Литвинов В.А. доктор технічних наук, професор | | |

7. Дата видачі завдання____2021 р_____

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів виконання кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Системний аналіз підприємства | | |
| 2 | Постановка задачі на проектування | | |
| 3 | Розробка логічної моделі бази даних | | |
| 4 | Розроблення інтерфейсу користувача в середовищі MS Visual Studio | | |
| 5 | Розробка інтерфейсу користувача | | |
| 6 | Оформлення пояснювальної записки | | |
| 7 | Оформлення презентації | | |

**Здобувач** _____ ___Бережний М.В._____
(підпис) (прізвище та ініціали)

**Керівник роботи** _____ __Литвинов В.А._____
( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

Магістерська робота «Дослідження та розроблення інформаційно-аналітичної системи підтримки управління підприємством лісового господарства (на прикладі ТОВ «ПОЛІССЯЛІССЕРВІС»)» складається з 126 сторінок, 9 таблиць, 27 рисунків, 1 додаток, 28 літературних джерел та різних інтернет ресурсів, які були використані в цій роботі.

В даній магістерській роботі було досліджено організаційну структуру, та рівень автоматизації лісничого відділу ТОВ «ПОЛІССЯЛІССЕРВІС» також проведений аналіз існуючих систем-аналогів для підтримки . Використовуючи отриману інформацю, мною було прийнято рішення про створення нової інформаційної системи, що буде задовольняти всі вимоги даного відділу. Для реалізації всіх функцій, описаних в функціональній моделі, було використано програмне забезпечення, описані вимоги до всіх технічних засобів та основного інтерфейсу, наведені функції що реалізуються системою, розроблений основний програмний продукт та інструкція для користувача.

**КЛЮЧОВІ СЛОВА**: ІНФОРМАЦІЙНА СИСТЕМА, ЛІСОВІ УГІДДЯ, ЛІСНИЧИЙ ВІДДІЛ, БАЗА ДАНИХ, ЛОГІЧНА МОДЕЛЬ, ФІЗИЧНА МОДЕЛЬ, C#

## ANNOTATION

The bachelor's thesis «Research and development of information and analytical system to support the management of forestry enterprises (for example, LLC "POLISSYALISSERVICE")» consists of 126 pages, 8 tables, 27 figures, 1 appendix, 28 literature sources and various Internet resources that were used in this work.

In this bachelor's thesis, the organizational structure and the level of automation of the forestry department of POLISSYALISSERVICE LLC were studied, as well as the analysis of existing systems-analogues for support. Using the information received, I decided to create a new information system that will meet all the requirements of this department. To implement all the functions described in the functional model, the software was used, the requirements for all hardware and the main interface are described, the functions implemented by the system are given, the main software product and user manual are developed.

**KEY WORDS**: INFORMATION SYSTEM, SALES, FOREST LAND, DATABASE, LOGICAL MODEL, PHYSICAL MODEL, C #.

Зміст

# *ВСТУП*

**Актуальність теми.** Актуальність цієї роботи пов’язані з недостатнім в теперішній час використанням сучасних інформаційних технологій для управління підприємствами лісового господарства. Робота спрямована на автоматизацію процесів введення, виведення, зберігання, обробки, реєстрації і аналізу необхідної для управління інформації.

**Зв’язок роботи з науковими програмами, планами, темами.** Наукова робота виконувалася згідно з планом та програмою наукових досліджень на кафедрі інформаційних систем Національного університету харчових технологій (НУХТ).

**Об’єкт дослідження.** Компанія ТОВ «ПОЛІССЯЛІССЕРВІС», яка займається лісовим господарством.

**Предмет дослідження.** Методи та засоби підтримки управління лісовим господарством компанії ТОВ «ПОЛІССЯЛІССЕРВІС».

**Мета та задачі дослідження**

Метою роботи є підвищення ефективності управління лісовим господарством шляхом створення і застосування інформаційно-аналітичної системи для обліку і аналізу відповідних даних:

- Аналіз існуючих систем для автоматизації ведення лісового господарства;
- Вивчення особливостей діяльності у сфері лісового господарства;
- Розробка архітектури інформаційно-аналітичної системи підтримки управління лісовим господарством;
- Вибір технологій та реалізація програмного забезпечення, що дозволить автоматизувати бізнес процеси компанії.

**Методи дослідження.**

В даній роботі були використані такі методи дослідження:

- Аналіз аналогічних систем, для виявлення всіх проблем, що доведеться вирішити в даній роботі;
- Визначення аналогій програмних продуктів, що можуть бути використані для автоматизації процесів управління лісним господарством;
- Дослідити алгоритми процесів роботи підприємства, що буде необхідно для розробки програмного забезпечення;
- Аналіз проблем та вимог, які будуть стояти перед розробкою програмного забезпечення.

**Наукова новизна** полягає в наданні властивостей типовості розробки, що створює можливості її використання не тільки для компанії ТОВ «ПОЛІССЯЛІССЕРВІС», але і для інших аналогічних підприємств лісового господарства.

**Практичне значення отриманих результатів.**

Практичного використання даної системи полягає у автоматизації бізнес процесів компанії ТОВ «ПОЛІССЯЛІССЕРВІС», що зменшить використання людських ресурсів та зекономить час.

**Структура та обсяг магістерської роботи.**

Магістерська робота «Розроблення інформаційно-облікової системи лісових угідь ТОВ «ПОЛІССЯЛІССЕРВІС»» складається з 126 сторінок, 8 таблиць, 27 рисунків, 1 додаток, 28 літературних джерел та різних інтернет ресурсів, які були використані в цій роботі.

# 1. Аналіз ТОВ «ПОЛІССЯЛІССЕРВІС» та аналіз існуючих систем автоматизації ведення лісового господарства

## 1.1 Організаційна структура ТОВ «ПОЛІССЯЛІССЕРВІС»

Компанія ТОВ «ПОЛІССЯЛІССЕРВІС» займається лісовим господарством від висадки деревини у своїх лісових угіддях до її обробки та продажу готового матеріалу.

Організаційна структура ТОВ «ПОЛІССЯЛІССЕРВІС»


Рис. 1.1. - Організаційна структура компанії ТОВ «ПОЛІССЯЛІССЕРВІС»

1. Генеральний директор — найвища адміністративна посада, найвищий директор. Ця особа є верховний керівником компанії.
2. Лісничий відділ – що включає в себе догляд за лісовими угіддями, а також проведенням робіт на їх території (висадка, прорубка, прорідження, вирубка тощо).
3. Виробничий цех – обробка деревини до кінцевих (на продаж) матеріалів (дошки, оброблені колоди, стовбці, тирса, тощо), та зберігання деревини або пиломатеріалів на виробничих складах.
4. Відділ продажу – продаж необробленої та обробленої деревини.
5. Відділ логістики - упорядкуванні робочих процесів з метою зменшення витрат, підвищення обсягу оборотного капіталу. В результаті, досягається підвищення прибутковості.

## 1.2 Особливості діяльності у сфері лісового господарства

Ліс як особливий вид біогеоценозу в економічному відношенні представляє сукупність ресурсів (землі, деревного запасу, ресурсів побічного користування, захисних властивостей та ін.), кожен з них яких є джерелом споживчих цін.

Форми лісокористування залежить від наявності лісових ресурсів і від потреб людини.

Важливою властивістю лісових ресурсів, що визначає характер організації лісокористування, є довгостроковий характер їх перетворення, відтворення та накопичення. Тому диференціація лісокористування за потребами у лісових ресурсах має бути стабільною та стійкою.

При цьому споживання деревини має вестися так, щоб не викликати зменшення інших ресурсів.

Відповідно до значення лісів у народному господарстві, їх місцезнаходженням та виконуваними функціями лісу поділяють на три категорії:

1. Захисні ліси – до них відносять ліси, які розташовані на природних територіях, , що особливо охороняються;

2. Експлуатаційні ліси – це ліси, що мають переважно експлуатаційне значення;

3. Резервні ліси, у яких протягом двадцяти років не планується здійснювати заготівлю деревини. Використання резервних лісів допускається після їх віднесення до експлуатаційних чи захисних лісів.

Класифікація лісів на категорії забезпечує раціональне використання та їх захист.

Основні завдання, які мають бути вирішені під час переходу до сталого лісокористування:

1. Оцінка можливості застосування принципів ефективного управління лісами на сучасних лісових підприємствах, виявлення характерних ознак та розробка нової структури підприємств, що відповідають умовам та вимогам раціонального використання лісових ресурсів;

2. Оцінка відповідності сучасних технічних засобів, економічним та екологічним умовам та вимогам, що існують у системі стійкого лісового господарства.

Раціональне лісове господарство означає утримання та використання лісів, за умови зберігання їх видової різноманітності, продуктивності та життєздатності.

Еколого-економічні аспекти лісокористування ґрунтуються на взаємодії природи та суспільства та повинні забезпечувати:

- ✓ раціональне використання деревних ресурсів, що не призводить ні до скорочення площі лісів, до їх якісного погіршення;
- ✓ збереження основних функцій лісів, таких як захист водних джерел, запобігання ерозії ґрунтів, забезпечення балансу кисню та вуглекислого газу в атмосфері, що стабілізує вплив на клімат тощо;
- ✓ забезпечення потреб населення в основних благах та функціях лісу - місцях для туризму та відпочинку, грибах та ягодах, чистій воді та свіжому повітрі.

Економічні аспекти державного регулювання включають:

- облік лісового фонду та лісових ресурсів;
- їх економічну оцінку;
- види користування, платежі та формування доходів;
- охорону та захист.

Саме ліс формує інші рекреаційні ресурси: клімат, гідрографічний режим, ґрунт, фауну. Отже, вирубування лісу знижує рекреаційний потенціал території та призводить до економічних витрат.

Залежно від місця розташування, від значення для довкілля усі ліси поділені на три групи.

Ліси – зелений пояс навколо міст, уздовж залізничних та шосейних доріг; водоохоронні ліси, що розкинулися схилами гір, берегами річок і озер; заповідники, лісопарки та пам'ятники природи – всі ці ліси відносяться до першої групи. Тут дозволені лише рубки догляду та санітарні рубки.

Різке обмеження вирубки у лісах першої групи грає позитивну і негативну роль. Безперечно, це призвело до збільшення запасів деревини. Але водночас погіршилося санітарний стан лісів: старі перестійні дерева ускладнюють зростання молодняку, у лісах накопичуються сухі дерева, виникають осередки шкідників. Тому вибіркова заготівля стиглих дерев дозволено в деяких лісах першої групи.

До другої групи віднесено ліси з обмеженим запасом деревини. Розташовані вони, як правило, неподалік великих промислових центрів і пов'язані з ними мережею хороших доріг.

Тут вигідно заготовляти деревину, водночас це ліси виснажені рубками минулих століть. Склад цих лісів значно змінено: сосняки та ялинники у багатьох місцях змінилися березняками та осинниками.

Обсяг рубок тут обмежений, має виконуватися правило: вирубав – посади. Найважливіше завдання в лісах другої групи – поступова заміна листяних порід хвойними, точніше, їх правильне господарської точки зору співвідношення.

Третя група лісів – найбільша. Заготівля деревини проводиться тут великими ділянками вздовж лісовозних доріг, що знову прокладаються. Ліси третьої групи поділяються приблизно на дві рівні частини. Перша –

заготівельна, друга – резервна (її ліси ще не втягнуті в промислову експлуатацію, тут немає доріг, селищ та промислових підприємств).

Найціннішими для народного господарства є хвойні породи: сосна, ялина, ялиця, кедр. З них найбільше значення в промисловості мають сосна і ялина з яких виготовляється Великий асортимент виробів.

Ліс, що відноситься за віком до перших двох класів, вважається молодим; до третього, а північних районах до третього та четвертого – середньовікових; наступний клас – ліс процвітаючий, потім слідують два класи стиглого лісу, і останній клас, у якому дерева практично припинили зростати, - до перестійних лісів. Ці дерева схильні до різних захворювань і нападу шкідників.

Поділ лісів за віком має велике значення для фахівців лісової справи, оскільки саме на основі цієї класифікації вирішується питання про розмір щорічної рубки лісу.

Призначення рубок догляду за лісом – прибирання дерев, що відмирають, і тих, що заважають зростанню твердих порід дерев. Приблизно з віку – сто п'ятдесят років, у хвойних лісах запаси деревини починають зменшуватися. Причин цього багато: знижується річний приріст деревини, частина дерев щороку засихає, на старі, ослаблені дерева нападають шкідники та хвороби.

### 1.3 Необхідність автоматизації бізнес-процесів

На сьогоднішній день на ринку представлено безліч систем для автоматизації ведення сільського господарства.

Завдяки автоматизації відбувається:

• підвищення продуктивність праці;

• зведення до мінімуму негативного впливу людського фактору на найважливіші бізнес-процеси;

• безпечне зберігання інформації;

• підвищення якості обслуговування клієнтів;

• швидке отримання звітів;

• швидка підготовка документів;

• підвищення ефективності виробництва.

Автоматизація необхідна в тих ділянках діяльності, де її поява збільшить швидкість виконання завдань і, відповідно, підвищить ефективність праці лісного господарства. Наочно це можна показати, наприклад, на роботі адміністратора компанії: адміністратор без установленого програмного забезпечення багато працюватиме і йому знадобиться велика кількість часу, тому що кожного разу при запиті будь-якого звіту доведеться діставати всі папери, проводити розрахунки самостійно на калькуляторі тощо. Але якщо впровадити програму для автоматизації ведення лісового господарства, то будь-який звіт можна буде сформувати за допомогою натискання однієї кнопки.

Автоматизація діяльності компанії є проектною роботою, розбитою на фази, метою якої зрештою є підвищення ефективності роботи працівників, підвищення якості обслуговування клієнтів, збільшення точності даних, швидкості виконання операцій тощо.

Автоматизація діяльності завжди є унікальним рішенням та робота над цим проектом завжди індивідуальна. Але робота з проектами побудована на однакових засадах.

Для того, щоб спростити процес автоматизації, проект поділяють на кілька етапів:

• проектне дослідження;

• розробка і впровадження;

• супровід.

На першому етапі вивчається предметна область компанії. Ставиться ціль, яка допоможе вирішити поставлені завдання. Також збирається інформація про те, як працює компанія. Після перерахованого вище

визначається вартість та терміни, необхідні для реалізації проекту.

На другому етапі робота поділяється на фази:

• написання технічного завдання;

• розробка;

• тестування;

• дослідна експлуатація;

• навчання користувачів;

• введення в експлуатацію.

На третьому етапі здійснюється супровід автоматизованої системи:

• виправлення помилок;

• доопрацювання для покращення роботи програми, консультація.

### 1.4 Аналіз календарного планування

Важливою частиною проведення робіт, як в лісовому господарстві так і в інших структурних підприємствах є - календарне планування робіт.

Календарне планування — це процес складання й коригування розкладу, в якому роботи, що виконуються різними організаціями, взаємопов'язуються між собою в часі і з можливостями їх забезпечення різними видами матеріально-технічних та трудових ресурсів.

В лісовому господарстві календарне планування, це один з найважливіших аспектів, адже підготовка деревени в нових лісових угддях це довготривалий процес, а дбайливий догляд, своечасно проведені роботи та правильно підібраний час для збору сировини, не лише збільшує кільнісні показники, а й сильно впливає на якість добутої деревини.

Лісничий відділ ТОВ «ПОЛІССЯЛІССЕРВІС» також використовув, та регулярно дотримувався календарних планів виконання робіт. Проведення робіт в даному підприємстві, як правило планувалася не на певну дату, а потрібно було виконати норму за квартал вчасно. Облік в кварталах зумовлювався кількістю проведених робіт, та великим обсягом роботи, на

який можуть впливати навіть такі незнічні фактори як погодні умови. Регулярний облік проводився лише в кількості зачищеної території ласового господарства та добутого з даного участка деревини в гектарах зачищеної території та кількості тон добутої сировини для продажу або для передачі та перероблення сировини інншими відділати.

Обсяги обробленої територі за день могли сягати до декількох соток землі за одну робочу добу, та декількаами тонами здобутої деривини. Такі результати зумовлені модернізацією лісового господарства та активним використанням новітного обладнання для догляду за лісовим господарством та вирубки лісів. Всі дані регулярно вносилися до таблиці загального обліку, що давало змогу контролювати кількість здобутої сировини, в алежності від потреб клієнтів, та планувати підготовку оголеної території під нові засадження.

| Дата | Кількість обробленої території (Га) | Кількість здобутої сировини (т) |
|---|---|---|
| 24.10.2021 | 0,76 | 80 |
| 27.10.2021 | 0,68 | 87 |
| 01.11.2021 | 0,86 | 101 |
| 02.11.2021 | 0,56 | 76 |
| 14.11.2021 | 0,88 | 118 |
| | 3,74 | 462 |

Табл 1.1 Таблица розрахунків здобутої сировини на площу обробленого угіддя

Згідно таблиці 1.1 ми можемо прободити обліки території та добутої з неї сировини, а також розрахувати насттупні дані.

1) Середній показник здобутої сировини на 1 Га лісового угіддя. (Zcep)

Якщо позначити площу лісового господарства як Pl, а сировину як S, то згідно проведених робіт отримаємо формули:

Pl1+Pl2+...+Pln=Plзаг

S1+S2+…+Sn=Sзаг

Sзаг/Plзаг=Zсер

2) Кількіть обробленого участку за місяць або за певний період

Якщо позначити дату виконаних робіт як D, а площу в Pl, то згідно проведених робіт отримаємо формули:

Pl1+Pl2+…+Pln=Plп (встановивши певний період від D1 до Dn в додатку) ми отримуємо площу на якій проводилися роботи за вказаний період часу (Plп)

3) Кількіть здобутої сировини за місяць або за певний період

Якщо позначити дату виконаних робіт як D, а сировину як S, то згідно проведених робіт отримаємо формули:

S1+S2+…+Sn=Sп (встановивши певний період від D1 до Dn в додатку) ми отримуємо площу на якій проводилися роботи за вказаний період часу (Sп)

4) Кількість проведених робіт за місяць або за певний період

Якщо позначити дату виконаних робіт як D=, то згідно проведених робіт отримаємо формули:

D1+D2+…+Dn=Dп (встановивши певний період від D1 до Dn в додатку) ми отримуємо кількість проведених робіт за вказаний період часу (Sп)

### 1.5 Порівняльний аналіз існуючих розробок

Проаналізуємо уже існуючі інформаційні системи та визначимо здатність впровадження таких систем до лісничого відділу ТОВ «ПОЛІССЯЛІССЕРВІС».

*1С:Підприємство "управління лісовим господарством"*

Даний програмний комплекс являє собою потужний інструмент для використання сучасних методів управління підприємством лісової галузі, охоплюючи таку специфіку діяльності, як лісовідведення, лісозаготівля,

переробка деревинию. Програма надає можливість зручного ведення бухгалтерського і податкового обліку з формуванням всього спектру галузевої і регламентованої звітності, підтримує інтеграцію з системою електронного обліку деревини (Рис 1.1).

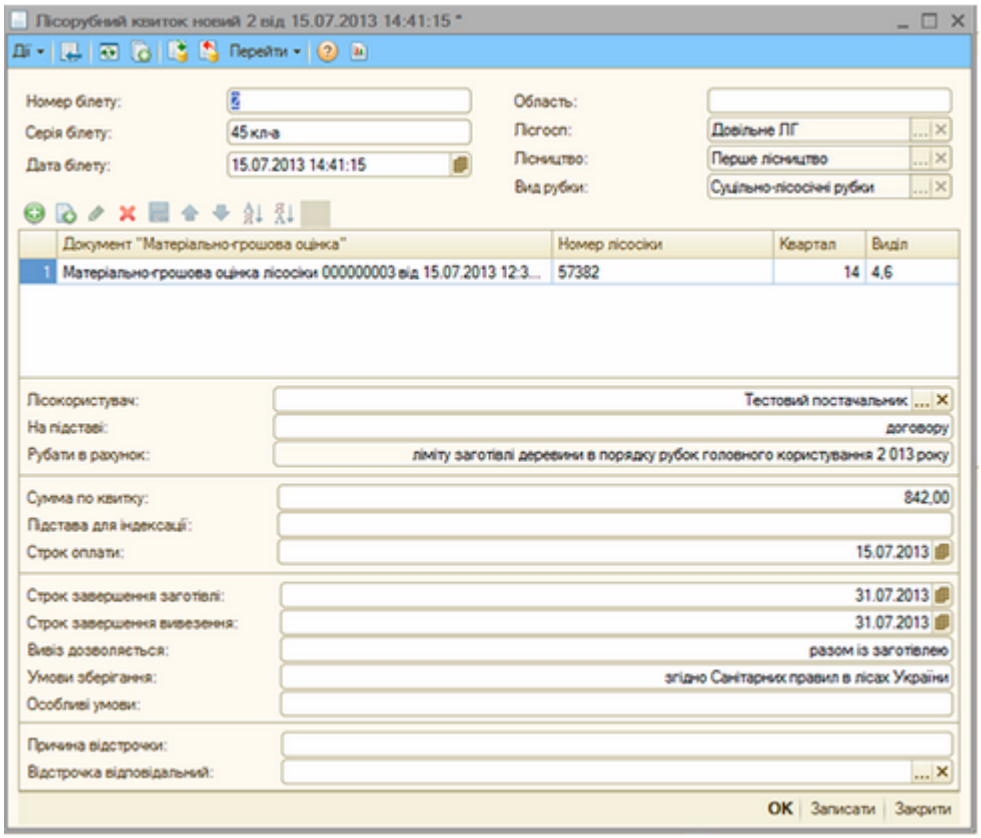


Рис 1.1 – Інтерфейс програми: 1С:Підприємство "управління лісовим господарством"

Для роботи усіх лісових господарств, на даний момент, є створений бухгалтерський облік, а відомча звітність - одна з найскладніших серед всіх виробничих підприємств України та всього світу. Це поєднання всіх даних, переробного виробництва, заробітної плати та валютного обліку, а також бюджетного фінансування та обліку деревини, понад 40 форм внутрішньогалузевої звітності по роботам. На даний момент важко уявити ділянку для обліку, якої не існує в лісових господарствах. уже існує розроблений програмний проект на платформі 1С:Підприємство 8 створений на базі "Бухгалтерії", що дозволило впровадити понад в 40 лісогосподарських

підприємствах україни. Вся звітність, інформаційний облік деревини, відвід, таксація та робоче місце керівника - це те, що є вже в цій конфігурації.

1С:Підприємство "Управління лісовим господарством", не дає в повній мірі вирішити всю проблематику лісничого відділу, так як він зосереджений на обліковіх функції та формуванню звітності. Вартість даного програмного забезпечення - 50000 гривень.

*СофтСервис: "Лесное хозяйство"*

Програма для автоматизації обліку державних лісогосподарських установ (ДЛГУ) та виробничих лісогосподарських об'єднань (ПЛГО). Програма 1С націлена на вимоги обласного ПЛГО та Міністерства лісового господарства Республіки Білорусії (Рис 1.2).

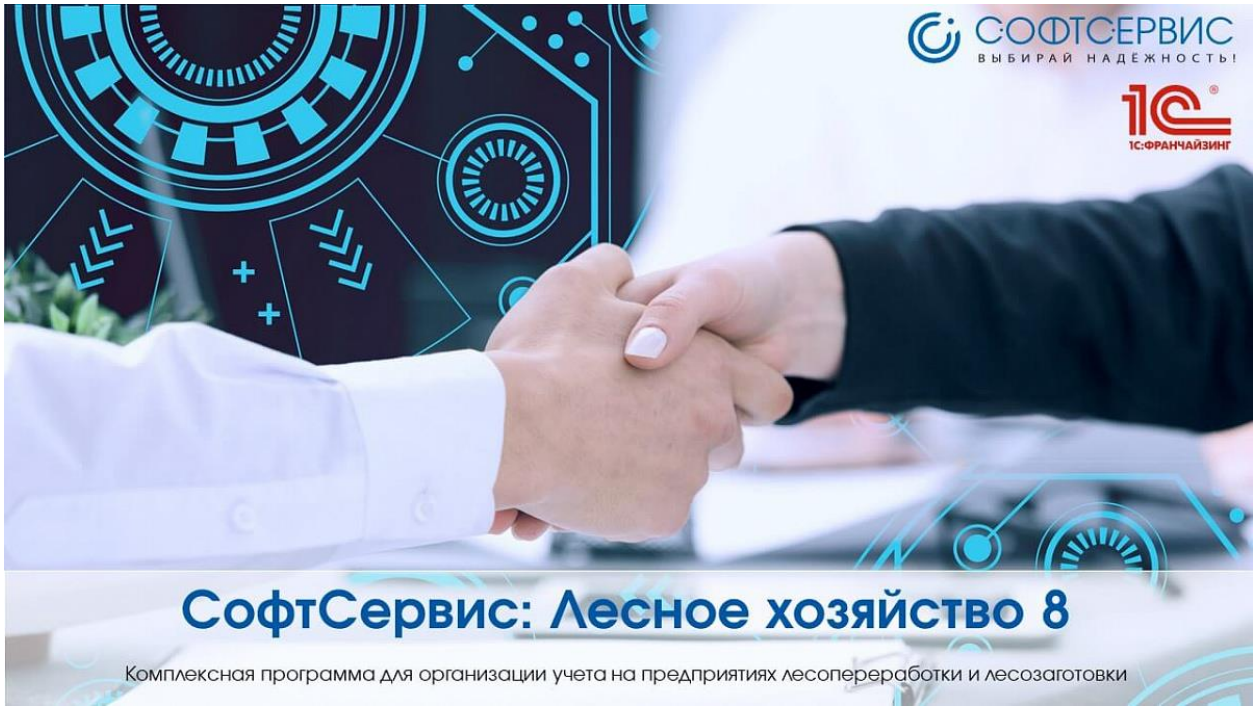


Рис 1.2 – Інтерфейс програми: СофтСервис: "Лесное хозяйство"

Вартість програми СофтСервис: "Лесное хозяйство" - договірна, розраховується індивідуально і залежить від масштабу підприємства, числа робочих місць, специфіки обліку підприємства.

Переваги:

- Ведення заробітної плати за видами фінансування.
- Наявність спеціальних перевірок для аналізу ведення обліку, що дозволяють обчислити помилки у первинних документах до закриття місяця.
- Можливість ведення обліку (лісгоспу та лісництв) в одній базі даних з використанням підключення лісництв через інтернет без необхідності обміну з лісництвами.
- Автоматичне надсилання звітності до Міністерства лісового господарства.
- Повний та остаточний розрахунок заробітної плати у лісництві.
- Автоматичний обмін інформацією із лісництвами.

Функціональні можливості:

1. Облік банківських та касових операцій.
2. Облік основних засобів (ОС) та нематеріальних активів (НМА).
3. Облік товарно-матеріальних цінностей (ТМЦ).
4. Кадровий облік та повний розрахунок заробітної плати
5. Облік лісопродукції.
6. Облік спецодягу.
7. Облік автотранспорту.
8. Облік лісорубних квитків.
9. Облік бланків суворої звітності (БСО).
10. Облік розрахунків із контрагентами.
11. Розподіл витратних рахунків.
12. Загальноживлення.

13. Облік виробництва.

14. Розрахунок реалізованих податків.

15. Облік тварин.

16. Комісійна торгівля.

Аналіз існуючих систем для ведення лісового господарства показує, що існуючі системи не зможуть вирішити проблеми щодо автоматизації лісничого відділу. З урахуванням всіх цих факторів актуальним є проектування, планування та розробка нової інформаційно-аналітичної системи підтримки управління підприємством лісовим господарством

Наявні комерційні продукти мають, здебільшого, надто широке призначення і, відповідно, суттєву надлишковість функціоналу по відношенню до поставлених задач. З іншої сторони, має місце фактор вартості, використання розглянутих продуктів потребує оновлення системи, що може дуже дорого обійтися. Таким чином рішення щодо створення спеціальної системи, орієнтованої саме на конкретні задачі, є обґрунтованим.

### 1.6 Постановка задачі

Дипломна робота припускає:

- розробку схеми БД;

- розробку та реалізацію додатка, що включає такі основні модулі:

1) модуль вводу/редагування та видалення інформації в БД;

2) модуль пошуку інформації;

3) модуль формування звітності.

Реалізація додатка виконується з використанням вивчених технологій. Результат – Windows додаток, що представляє собою **Інформаційно-аналітичну систему підтримки управління лісовим господарством**, яка містить:

**Сутності:**

- Працівник (інформація про працівника): ідентифікатор працівника, прізвище, ім’я, № телефону, домашня адреса, електронний адрес, дата народження.
- Посади (посади працівників лісного господарства): ідентифікатор посади, назва та опис;
- Насадження (дані про насадження): ідентифікатор насадження, назва породи дерева, дата висадки, кількість насаджень, площа насадження.
- Роботи (дані про роботи): ідентифікатор роботи, назва роботи, опис та дата роботи.
- Угіддя (дані про угіддя): ідентифікатор угіддя, назва угіддя, площа угіддя, ідентифікатор насадження, ідентифікатор працівника, ідентифікатор роботи, дата початку роботи, дата закінчення робіт.

**Введення:**

Вводиться і редагується: працівники, посади, роботи.

Проводяться насадження дерев.

Створюються угіддя: визначаються насадження дерев, призначаються робітники для проведення робіт та назначається робота.

**Пошук:**

- По працівниках;
- По угіддях;
- По насадженнях.

**Звітність:**

- По вибраній роботі за період часу: на яких угіддях проводилась дана робота, які працівники були задіяні в даній роботі, площа угіддя, які насадження проводились.
- По вибраній посаді: виводиться інформація про всіх працівників, що відповідають даній посаді.

# *2.* Розробка інформаційно-аналітичної системи підтримки управління лісовим господарством

## 2.1 Аналіз вимог. Use-case діаграми. Основні прецеденти

Згідно з поставленою задачею можна висунути такий список вимог до проекту:

Таблиця 2.1. Функціональні вимоги до додатка «Система підтримки управління лісовим господарством»

| **Вимоги** | **Опис** |
|---|---|
| REQ-1 | Система повинна дозволяти користувачеві додати та редагувати інформацію про робітників |
| REQ-2 | Система повинна дозволяти користувачеві додати та редагувати інформацію про посади працівників |
| REQ-3 | Система повинна дозволяти користувачеві додати та редагувати інформацію про роботи |
| REQ-4 | Система повинна дозволяти користувачеві додати та редагувати інформацію про угіддя |
| REQ-5 | Система повинна дозволяти користувачеві додати та редагувати інформацію про насадження |
| REQ-6 | Система повинна дозволяти користувачеві здійснювати пошук по: працівниках, насадженнях та угіддях |
| REQ-7 | Система повинна дозволяти користувачеві отримувати звітність по вибраній роботі: назва угіддя, інформацію про робітника, дату початку роботи, дату кінця роботи та площу угіддя. |
| REQ-8 | Система повинна дозволяти користувачеві отримувати звітність по вибраній посаді: виводиться детальний протокол по всіх робітниках, що займають дану посаду. |

Таблиця 2.2. Нефункціональні вимоги до додатка «Система підтримки управління лісовим господарством»

| **Вимоги** | **Опис** |
|---|---|
| REQ-10 | Додаток повинен мати простий дизайн та зручну навігації |
| REQ-11 | Поля повинні бути не порожніми, унікальними відносно вже існуючих записів |

Таблиця 2.3. Актори та цілі додатка «Система підтримки управління лісовим господарством»

| **Актори** | **Цілі** |
|---|---|
| Користувач | Мета користувача полягає в роботі з системою. |
| База даних | Мета бази даних полягає у зберігання інформації про лісництво |

Таблиця 2.4. Опис варіантів використання додатка «Система підтримки управління лісовим господарством»

| **Варіант використання** | **Ім’я** | **Опис** |
|---|---|---|
| UC1 | Вивід каталогу працівників | Дозволяє користувачу вивести каталог всіх працівників, які працюють на СТО |
| UC2 | Додати працівника | Дозволяє користувачу додати працівника |
| UC3 | Редагувати працівника | Дозволяє користувачеві редагувати інформацію вибраного із списку працівника |
| UC4 | Видалити працівника | Дозволяє користувачеві видалити вибраного працівника |

| | | |
|---|---|---|
| UC5 | Вивід каталогу посад | Дозволяє користувачу вивести список всіх посад |
| UC6 | Додати посаду | Дозволяє користувачу додати нову посаду |
| UC7 | Редагувати посаду | Дозволяє користувачеві редагувати інформацію вибраної із списку посади |
| UC8 | Видалити посаду | Дозволяє користувачеві видалити вибрану із списку посаду |
| UC9 | Вивід каталогу робіт | Дозволяє користувачу вивести список робіт |
| UC10 | Додати роботу | Дозволяє користувачу додати нову роботу |
| UC11 | Редагувати роботу | Дозволяє користувачеві редагувати інформацію вибраної із списку роботи |
| UC12 | Видалити роботу | Дозволяє користувачеві видалити вибрану роботу |
| UC13 | Вивід каталогу угідь | Дозволяє користувачу вивести список угідь |
| UC14 | Додати угіддя | Дозволяє користувачу додати нове угіддя |
| UC15 | Редагувати угіддя | Дозволяє користувачеві редагувати інформацію вибраного із списку угіддя |
| UC16 | Видалити угіддя | Дозволяє користувачеві видалити вибране угіддя |
| UC17 | Вивід каталогу насаджень | Дозволяє користувачу вивести список насаджень |
| UC18 | Додати насадження | Дозволяє користувачу додати нове насадження |
| UC19 | Редагувати насадження | Дозволяє користувачеві редагувати інформацію вибраного із списку насадження |
| UC20 | Видалити насадження | Дозволяє користувачеві видалити вибране насадження |

| | | |
|---|---|---|
| UC21 | Пошук по працівниках | Дозволяє користувачу здійснювати пошук по працівниках |
| UC22 | Пошук по насадженнях | Дозволяє користувачу здійснювати пошук по насадженнях |
| UC23 | Пошук по угіддях | Дозволяє користувачу здійснювати пошук по угіддях |
| UC24 | Звітність по роботах | Дозволяє користувачеві отримувати звітність по вибраній роботі: назва угіддя, інформацію про робітника, дату початку роботи, дату кінця роботи та площу угіддя |
| UC25 | Звітність по посадах | Дозволяє користувачеві отримувати звітність по вибраній посаді: виводиться детальний протокол по всіх робітниках, що займають дану посаду. |

З поставлених вимог (див. розділ 1) тепер можна виставити повний опис вимог із сценаріями, що будуть вхідними даними для візуального моделювання мовою UML.

**UC1 Вивід каталогу працівників**

Актор: користувач.

Ціль актора: вивести інформацію про працівників.

Задіяний актор: база даних.

Передумова: користувач натиснув на пункт меню «Працівники».

Післяумова: система відображає екран для виведення списку всіх працівників.

**UC2 Додати працівника**

Актор: користувач.

Ціль актора: додати нового працівника на СТО.

Задіяний актор: база даних.

Передумова: користувач ввівши всю необхідну інформацію про працівника натискає на кнопку «Додати».

Післяумова: система додає нового працівника і відображає екран із списком всіх працівників.

**UC3 Редагувати працівника**

Актор: користувач.

Ціль актора: редагувати вибраного із списку працівника.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідної із списку працівника.

Післяумова: система відображає екран для редагування інформації про вибраного працівника.

**UC4 Видалити працівника**

Актор: користувач.

Ціль актора: видалити працівника із бази даних.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідного із списку працівника.

Післяумова: система відображає екран з можливістю видалення даних про працівника.

**UC5 Вивід каталогу посад**

Актор: користувач.

Ціль актора: додати інформацію про посаду.

Задіяний актор: база даних.

Передумова: користувач натиснув на пункт меню «Посади».

Післяумова: система відображає екран для виведення списку всіх посад.

**UC6 Додати посаду**

Актор: користувач.

Ціль актора: додати нову посаду.

Задіяний актор: база даних.

Передумова: користувач ввівши всю необхідну інформацію про посаду натискає на кнопку «Додати».

Післяумова: система додає нову посаду і відображає екран із списком всіх посад.

**UC7 Редагувати посаду**

Актор: користувач.

Ціль актора: редагувати інформацію про вибрану посаду.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідну із списку посаду.

Післяумова: система відображає екран для редагування інформації про вибрану посаду.

**UC8 Видалити посаду**

Актор: користувач.

Ціль актора: видалити посаду із бази даних.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідну із списку посаду.

Післяумова: система відображає екран з можливістю видалення даних про посаду.

**UC9 Вивід каталогу робіт**

Актор: користувач.

Ціль актора: вивести інформацію про всі роботи.

Задіяний актор: база даних.

Передумова: користувач натиснув на пункт меню «Роботи».

Післяумова: система відображає екран для виведення списку всіх робіт.

**UC10 Додати роботу**

Актор: користувач.

Ціль актора: додати нову роботу.

Задіяний актор: база даних.

Передумова: користувач ввівши всю необхідну інформацію про роботу натискає на кнопку «Додати».

Післяумова: система додає нову роботу і відображає екран із списком всіх робіт.

**UC11 Редагувати роботу**

Актор: користувач.

Ціль актора: редагувати вибрану із списку роботу.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідну із списку роботу.

Післяумова: система відображає екран для редагування інформації про вибрану роботу.

**UC12 Видалити роботу**

Актор: користувач.

Ціль актора: видалити роботу із бази даних.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідну із списку роботу.

Післяумова: система відображає екран з можливістю видалення даних про роботу.

**UC13 Вивід каталогу угідь**

Актор: користувач.

Ціль актора: вивести інформацію про всі угіддя.

Задіяний актор: база даних.

Передумова: користувач натиснув на пункт меню «Угіддя».

Післяумова: система відображає екран для виведення списку всіх угідь.

**UC14 Додати угіддя**

Актор: користувач.

Ціль актора: додати нове угіддя.

Задіяний актор: база даних.

Передумова: користувач ввівши всю необхідну інформацію про угіддя натискає на кнопку «Додати».

Післяумова: система додає нове угіддя і відображає екран із списком всіх угідь.

**UC15 Редагувати угіддя**

Актор: користувач.

Ціль актора: редагувати вибране із списку угіддя.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідне із списку угіддя.

Післяумова: система відображає екран для редагування інформації про вибране угіддя.

**UC16 Видалити угіддя**

Актор: користувач.

Ціль актора: видалити угіддя із бази даних.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідне із списку угіддя.

Післяумова: система відображає екран з можливістю видалення даних про угіддя.

**UC17 Вивід каталогу насаджень**

Актор: користувач.

Ціль актора: вивести інформацію про всі насадження.

Задіяний актор: база даних.

Передумова: користувач натиснув на пункт меню «Насадження».

Післяумова: система відображає екран для виведення списку всіх насаджень.

**UC18 Додати насадження**

Актор: користувач.

Ціль актора: додати нове насадження.

Задіяний актор: база даних.

Передумова: користувач ввівши всю необхідну інформацію про насадження натискає на кнопку «Додати».

Післяумова: система додає нове  насадження і відображає екран із списком всіх насаджень.

**UC19 Редагувати насадження**

Актор: користувач.

Ціль актора: редагувати вибране із списку насадження.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідне із списку насадження.

Післяумова: система відображає екран для редагування інформації про вибране насадження.

**UC20 Видалити насадження**

Актор: користувач.

Ціль актора: видалити насадження із бази даних.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідне із списку насадження.

Післяумова: система відображає екран з можливістю видалення даних про насадження.

**UC21 Пошук  по працівниках**

Актор: користувач.

Ціль актора: здійснити пошук всіх працівників за прізвищем або ім’ям.

Задіяний актор: база даних.

Передумова: користувач вибирає пункт меню «Пошук» -> «Працівників».

Післяумова: система відображає екран з можливістю пошуку працівників.

**UC22 Пошук  по насадженнях**

Актор: користувач.

Ціль актора: здійснити пошук насаджень по породі дерева або площею насаджень.

Задіяний актор: база даних.

Передумова: користувач вибирає пункт меню «Пошук» -> «Насадження».

Післяумова: система відображає екран з можливістю пошуку насаджень.

**UC23 Пошук  по угіддях**

Актор: користувач.

Ціль актора: здійснити пошук угідь по назві або площі.

Передумова: користувач вибирає пункт меню «Пошук» -> «Угіддя».

Післяумова: система відображає екран з можливістю пошуку угідь.

**UC24 Звітність по роботах**

Актор: користувач.

Ціль актора: створити звіт по вибраній роботі.

Задіяний актор: база даних.

Передумова: користувач вибирає пункт меню «Звіти» -> «По роботі».

Післяумова: система відображає екран з можливістю створення звітів по вибраній роботі, а саме: назва угіддя, інформацію про робітника, дату початку роботи, дату кінця роботи та площу угіддя.

**UC25 Звітність по посадах**

Актор: користувач.

Ціль актора: вивести звіт по вибраній посаді.

Задіяний актор: база даних.

Передумова: користувач вибирає пункт меню «Звіти» -> «По посаді».

Післяумова: система відображає екран з можливістю створення звітів по вибраному ремонту, а саме дає можливість вивести детальний протокол по всіх робітниках, що займають дану посаду.

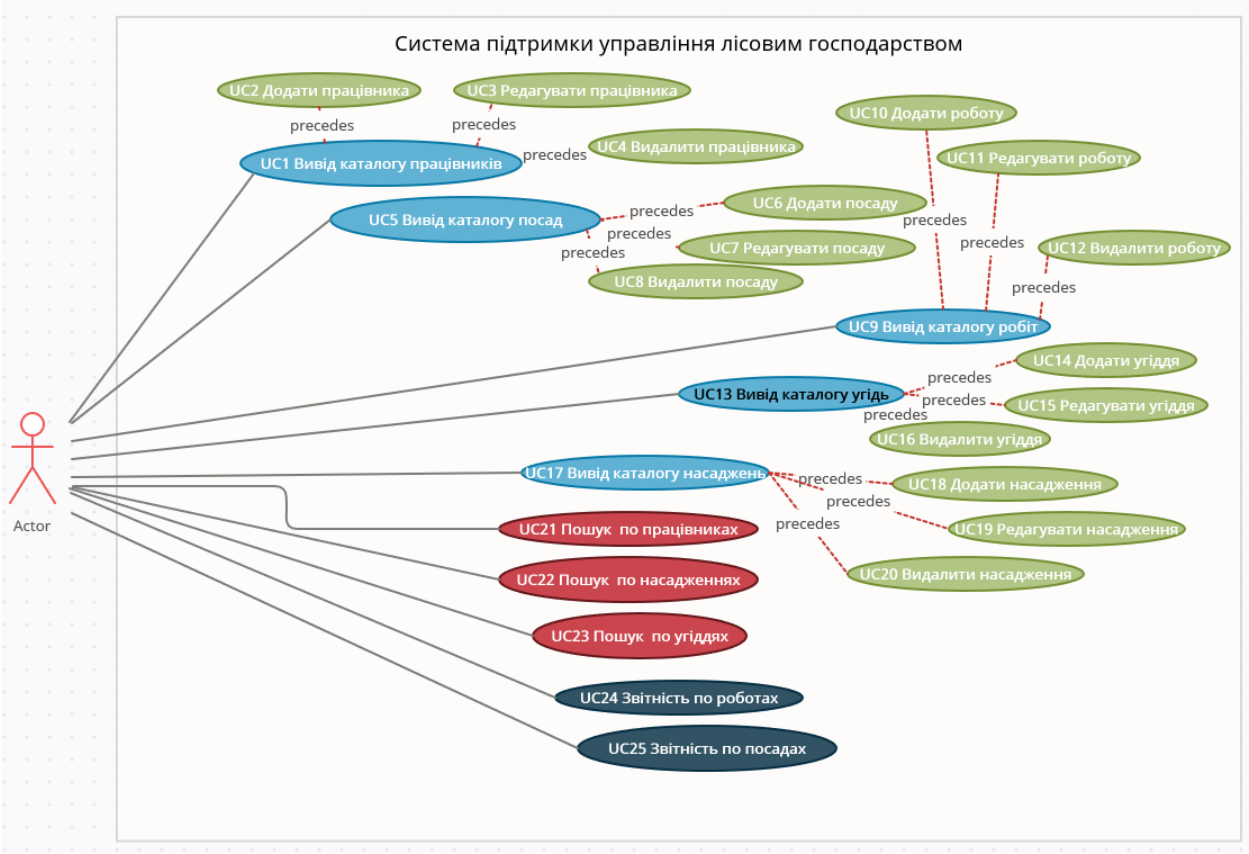


Рис. 2.1. – Діаграма use-case

## 2.2 Архітектура проекту

Розроблений продукт повинен відповідати характеристикам якості, таким як: стійкість, корисність, доступність, масштабованість, відкритість, гнучкість, можливість тестування. Це вимагає від процесу розробки додаткові обмеження/правила, а саме:

- дотримання шаблонів і стилів;
- документування розробки на різних рівнях;
- тестування компонентів, окремих модулів, підсистем;

- управління проектами, процесами.

З урахуванням вимог до забезпечення стійкості та гнучкості системи при її розробці було обрано шаблон Layers, який розбиває систему на дві частини: клієнт та сервер.

Проектування системи буде покладатись на предметну область (DDD підхід) та принципи SOLID.

При розробці клієнта був обраний користувацький інтерфейс Windows Forms.

Сервер, в свою чергу, буде складатись з таких модулів:

1) BLL (Business Logic Layer) – логіка та всі необхідні обчислення додатку на мові бізнесу;
2) DAL (Data Acces Layer) – рівень доступу до даних.
3) DB (Data Base) – база даних для зберігання даних.

З урахування всіх вище перерахованих шаблонів структура проекту буде виглядати наступним чином:

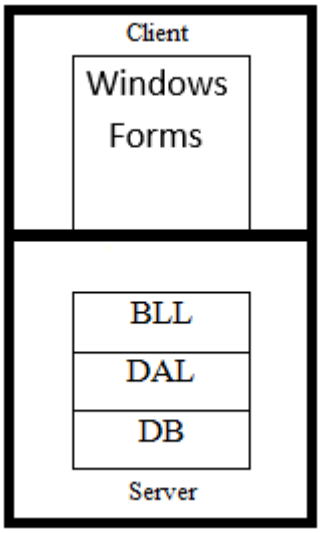


Рис. 2.2. – Структура проекту

Кожен компонент буде реалізувати контракт (інтерфейс), який надає на гнучкість компоненту.

Оскільки бізнес постійно і відносно швидко змінюється, то кожен із компонентів повинен швидко адаптуватись під ці зміни. Для цього використаємо шаблон Dependency Injection (DI).

### 2.3 Особливості розробки бази даних. Фізична та логічна модель бази даних з описанням сутностей

Логічне проектування полягає в створенні логічної моделі на основі вибраної моделі даних. Згідно постановки задачі будуємо логічну модель бази даних (Рис 2.3).

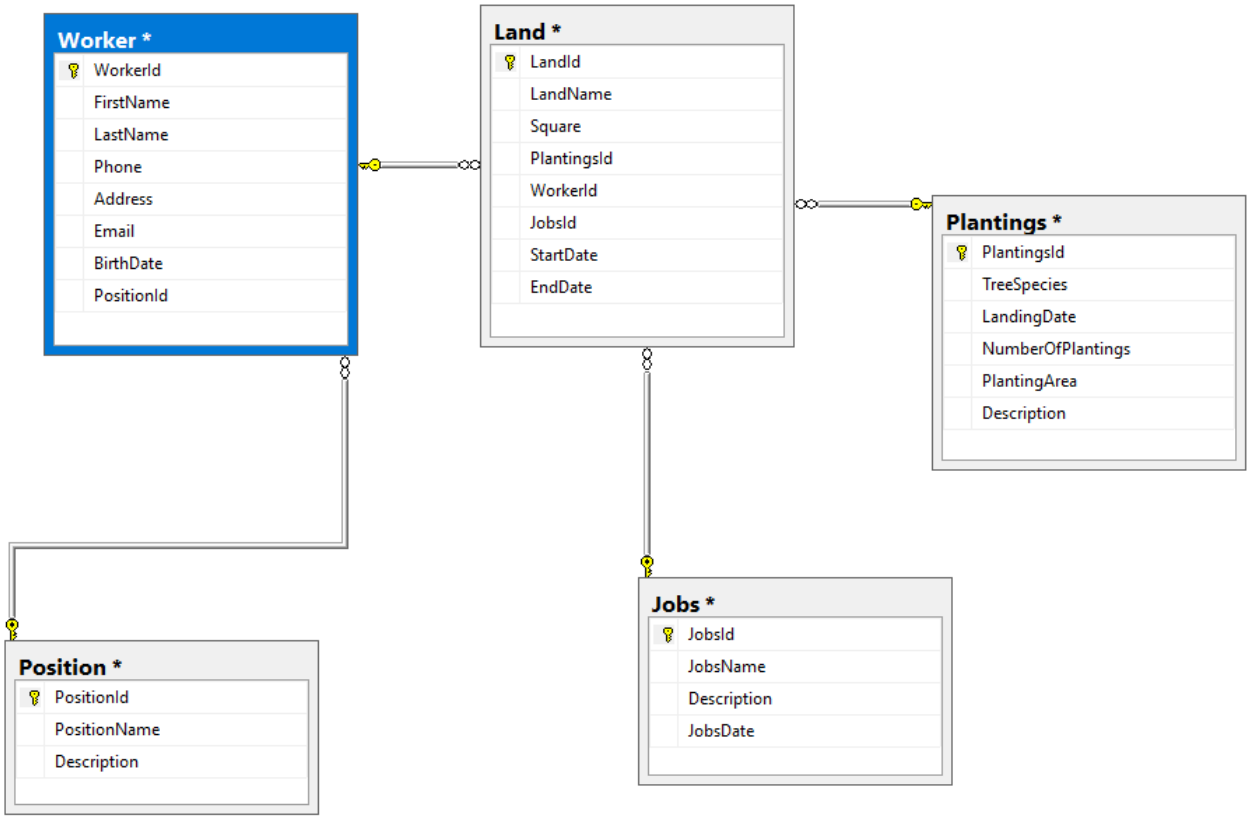


Рис. 2.3 – Логічна модель бази даних.

Фізичне проєктування — створення схеми бази даних для конкретної СУБД. Специфіка конкретної СУБД може включати в себе обмеження на іменування об'єктів бази даних, обмеження на підтримувані типи даних та інші. Крім того, специфіка конкретної СУБД при фізичному проєктуванні включає вибір рішень, пов'язаних з фізичним середовищем зберігання даних (вибір методів управління дисковою пам'яттю, поділ БД по файлам і пристроям, методів доступу до даних), створення індексів та інші.

Фізичне проектування полягає в описі засобів фізичної реалізації логічного проекту бази даних. Фізичні моделі визначають засоби розміщення даних в середовищі зберігання і засоби доступу до цих даних, які підтримуються на фізичному рівні. У процесі проектування визначається структура реляційної бази даних (склад таблиць, їх структура і логічні зв'язки). Структура таблиці визначається складом стовпців, типом даних, ключами таблиці (Рис 2.4).

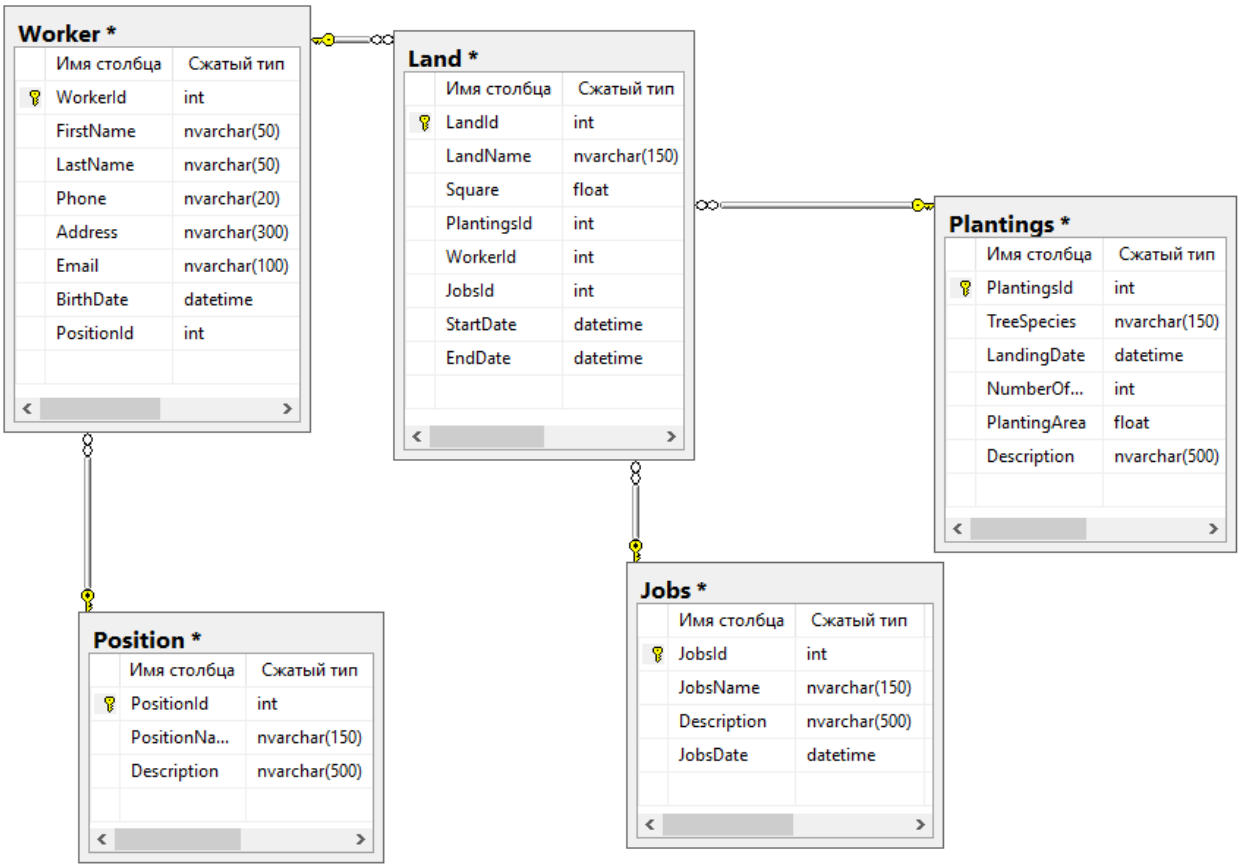


Рис. 2.4 – Фізична модель бази даних.

Як ми бачимо на рис. 2.3- 2.4 наша база даних складається із 5 сутностей. Кожна сутність має свою таблицю, а саме:

1. Таблиця «Worker» – зберігає інформацію про всіх працівників.
2. Таблиця «Land» - зберігає інформацію про всі угіддя лісового господарства.
3. Таблиця «Plantings» - зберігає інформацію про всі насадження в лісовому господарстві.

4. Таблиця «Position» - зберігає інформацію про всі посади, які є у лісовому господарстві.
5. Таблиця «Jobs» - зберігає інформацію про опис робіт, який проводяться у лісовому господарстві.

### 2.4 Особливість реалізації бізнес логіки – диаграмма домена.

Будуємо діаграму класів домена. Кожен клас описує конкретну таблицю бази даних для зручного опрацювання даних.

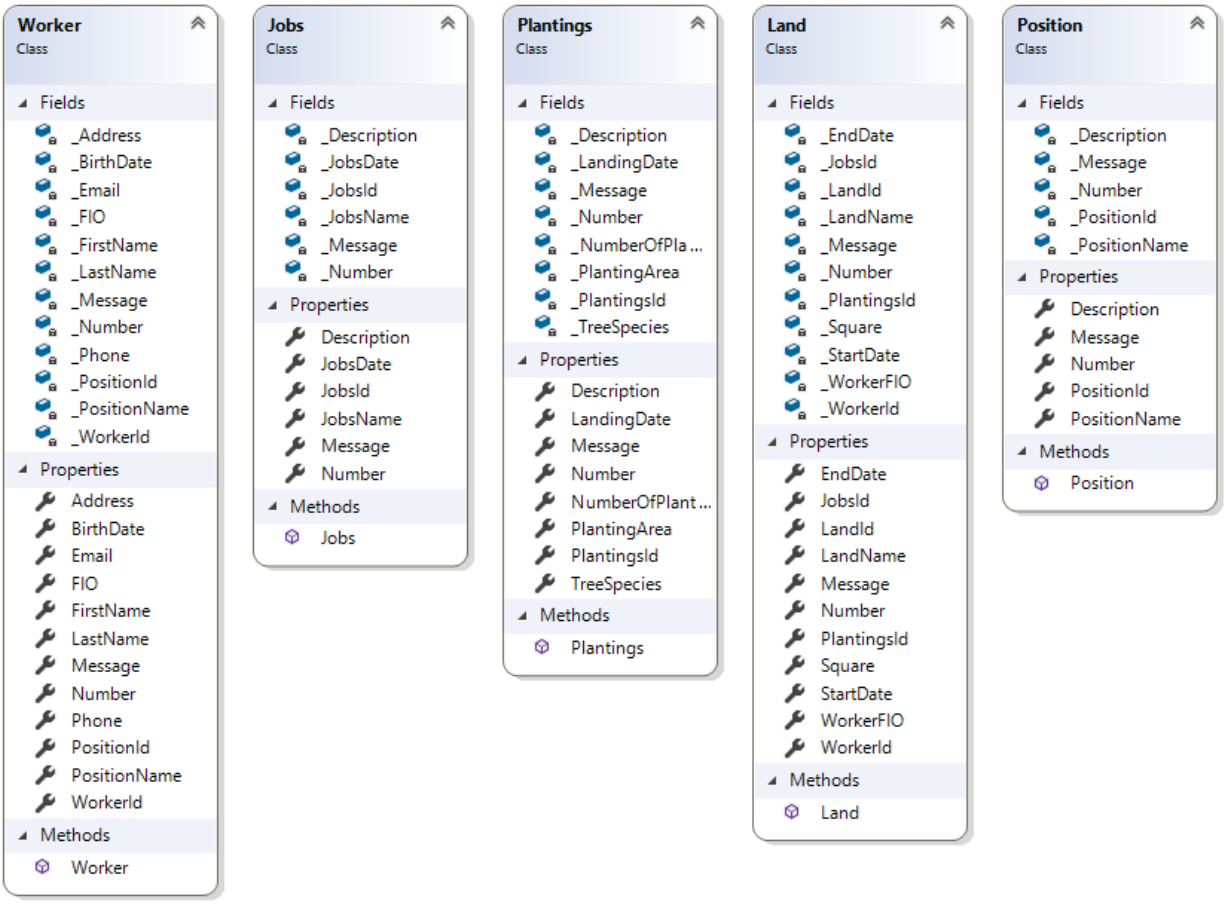


Рис. 2.5 – Діаграма класів домена

### 2.5 Особливості розробки рівня UI

В даному проекті для розробки користувацького інтерфейсу були використані Windows Forms.

Windows Forms - це платформа користувача інтерфейсу для створення класичних додатків Windows. Вона забезпечує один з найефективніших

способів створення класичних додатків за допомогою візуального конструктора в Visual Studio.

На рис. 2.6 показана діаграма додатку «Система підтримки управління лісовим господарством».

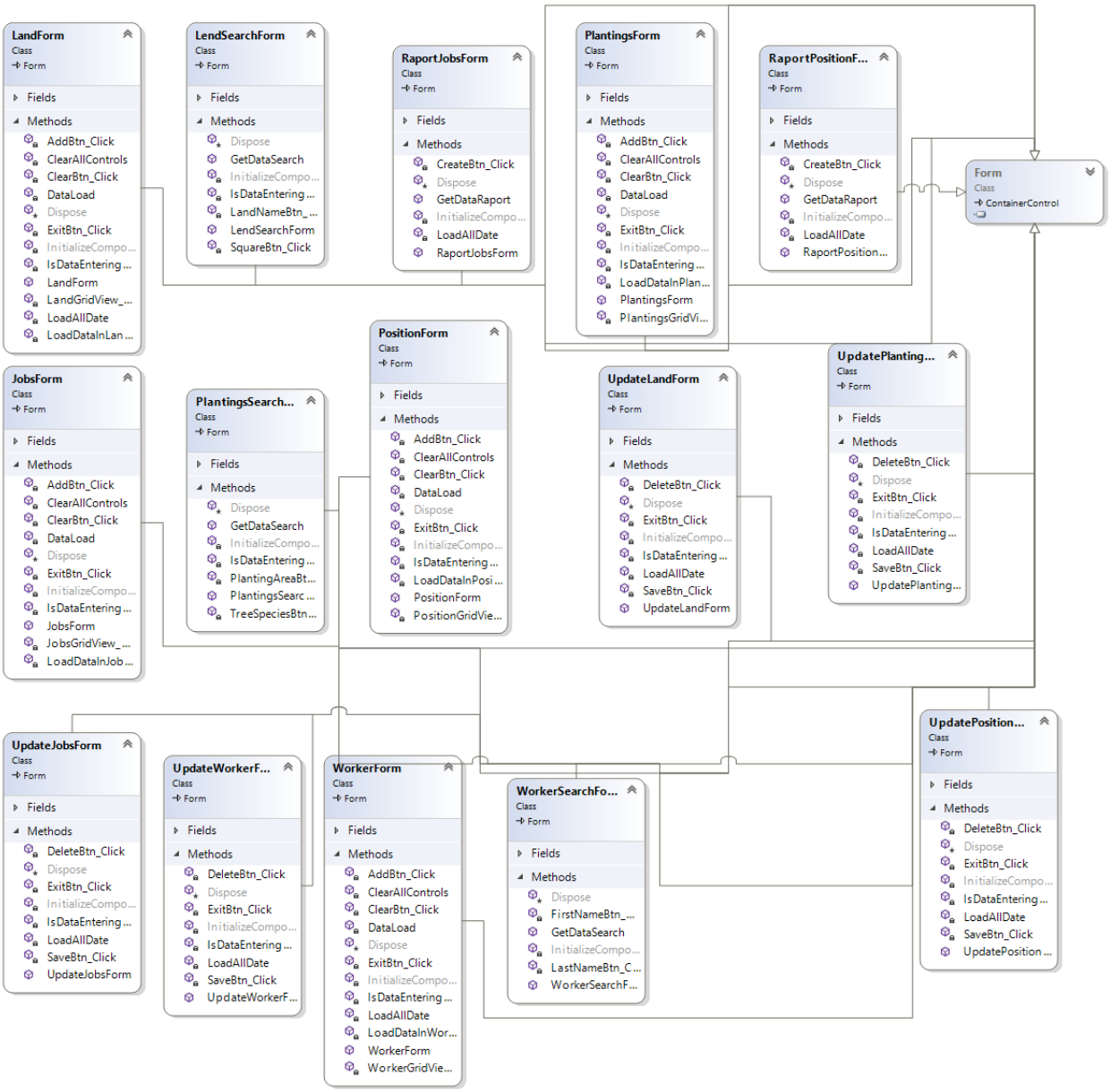


Рис. 2.6 – Діаграма інтерфейсу додатку «Система підтримки управління лісовим господарством»

### 2.6 Особливості розробки DAL

Для роботи з базою даних було реалізовано 5 класів. Назва кожного класу відповідає назві таблиці в базі даних. Нижче описано роботу кожного класу та всіх публічних методів:

1. Клас «JobsProvider» містить 5 методів, які призначені для додавання, вибірки, редагування, вибірки всього списку та видалення робіт із бази даних.
2. Клас «PositionProvider» містить 5 методів для додавання нових посад, редагування даних про посади, вибірки списку всіх посад, вибірки певної посади за її ідентифікатором та видалення посади із бази даних при необхідності.
3. Клас «LandProvider» призначений для роботи з таблицею «угіддя» містить 8 методів, а саме:

   3.1. Метод «InsertLand» – призначений для додавання нової інформації про угіддя;

   3.2. Метод «SelectedLandByLandId» – призначений для вибору всієї інформації про одне угіддя. Для його виклику необхідно передати ідентифікатор угіддя.

   3.3. Метод «GetAllLand» – повертає список всіх угідь.

   3.4. Метод «UpdateLand» – призначений для редагування інформації вибраного із списку угіддя.

   3.5. Метод «DeleteLandByLandId» – призначений для видалення даних про угіддя.

   3.6. Метод «GetAllLandByJobsId» – призначений для вибору всіх угідь, в яких була призначена робота.

   3.7. Метод «GetAllLandByLandName» – призначений для повернення списку всіх угідь по назві або частини назви угіддя.

   3.8. Метод «GetAllLandBySquare» – призначений для повернення списку всіх угідь, площа яких менша за вказану у параметрі пошуку.

4. Клас «PlantingsProvider» містить 7 методів та призначений для роботи із насадженням:

   4.1. Метод «InsertPlantings» – призначений для додавання нової інформації про насадження;

   4.2. Метод «SelectedPlantingsByPlantingsId» – призначений для вибору всієї інформації про одне насадження. Для його виклику необхідно передати ідентифікатор насадження.

   4.3. Метод «GetAllPlantings» – повертає список всіх насаджень.

   4.4. Метод «GetAllPlantingsByTreeSpecies» – призначений для повернення списку всіх насаджень по назві або частини назви порід дерева.

   4.5. Метод «GetAllLandBySquare» – призначений для повернення списку всіх насаджень, площа насаджень яких менша за вказану у параметрі пошуку.

   4.6. Метод «UpdatePlantings» – призначений для редагування інформації вибраного із списку насадження.

   4.7. Метод «DeletePlantingsByPlantingsId» – призначений для видалення даних про насадження.

5. Клас «WorkerProvider» містить 8 методів для роботи з даними працівників лісного господарства, саме:

   5.1. Метод «InsertWorker» – призначений для додавання інформації про нового працівника;

   5.2. Метод «SelectedWorkerByWorkerId» – призначений для вибору інформації про одного працівника. Для його виклику необхідно передати ідентифікатор працівника;

   5.3. Метод «GetAllPlantings» – повертає список всіх працівників, що зареєстровані у програмі;

   5.4. Метод «GetAllWorkerByFirstName» – призначений для пошуку та повертає список всіх працівників по його імені або частини імені;

5.5. Метод «GetAllWorkerByLastName» – призначений для пошуку та повертає список всіх працівників по його прізвищу або частини прізвища;

5.6. Метод «GetAllWorkerByPositionId» – повертає список всіх працівників, яким належить одна посада. Призначений для формування звітності по вибраній посаді.

5.7. Метод «UpdatePlantings» – призначений для редагування інформації вибраного із списку працівника;

5.8. Метод «DeletePlantingsByPlantingsId» – призначений для видалення даних про працівника.

На рис. 2.7 приведено діаграму класів з методами для роботи з базою даних.

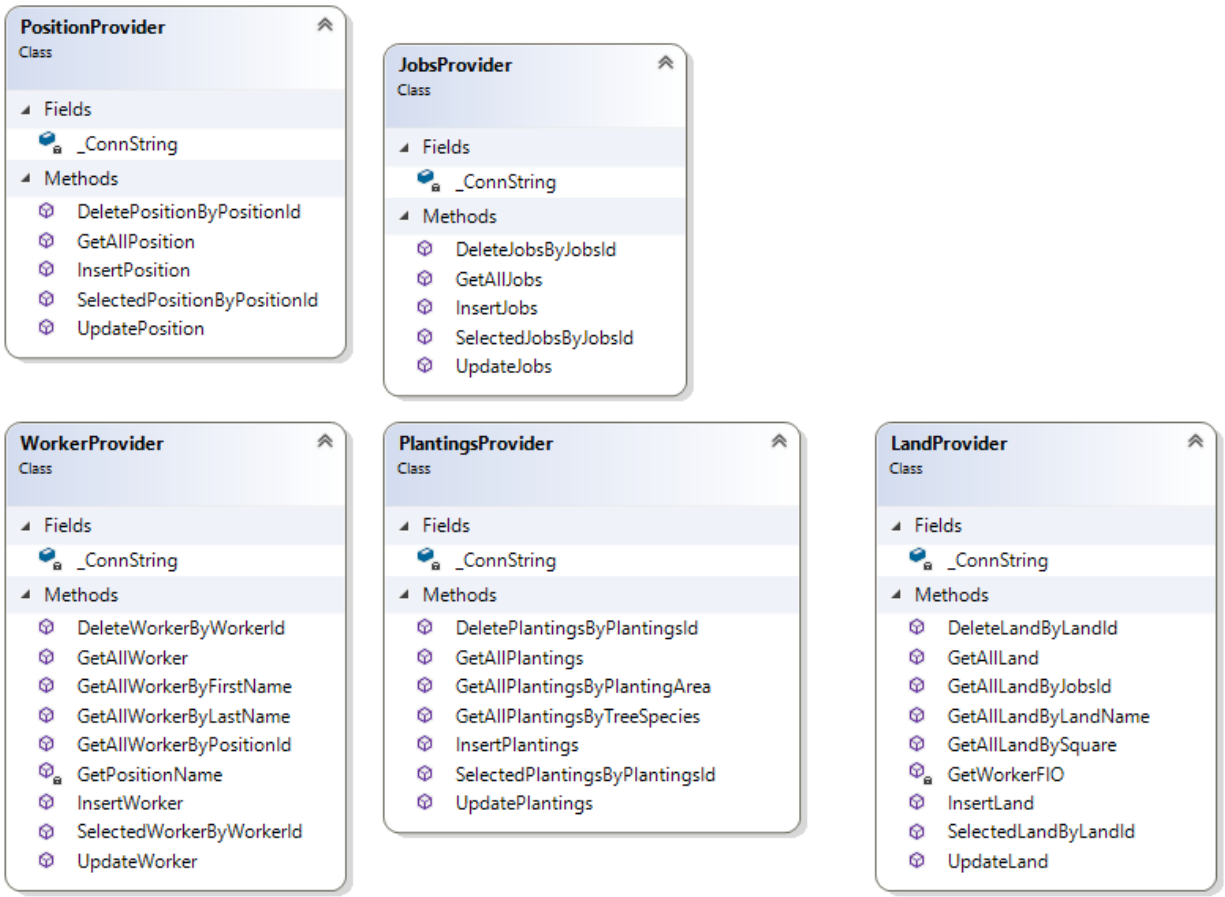


Рис. 2.7 – Діаграма класів з методами для роботи з базою даних

# *3* Реалізація

## 3.1 Вибір технологій

В якості середовища для розробки програмного забезпечення було вибрано програму MS Visual Studio.

Для побудови користувацького інтерфейсу по шаблону будемо використовувати WinForms. Мова програмування буде C#.

Windows Forms – технологія інтелектуальних клієнтів для NET Framework. Вона являє собою набір керованих бібліотек, що спрощують виконання стандартних завдань, як робота із базою даних. За допомогою такого середовища розробки, як Visual Studio, можна створювати інтелектуальні клієнтські програми Windows Forms, які відображають інформацію, вимагають введення від користувачів та обмінюються даними з віддаленими комп'ютерами по мережі.

У Windows Forms форма – це візуальна поверхня, на якій відображається інформація для користувача. Зазвичай програма Windows Forms будується шляхом розміщення елементів керування на форму та написання коду для реагування на дії користувача, такі як клацання миші або натискання клавіш. Елемент управління — це окремий елемент інтерфейсу користувача, призначений для відображення або введення даних.

При виконанні користувачем будь-якої дії з формою або одним з елементів управління створюється подія. Програма реагує на ці події за допомогою коду та обробляє події при їх виникненні. Докладніше див. у розділі Створення обробників подій у Windows Forms.

Windows Forms включає широкий набір елементів керування, які можна додавати на форми: текстові поля, кнопки, списки, перемикачі і навіть веб-сторінки. Список всіх елементів керування, які можна використовувати у формі, наведено в розділі Елементи керування для використання у формах Windows Forms. Якщо існуючий елемент керування не відповідає потребам, у

Windows Forms можна створити власні елементи керування за допомогою класу UserControl.

Для розробки інформаційної бази використовувався MS SQL Server.

SQL Server є однією з найпопулярніших систем управління базами даних (СУБД) у світі. Ця СУБД підходить для різних проектів: від невеликих додатків до великих високонавантажених проектів.

SQL Server характеризується такими особливостями як:

- Продуктивність. SQL Server працює дуже швидко.
- Надійність та безпека. SQL Server надає шифрування даних.
- Простота. З цієї СУБД щодо легко працювати та вести адміністрування.

Центральним аспектом у MS SQL Server, як і будь-який СУБД, є база даних. База даних представляє сховище даних, організованих певним способом. Нерідко фізично база даних представляє файл на жорсткому диску, хоча така відповідність необов'язково. Для зберігання та адміністрування баз даних застосовуються системи управління базами даних (database management system) або СУБД (DBMS). І саме MS SQL Server є однією з таких СУБД.

Для організації бази даних MS SQL Server використовує реляційну модель. Ця модель баз даних була розроблена ще в 1970 Едгаром Коддом. А на сьогодні вона фактично є стандартом для організації бази даних.

Реляційна модель передбачає зберігання даних у вигляді таблиць, кожна з яких складається з рядків та стовпців. Кожен рядок зберігає окремий об'єкт, а стовпці розміщуються атрибути цього об'єкта.

Для взаємодії з базою даних використовується мова SQL (Structured Query Language).

Для роботи з Базою даних реалізовуємо класи (рис.3.1). В кожному класі є методи, за допомогою яких можна: вибирати, вставляти, редагувати та видаляти дані. Також є методи, які працюють з вибіркою даних із декількох таблиць.

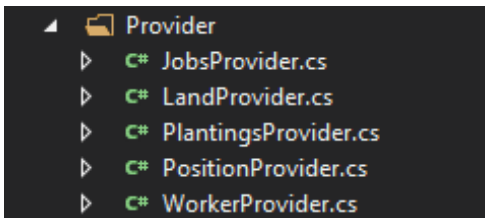


Рис.3.1 – Класи для роботи з базою даних

### 3.2 Результати функціонального тестування розробленого додатку

Для розробленого програмного забезпечення було проведене функціональне тестування.

**Тестовий приклад:** № 1.

**Призначення:** перевірка того, що програмна система дозволяє виконувати читання даних із бази даних та коректно відображати їх у графічному інтерфейсі користувача (Таб. 3.1.).

**Тест-вимоги, що перевіряються:** вивести дані про працівників з бази даних.

**Передумови для тесту:** програмна система повинна бути запущена, а на диску комп'ютера має знаходитися файл із даними у визначеному форматі.

**Критерій проходження тесту:** реальна поведінка програмної системи збігається з очікуваною.

Таблиця 3.1. Виведення каталогу працівників.

| № з/п | Крок сценарію | Очікуваний результат | Отриманий результат | Відмітка про проходження кроку сценарію (Так/Ні) |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1 | У меню "Довідник" вибрати пункт "Працівники" | Повинно з’явитися дочірнє вікно в якому виводиться список всіх працівників | З’являється дочірнє вікно в якому виведено список всіх працівників | Так |

**Тестовий приклад:** № 2.

**Призначення:** перевірка того, що програмна система дозволяє виконувати записувати дані в базу даних та коректно відображати їх у графічному інтерфейсі користувача (Таб. 3.2.).

.

**Тест-вимоги, що перевіряються:** записати дані про нового співробітника в базу даних.

**Передумови для тесту:** програмна система повинна бути запущена, а на диску комп'ютера має знаходитися базу даних із даними у визначеному форматі.

**Критерій проходження тесту:** реальна поведінка програмної системи збігається з очікуваною.

Таблиця 3.2. Додавання нового співробітника.

| № з/п | Крок сценарію | Очікуваний результат | Отриманий результат | Відмітка про проходження кроку сценарію (Так/Ні) |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1 | У меню "Довідник" вибрати пункт "Працівники" | Повинно з’явитися дочірнє вікно в якому виводиться список всіх працівників | З’являється дочірнє вікно в якому виведено список всіх працівників | Так |
| 2 | В лівій частині дочірнього вікна не заповнюємо дані про нового співробітника та натискаємо кнопку «Додати» | Повинна відобразитись підказка про те, що дані про нового співробітника були введені не коректно | Програма відображає підказку про те, що дані про нового співробітника були введені не коректно | Так |
| 3 | В лівій частині дочірнього вікна заповнюємо дані про нового співробітника та | Інформація про нового співробітника повинна записатись в | Інформація про нового співробітника записалась в базу даних та | Так |

| | | | | |
|---|---|---|---|---|
| | натискаємо «Додати» | базу даних та відобразитись в каталозі працівників | відобразилась в каталозі працівників. | |

**Тестовий приклад:** № 3.

**Призначення:** перевірка того, що програмна система дозволяє виконувати читання даних із бази даних та коректно відображати їх у графічному інтерфейсі користувача (Таб. 3.3.).

**Тест-вимоги, що перевіряються:** вивести дані про угіддя з бази даних.

**Передумови для тесту:** програмна система повинна бути запущена, а на диску комп'ютера має знаходитися файл із даними у визначеному форматі.

**Критерій проходження тесту:** реальна поведінка програмної системи збігається з очікуваною.

Таблиця 3.3. Виведення каталогу угідь.

| № з/п | Крок сценарію | Очікуваний результат | Отриманий результат | Відмітка про проходження кроку сценарію (Так/Ні) |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1 | У меню "Створити" вибрати пункт "Угіддя" | Повинно з’явитися дочірнє вікно в якому виводиться список всіх угідь | З’являється дочірнє вікно в якому виведено список всіх угідь | Так |

**Тестовий приклад:** № 4.

**Призначення:** перевірка того, що програмна система дозволяє виконувати записувати дані в базу даних та коректно відображати їх у графічному інтерфейсі користувача (Таб. 3.4.).

.

**Тест-вимоги, що перевіряються:** записати дані про нове угіддя в базу даних.

**Передумови для тесту:** програмна система повинна бути запущена, а на диску комп'ютера має знаходитися базу даних із даними у визначеному форматі.

**Критерій проходження тесту:** реальна поведінка програмної системи збігається з очікуваною.

Таблиця 3.4. Додавання нового угіддя.

| № з/п | Крок сценарію | Очікуваний результат | Отриманий результат | Відмітка про проходження кроку сценарію (Так/Ні) |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1 | У меню "Довідник" вибрати пункт "Угіддя" | Повинно з’явитися дочірнє вікно в якому виводиться список всіх угідь | З’являється дочірнє вікно в якому виведено список всіх угідь | Так |
| 2 | В лівій частині дочірнього вікна не заповнюємо дані про нове угіддя та натискаємо кнопку «Додати» | Повинна відобразитись підказка про те, що дані про нове угіддя були введені не коректно | Програма відображає підказку про те, що дані про нове угіддя були введені не коректно | Так |
| 3 | В лівій частині дочірнього вікна заповнюємо дані про нове угіддя та натискаємо «Додати» | Інформація про нове угіддя повинна записатись в базу даних та відобразитись в каталозі угідь | Інформація про нове угіддя записалась в базу даних та відобразилась в каталозі угідь. | Так |

**Відмітка про проходження тесту (пройдено/не пройдено):** пройдений.

**Тестових прикладів виконано:** 4.
**Тестових прикладів пройдено:** 4.+

### 3.3 Інструкція користувачеві програми

#### 3.3.1 Призначення програмного продукту

Програмний продукт призначено для підтримки управління лісовим господарством. Програми дозволяє працювати з нею користувачам лісного господарства. Користувач може працювати із такими інструментами: створення, видалення, редагування інформації про працівників, посади, роботи, додавати інформацію про нові насадження та угіддя. Також користувач системи може здійснювати пошук та формувати звітність для організації по посадах та проведених роботах.

#### 3.3.2 Використання програмного продукту

**Запуск програми**

Запуск програми в операційній системі сімейства Windows здійснюється одним з стандартних способів:

а) подвійним клацанням лівою кнопкою миші на ярлику програми;

б) викликом контекстного меню з вибором його пункту "Відкрити";

в) натисканням кнопки "Пуск" панелі завдань із подальшим вибором пункту "Усі програми" та подвійним клацанням лівою кнопкою миші на ярлику програми.

**Вхід користувача в систему**

Після запуску програми на екрані монітора з'являється батьківське вікно програми з головним меню (рис. 3.2).

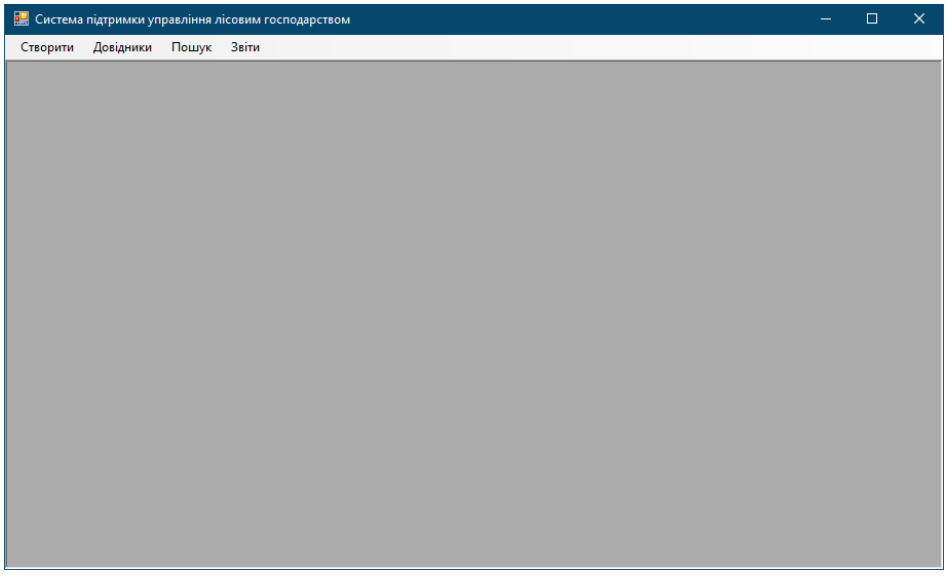


Рис. 3.2. Батьківське вікно програми з головним меню.

Меню програми містить усі команди для керування її виконанням. Воно має наступну структуру:

а) "Створити"

1. "Угіддя";

2. «Насадження»;

3. "Вихід".

б) "Довідники"

1. "Працівники";

2. «Посади»;

3. «Роботи».

в) "Пошук":

1. «Працівників»;

2. «Угіддя»;

3. «Насадження».

г) «Звіти».

1. «По роботі»;

2. «По посаді».

## Робота з програмою

На початку необхідно запустити додаток «Система підтримки управління сільським господарством».

Для початку роботи необхідно наповнити інформаційну систему, а саме: інформацію про посади, інформацію про працівників та роботи.

1) Інтерфейс вікна для виводу каталогу всіх посад (Рис 3.3).

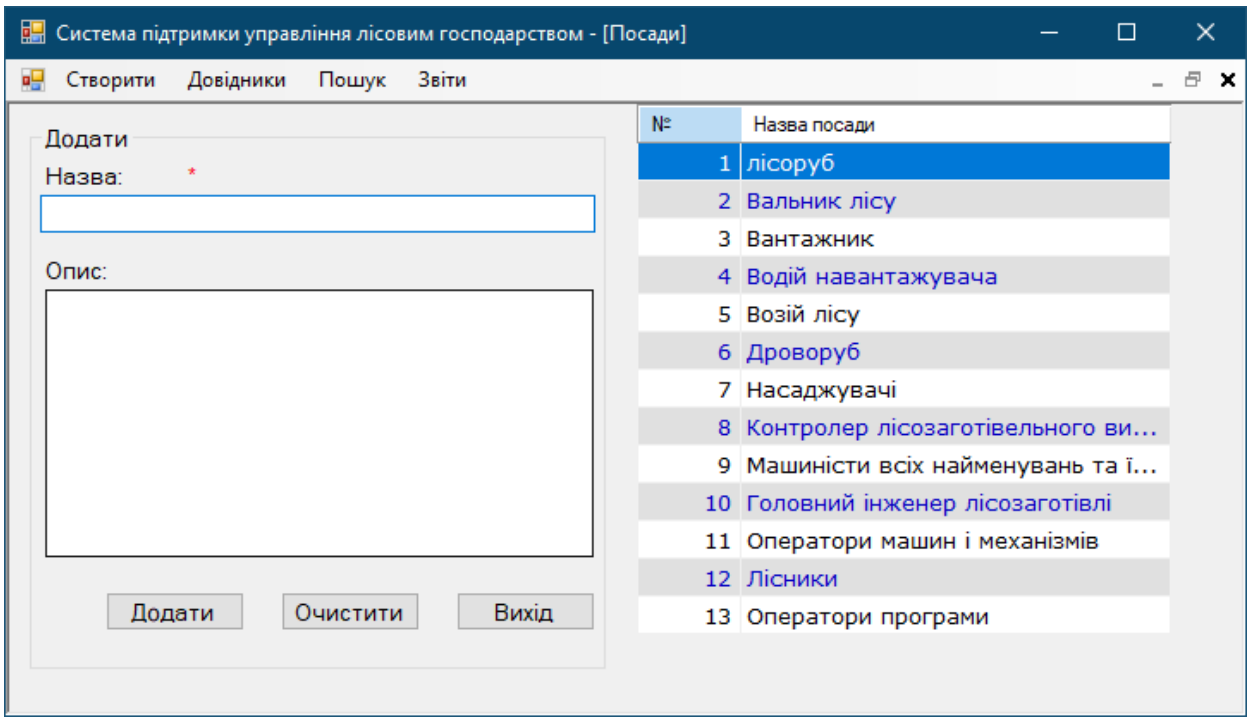


Рис. 3.3 – Вікно для роботи з довідником «Посади».

Дане вікно призначене для ведення інформації про всі посади лісного господарства. Тут можна побачити каталог вже існуючих записів про посади, та додати інформацію про нові посади. Для того, щоб додати інформацію про нову посаду, необхідно в лівій частині вікна заповнити текстові поля даними, тобто вказати наступні дані:

1. Назву посади;
2. Опис посади (не обов’язкове поле).

2) Інтерфейс вікна для редагування даних вибраної посади (Рис 3.4).

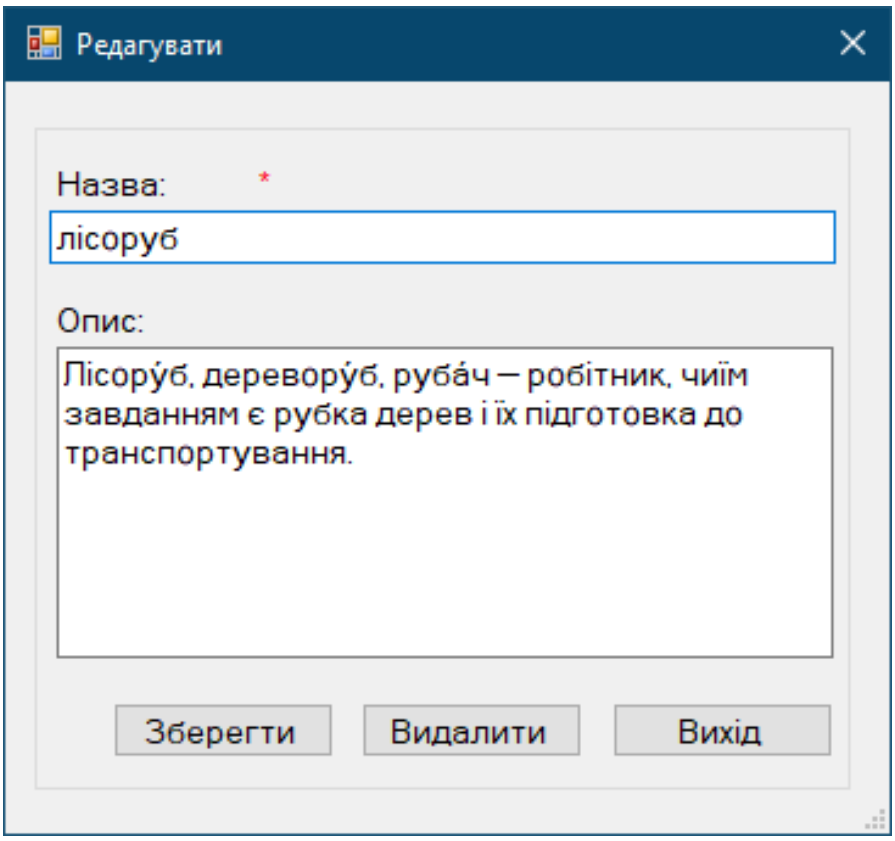


Рис. 3.4 – Вікно для редагування посади.

3) Інтерфейс вікна для виводу каталогу всіх робіт (Рис 3.5).

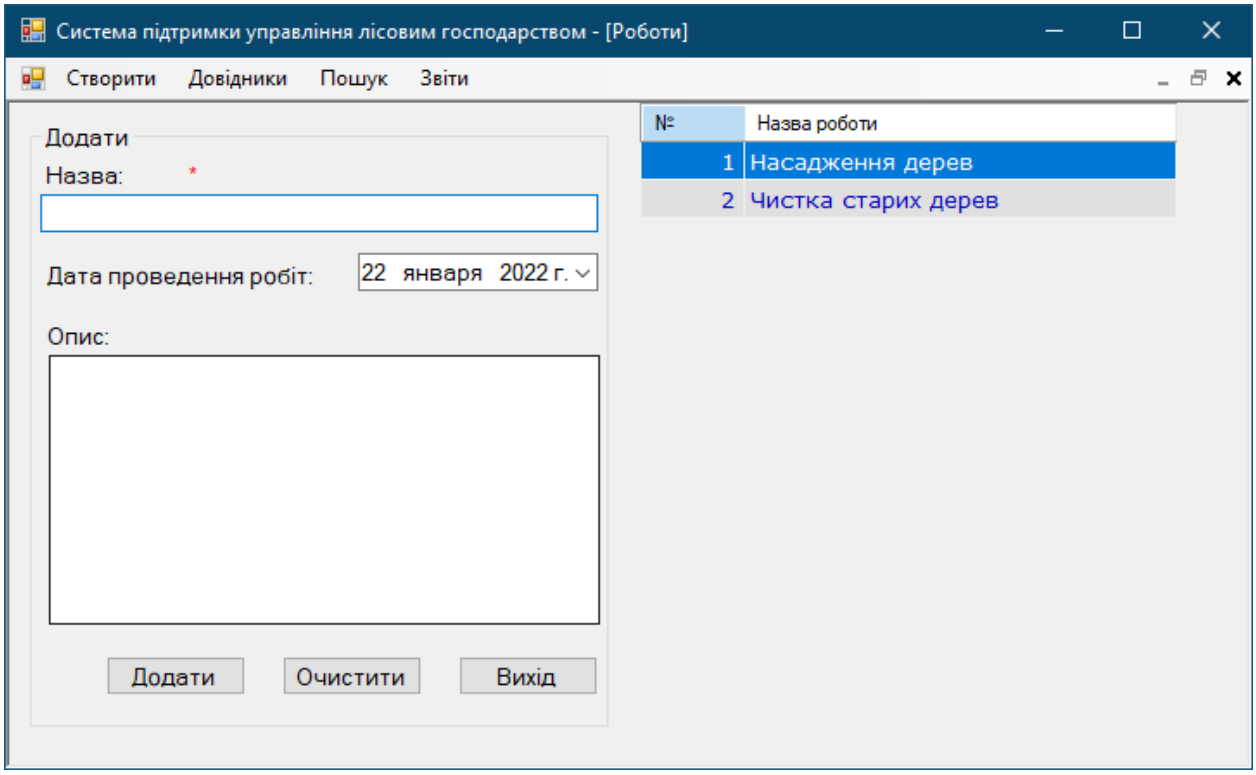


Рис. 3.5 – Вікно для роботи з довідником «Роботи».

Дане вікно призначене для ведення інформації про всі роботи лісного господарства. Тут можна побачити каталог вже існуючих записів про роботи, та додати інформацію про нові роботи. Для того, щоб додати інформацію про нову роботу, необхідно в лівій частині вікна заповнити текстові поля даними, тобто вказати наступні дані:

- ✓ Назву роботи;
- ✓ Дату проведення роботи;
- ✓ Опис посади (не обов’язкове поле).

4) Інтерфейс вікна для редагування даних вибраної роботи (Рис 3.6).

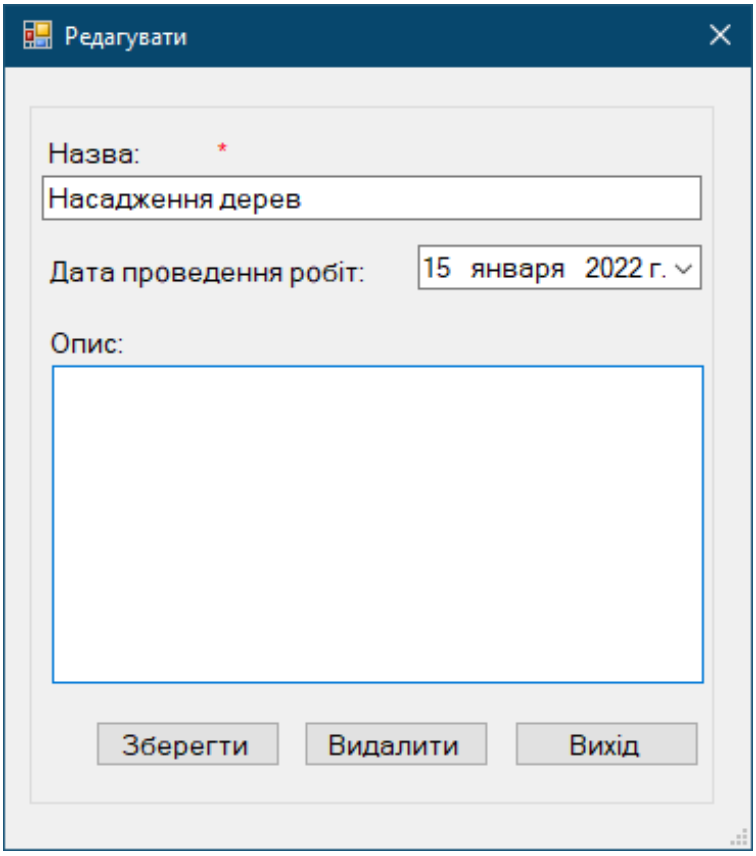


Рис. 3.6 – Вікно для редагування роботи.

5) Інтерфейс вікна для виводу каталогу всіх працівників (Рис 3.7).

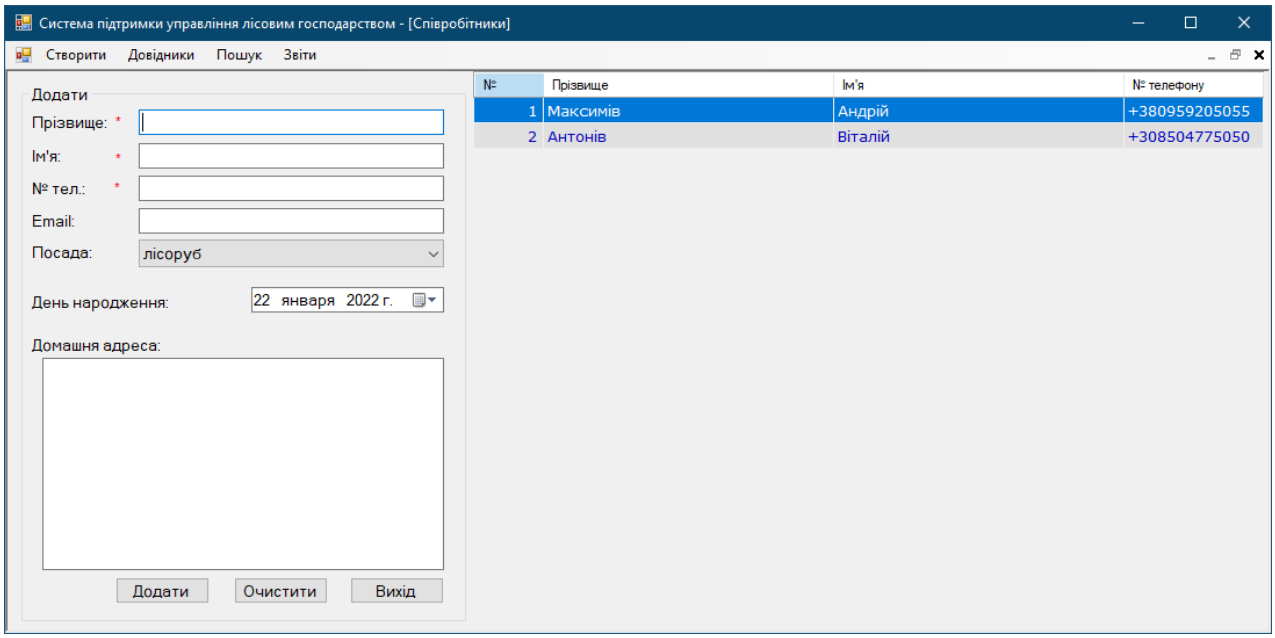


Рис. 3.7 – Вікно для роботи з довідником «Працівники».

Дане вікно призначене для ведення інформації про всіх працівників лісного господарства. Тут можна побачити каталог вже існуючих записів про працівників, та додати інформацію про нових працівників. Для того, щоб додати інформацію про нового працівника, необхідно в лівій частині вікна заповнити текстові поля даними, тобто вказати наступні дані:

- ✓ Прізвище;
- ✓ Ім’я;
- ✓ Номер телефону;
- ✓ Електронну адресу;
- ✓ Посаду;
- ✓ День народження;
- ✓ Домашню адресу..

6) Інтерфейс вікна для редагування даних вибраної роботи (Рис 3.8).

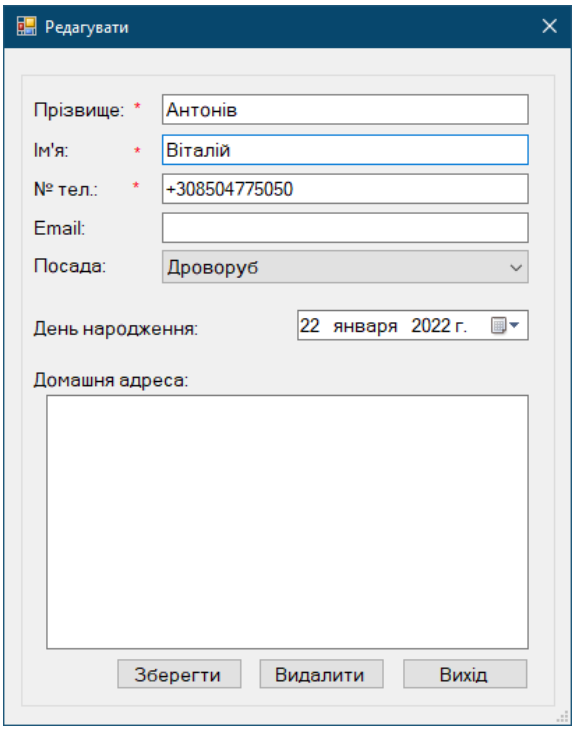


Рис. 3.8 – Вікно для редагування інформації про працівника.

7) Інтерфейс вікна для виводу каталогу всіх угідь (Рис 3.9).

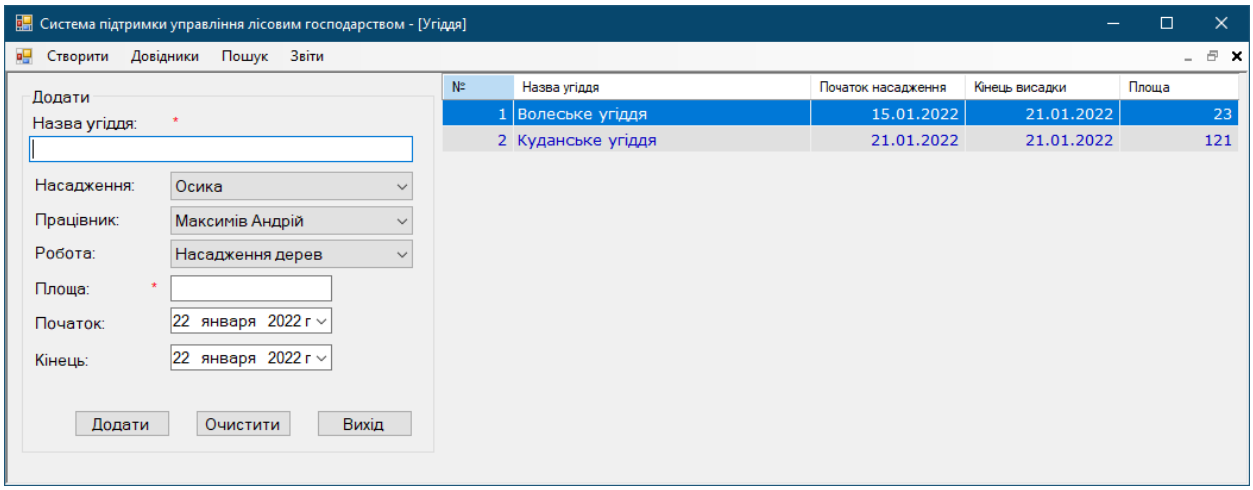


Рис. 3.9 – Інтерфейс вікна «Угіддя».

Дане вікно призначене для ведення інформації про всі угіддя лісного господарства. Тут можна побачити каталог вже існуючих записів про угіддя, та додати інформацію про нові угіддя. Для того, щоб додати інформацію про нове угіддя, необхідно в лівій частині вікна заповнити текстові поля даними, тобто вказати наступні дані:

- ✓ Назву угіддя;

- ✓ Насадження;
- ✓ Працівник;
- ✓ Робота;
- ✓ Площа;
- ✓ Початок робіт;
- ✓ Кінець робіт.

8) Інтерфейс вікна для редагування даних вибраного угіддя (Рис 3.10).

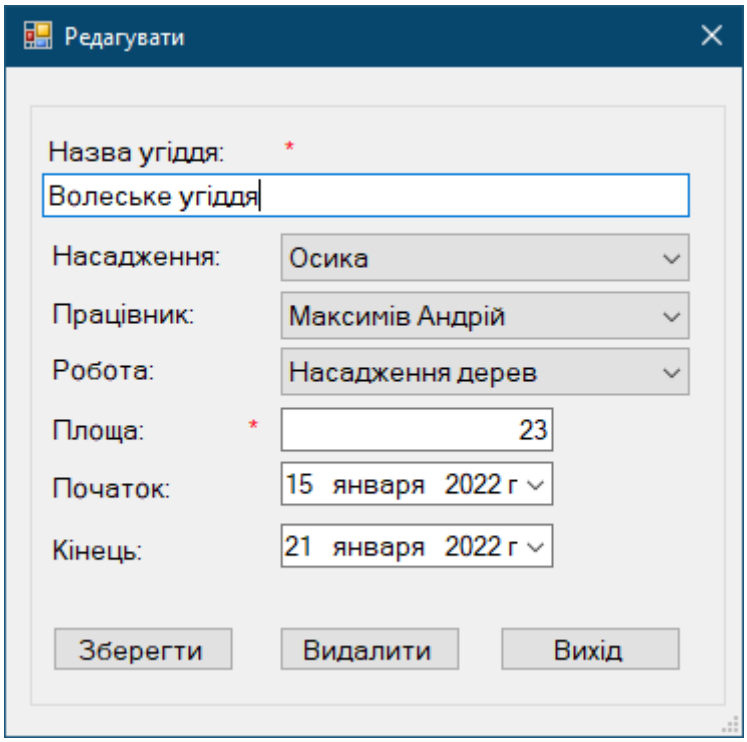


Рис. 3.10 – Вікно для редагування інформації про угіддя.

9) Інтерфейс вікна для виводу каталогу всіх насаджень (Рис 3.11).

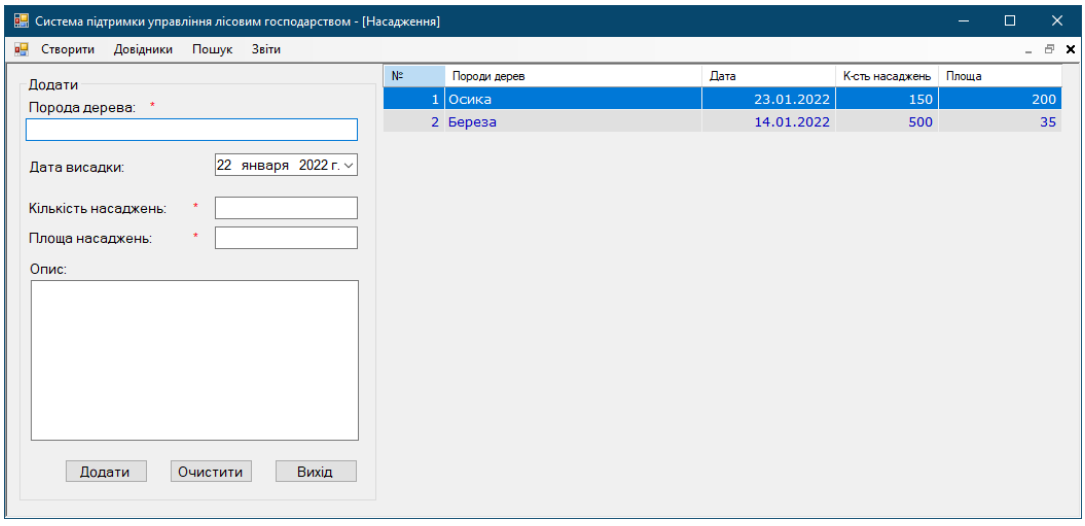


Рис. 3.11 – Інтерфейс вікна «Насадження».

Дане вікно призначене для ведення інформації про всі насадження лісного господарства. Тут можна побачити каталог вже існуючих записів про насадження, та додати інформацію про нові насадження. Для того, щоб додати інформацію про нове насадження, необхідно в лівій частині вікна заповнити текстові поля даними, тобто вказати наступні дані:

- ✓ Породу дерева;
- ✓ Дата висадки;
- ✓ Кількість насаджень;
- ✓ Площа насадження;
- ✓ Опис.

10) Інтерфейс вікна для редагування даних вибраного насадження (Рис 3.12).

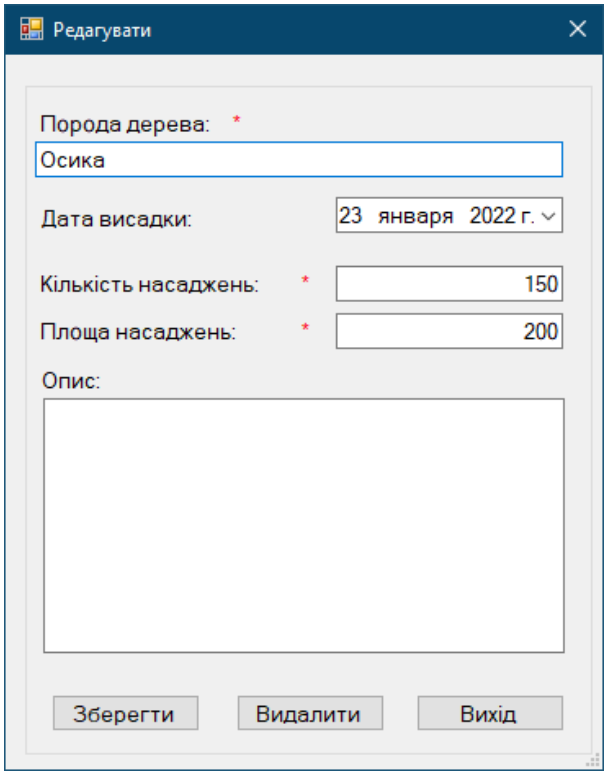


Рис. 3.12 – Вікно для редагування інформації про насадження.

11) Інтерфейс вікна для пошуку працівників по прізвищу або імені працівника (Рис 3.13). Також пошук можна здійснювати по частині прізвища або імені, не вказуючи його повністю. Буде виведено список всіх співпадінь.

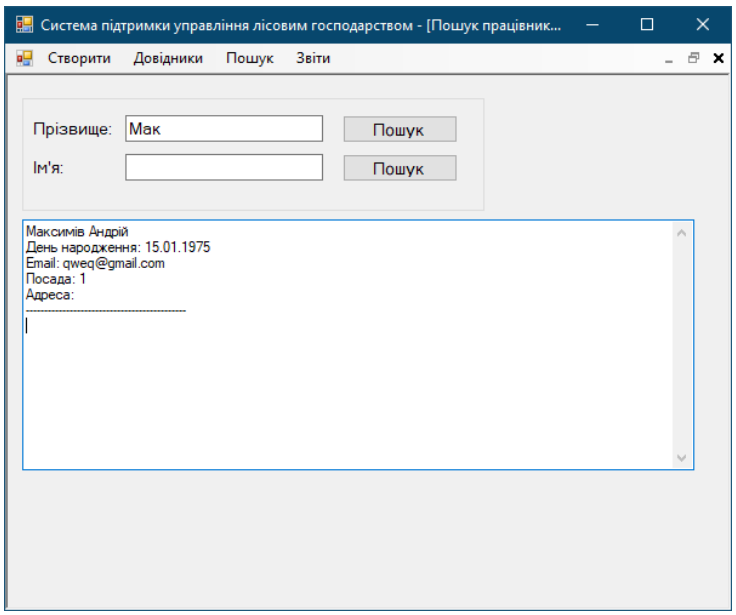


Рис. 3.13 – Вікно для пошуку інформації про працівників.

12) Інтерфейс вікна для пошуку угідь по назві угіддя або площі угіддя (Рис 3.14). Буде виведено список всіх співпадінь.

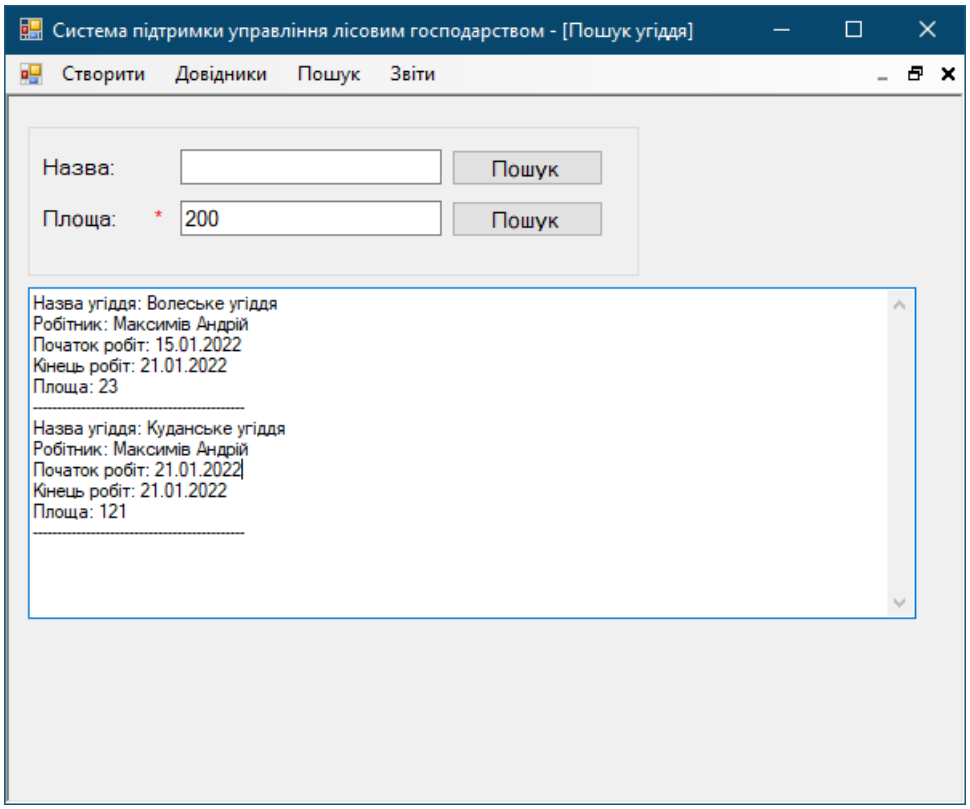


Рис. 3.14 – Вікно для пошуку інформації про угіддя.

13) Інтерфейс вікна для пошуку насаджень по породі дерева насадження або площі насаджень (Рис 3.15). Буде виведено список всіх співпадінь.

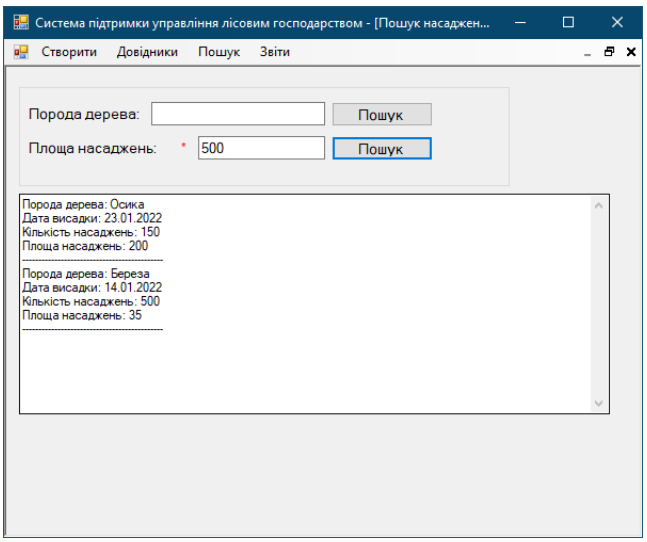


Рис. 3.15 – Вікно для пошуку інформації про насадження.

14) Інтерфейс вікна для формування звітності по вибраній роботі (Рис 3.16). Для цього необхідно вибрати необхідну роботу та натиснути кнопку «Створити».

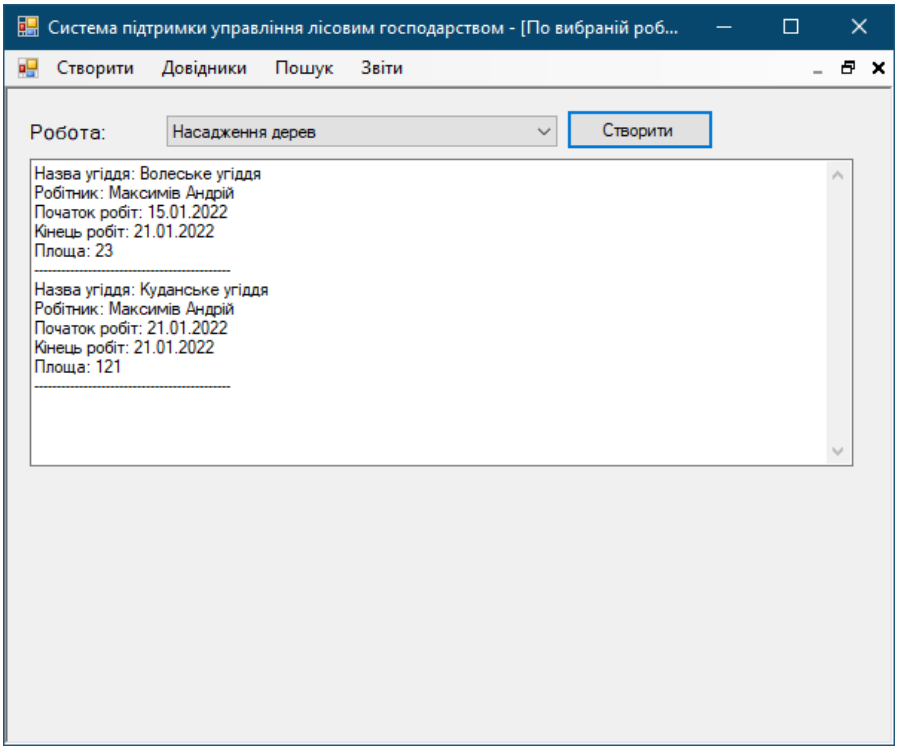


Рис. 3.16 – Вікно для формування звітності по вибраній роботі.

15) Інтерфейс вікна для формування звітності по вибраній посаді (Рис 3.17). Для цього необхідно вибрати необхідну посаду та натиснути кнопку «Створити». Після чого буде виведено список всіх працівників, що займають дану посаду.

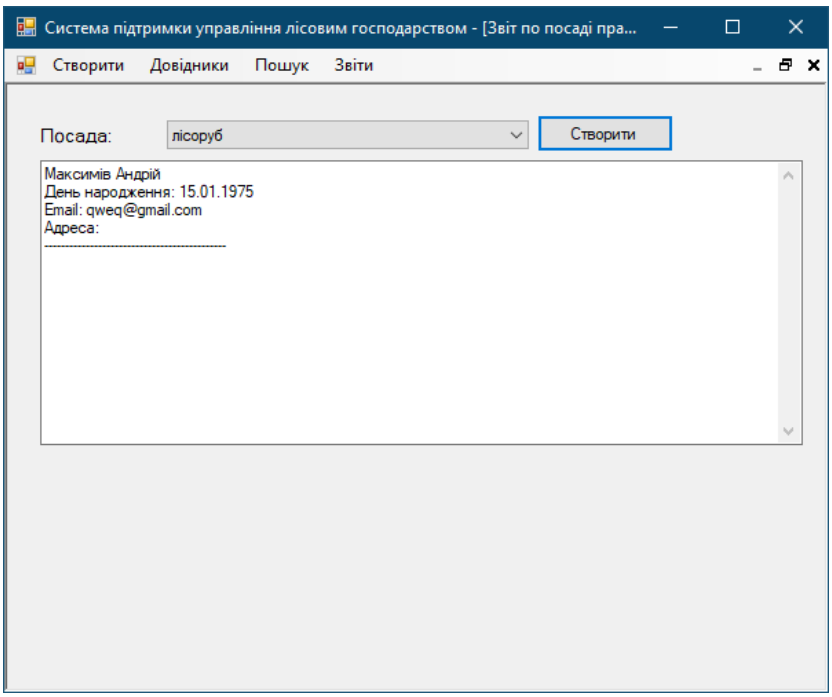


Рис. 3.17 – Вікно для формування звітності по вибраній посаді.

# ВИСНОВКИ

Мета роботи полягала у дослідженні та розробці інформаційно-аналітичної системи підтримки управління підприємством лісового господарства, яка допоможе працівникам лісничого відділу ефективніше працювати, тратити менше часу на виконання завдання та оформлення звітів..

Було проведено дослідження предметної галузі та проаналізовано процес організації роботи лісового господарства. У процесі вивчення документації, спостереження та опитування співробітників, були виділені особливості поточного стану автоматизації в лісовому господарстві, також описані основні процеси розробки.

На основі цих даних були вироблені рекомендації щодо автоматизації процесу підтримки управління лісовим господарством, визначено цілі та завдання автоматизації, сформовано вимоги до програмного продукту, які враховують особливості опрацювання інформації.

Відповідно до поставлених завдань було розроблено програму на платформі .NET засобами мови програмування C#.

За допомогою розробленого програмного забезпечення автоматизовані:

• ведення бази даних працівників;

• ведення бази посад;

• ведення бази даних робіт лісного господарства;

• опрацювання інформації по насадженнях;

• опрацювання інформації по угіддях;

• здійснювання пошуку по працівниках, насадженнях та угіддях;

• формування звітів по посадах працівників та по проведених роботах.

Експлуатація показала, що розроблений програмний продукт досить ефективний та має зручний інтерфейс, а також дозволяє автоматизувати основні бізнес процеси.

Таким чином, можна зробити висновок про те, що всі завдання виконані, мета досягнута.

# СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ


1. Маклаков С.В. опис BPWin і ERWin. CASE-средства разработки информационных систем. — М.: Диалог-Мифи, 1999.
2. Структуроване моделювання ІС: Метод.та вказівки до викон. курсової роботи для студ. напряму 6.050101 «Комп'ютерні науки» денної та заочної форм навчання / уклад. О.М. М`якшило, О.В. Харкянен – К.: НУХТ, 2010.– 14 с.
3. Прометей. Електронний ресурс: https://www.prometeus.ru/
4. ATutor. Електронний ресурс: http://www.atutor.ca
5. DOCEBO. Електронний ресурс: https://www.docebo.com
6. Конспект лекцій з дисципліни «Автоматизоване проектування комп'ютерних систем».
7. Закони України «Про охорону праці». – К., 2002 – 46с
8. Закон України «Про пожежну безпеку». – К., 1993- 22с
9. ДБН В.2.5.-28-2006. Державні будівельні норми україни. Штучне і природне освітлення.-К.:Мінбуд. України, 2006.-76с
10. Правила пожежної безпеки в Україні (НАПБ А.01.001-95). – К.: Основа, 2002.-176с.
11. Бази даних. Теоретичні основи. Моделювання. Реалізація. Навчальний посібник для студ. спец. 7.080401 «Інформаційні управляючі системи та технології» та інших споріднених спеціальностей / уклад.: О.М. М'якшило, Л.Г. Загоровська – К.: НУХТ, 2006 – 168 с.
12. М'якшило О. М. Моделювання баз даних засобами CASE-технології ERWin: Конспект лекцій з дисципліни «Структурне моделювання систем» для студ. спец. 6.080400 напряму «Комп'ютерні науки» всіх форм навчання. – К.: НУХТ, 2008. – 60с.

13. Методичні вказівки до виконання кваліфікаційної магістерської роботи для студентів за напрямом підготовки 6.050101 «Комп'ютерні науки» денної та заочної форм навчання / уклад.: В.В. Самсонов, Л.Ю. Маноха, Т.М. Горлова, Л.Г. Загоровська, О.М. М`якшило, О.А Хлобистова. – К.: НУХТ, 2011. – 15с.
14. Береза А. М. Основи створення інформаційних систем /Навч. посіб. – К.: КНЕЧ, 1998. – 140 с.
15. Робочі програми експлуатаційної та переддипломної практик студ. 5 курсу спец. 7.080401 "Інформаційні управляючі системи та технології" /уклад. С.І. Сіренко, Л.Г. Загоровська. – К.: УДУХТ, 2002. – 16 с.
16. Викрам Васвани. Разработка веб-приложений на PHP. – Питер, 2012
17. Баричев С.Г, Серов Р.Е. Основы современной криптографии: учебное пособие. – М.: Горячая линия – Телеком, 2002
18. Самборский В. І. Анализ хозяйственной деятельности в бюджетных и научных учреждения. — М.: Финансы и статистика, 1989. — 375 с.
19. Линн Бейли, Майкл Моррисон. Изучаем PHP и MYSQL. – Эксмо, 2010
20. Дубейковский В.И. Эффективное моделирование с AllFusion Process Modeler. – Диалог-МИФИ, 2007. – 384с.
21. Міжнародне законодавство про охорону праці: у 3-х т. — К.: Основа, 1997.
22. Конспект лекцій з дисципліни "Теорія прийняття рішень".
23. Навчальна інформація по BPwin: itteach.ru/bpwin.
24. Базы данных. Модели и языки, С. Д. Кузнецов; Бином-Пресс; 2008 г.
25. Купчик М. П., Гандзюк М. П., Степанець І. Ф., Вендичанський В. Н., Литвиненко А. М., Іваненко О. В. Основи охорони праці. – К.: Основа, 2000. – 416 с.
26. Береза А. М. Основи створення інформаційних систем: Навч. посібник. – К.: КНЕУ, 1999. – 140 с.
27. Гаркавенко С. С. Маркетинг: Підручник для вузів. – К.: Лібра, 1998. -384 с.
28. https://uk.wikipedia.org/wiki/Електронний_освітній_ресурс

# ДОДАТКИ

## Додаток А. Лістинги програми

Лістинг 1. Клас «LandForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Forestry.Forms.Create {
  public partial class LandForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    private LandProvider _LandProvider = new LandProvider();
    private List<Land> _LandList = new List<Land>();
    private WorkerProvider _WorkerProvider = new WorkerProvider();
    private List<Worker> _WorkerList = new List<Worker>();
    private PlantingsProvider _PlantingsProvider = new PlantingsProvider();
    private List<Plantings> _PlantingsList = new List<Plantings>();
    private JobsProvider _JobsProvider = new JobsProvider();
    private List<Jobs> _JobsList = new List<Jobs>();

    public LandForm() {
      InitializeComponent();
      LoadAllDate();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _LandProvider.InsertLand(LandNameTBox.Text, double.Parse(SquareTBox.Text,
CultureInfo.InvariantCulture), Convert.ToInt32(PlantingsCBox.SelectedValue),
          Convert.ToInt32(WorkerCBox.SelectedValue),
Convert.ToInt32(JobsCBox.SelectedValue), StartDateDTP.Value, EndDateDTP.Value);
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
```

```
  this.Close();
}

private void DataLoad() {
  int firstRowIndex = 0;
  if (LandGridView.FirstDisplayedScrollingRowIndex > 0) {
    firstRowIndex = LandGridView.FirstDisplayedScrollingRowIndex;
  }
  try {
    _LandList = _LandProvider.GetAllLand();
    LoadDataInLandGridView(_LandList);
    if (_selectedRowIndex == LandGridView.Rows.Count) {
      _selectedRowIndex = LandGridView.Rows.Count - 1;
    }
    if (_selectedRowIndex >= 0) {
      LandGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
      LandGridView.Rows[_selectedRowIndex].Selected = true;
    }
  } catch { }
}

private void LoadAllDate() {
  _WorkerList = _WorkerProvider.GetAllWorker();
  WorkerCBox.DataSource = _WorkerList;
  WorkerCBox.ValueMember = "WorkerId";
  WorkerCBox.DisplayMember = "FIO";

  _PlantingsList = _PlantingsProvider.GetAllPlantings();
  PlantingsCBox.DataSource = _PlantingsList;
  PlantingsCBox.ValueMember = "PlantingsId";
  PlantingsCBox.DisplayMember = "TreeSpecies";

  _JobsList = _JobsProvider.GetAllJobs();
  JobsCBox.DataSource = _JobsList;
  JobsCBox.ValueMember = "JobsId";
  JobsCBox.DisplayMember = "JobsName";

}

private void LoadDataInLandGridView(List<Land> LandList) {
  LandGridView.DataSource = null;
  LandGridView.Columns.Clear();
  LandGridView.AutoGenerateColumns = false;
  LandGridView.RowHeadersVisible = false;

  LandGridView.DataSource = LandList;

  if (LandList.Count > 0) {
    if (LandList[0].Message == NamesMy.NoDataNames.NoDataInLand) {
      DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
      messageColumn.DataPropertyName = "Message";
      messageColumn.Width = LandGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
```

```
        LandGridView.Columns.Add(messageColumn);
       } else {
        DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
        DetailIdColumn.DataPropertyName = "LandId";
        LandGridView.Columns.Add(DetailIdColumn);
        LandGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
        numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        LandGridView.Columns.Add(numberColumn);

        DataGridViewColumn LandNameColumn = new DataGridViewTextBoxColumn();
        LandNameColumn.HeaderText = "Назва угіддя";
        LandNameColumn.DataPropertyName = "LandName";
        LandNameColumn.Width = NamesMy.SizeOptins.NameSize;
        LandGridView.Columns.Add(LandNameColumn);

        DataGridViewColumn LandingDateColumn = new DataGridViewTextBoxColumn();
        LandingDateColumn.HeaderText = "Початок насадження";
        LandingDateColumn.DataPropertyName = "StartDate";
        LandingDateColumn.DefaultCellStyle.Format = "dd/MM/yyyy";
        LandingDateColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        LandingDateColumn.Width = NamesMy.SizeOptins.Date;
        LandGridView.Columns.Add(LandingDateColumn);

        DataGridViewColumn EndDateColumn = new DataGridViewTextBoxColumn();
        EndDateColumn.HeaderText = "Кінець висадки";
        EndDateColumn.DataPropertyName = "EndDate";
        EndDateColumn.DefaultCellStyle.Format = "dd/MM/yyyy";
        EndDateColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        EndDateColumn.Width = NamesMy.SizeOptins.Date;
        LandGridView.Columns.Add(EndDateColumn);

        DataGridViewColumn NumberOfLandColumn = new DataGridViewTextBoxColumn();
        NumberOfLandColumn.HeaderText = "Площа";
        NumberOfLandColumn.DataPropertyName = "Square";
        NumberOfLandColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        NumberOfLandColumn.Width = 100;
        LandGridView.Columns.Add(NumberOfLandColumn);

       }
       for (int i = 0; i < LandGridView.Columns.Count; i++) {
        LandGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
       }
      }
```

```
    }

    private void ClearAllControls() {
      LandNameTBox.Text = String.Empty;
      SquareTBox.Text = String.Empty;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(LandNameTBox.Text)) {
        LandNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        LandNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataConvertToDouble(SquareTBox.Text)) {
        SquareValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        SquareValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

    private void LandGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
      if (e.RowIndex >= 0 && LandGridView[0, e.RowIndex].Value.ToString() !=
_PlantingsList[0].Message) {
        _selectedRowIndex = e.RowIndex;
        UpdateLandForm updateLandForm = new
UpdateLandForm(Convert.ToInt32(LandGridView[0, e.RowIndex].Value.ToString()));
        updateLandForm.ShowDialog();
        DataLoad();
      }
    }
  }
}
```

Лістинг 2. Клас «PlantingsForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Forestry.Forms.Create {
```

```
  public partial class PlantingsForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    private PlantingsProvider _PlantingsProvider = new PlantingsProvider();
    private List<Plantings> _PlantingsList = new List<Plantings>();

    public PlantingsForm() {
      InitializeComponent();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _PlantingsProvider.InsertPlantings(TreeSpeciesTBox.Text, DescriptionTBox.Text,
LandingDateDTP.Value,
         Convert.ToInt32(NumberOfPlantingsTBox.Text),
Convert.ToDouble(PlantingAreaTBox.Text ));
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void DataLoad() {
      int firstRowIndex = 0;
      if (PlantingsGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = PlantingsGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _PlantingsList = _PlantingsProvider.GetAllPlantings();
        LoadDataInPlantingsGridView(_PlantingsList);
        if (_selectedRowIndex == PlantingsGridView.Rows.Count) {
          _selectedRowIndex = PlantingsGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          PlantingsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          PlantingsGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
    }

    private void LoadDataInPlantingsGridView(List<Plantings> PlantingsList) {
      PlantingsGridView.DataSource = null;
      PlantingsGridView.Columns.Clear();
      PlantingsGridView.AutoGenerateColumns = false;
```

```
    PlantingsGridView.RowHeadersVisible = false;

    PlantingsGridView.DataSource = PlantingsList;

    if (PlantingsList.Count > 0) {
      if (PlantingsList[0].Message == NamesMy.NoDataNames.NoDataInPlantings) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = PlantingsGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
        PlantingsGridView.Columns.Add(messageColumn);
      } else {
        DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
        DetailIdColumn.DataPropertyName = "PlantingsId";
        PlantingsGridView.Columns.Add(DetailIdColumn);
        PlantingsGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
        numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        PlantingsGridView.Columns.Add(numberColumn);

        DataGridViewColumn PlantingsNameColumn = new DataGridViewTextBoxColumn();
        PlantingsNameColumn.HeaderText = "Породи дерев";
        PlantingsNameColumn.DataPropertyName = "TreeSpecies";
        PlantingsNameColumn.Width = NamesMy.SizeOptins.NameSize;
        PlantingsGridView.Columns.Add(PlantingsNameColumn);

        DataGridViewColumn LandingDateColumn = new DataGridViewTextBoxColumn();
        LandingDateColumn.HeaderText = "Дата";
        LandingDateColumn.DataPropertyName = "LandingDate";
        LandingDateColumn.DefaultCellStyle.Format = "dd/MM/yyyy";
        LandingDateColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        LandingDateColumn.Width = NamesMy.SizeOptins.Date;
        PlantingsGridView.Columns.Add(LandingDateColumn);

        DataGridViewColumn NumberOfPlantingsColumn = new
DataGridViewTextBoxColumn();
        NumberOfPlantingsColumn.HeaderText = "К-сть насаджень";
        NumberOfPlantingsColumn.DataPropertyName = "NumberOfPlantings";
        NumberOfPlantingsColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        NumberOfPlantingsColumn.Width = 100;
        PlantingsGridView.Columns.Add(NumberOfPlantingsColumn);

        DataGridViewColumn PlantingAreaColumn = new DataGridViewTextBoxColumn();
        PlantingAreaColumn.HeaderText = "Площа";
        PlantingAreaColumn.DataPropertyName = "PlantingArea";
```

```
        PlantingAreaColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        PlantingAreaColumn.Width = 120;
        PlantingsGridView.Columns.Add(PlantingAreaColumn);

      }
      for (int i = 0; i < PlantingsGridView.Columns.Count; i++) {
        PlantingsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }

  private void ClearAllControls() {
    TreeSpeciesTBox.Text = String.Empty;
    DescriptionTBox.Text = String.Empty;
  }

  private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (_validation.IsDataEntering(TreeSpeciesTBox.Text)) {
      TreeSpeciesValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      TreeSpeciesValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_validation.IsDataConvertToInt(NumberOfPlantingsTBox.Text)) {
      NumberOfPlantingsValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      NumberOfPlantingsValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_validation.IsDataConvertToInt(PlantingAreaTBox.Text)) {
      PlantingAreaValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      PlantingAreaValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    return isCorrect;
  }

  private void PlantingsGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
    if (e.RowIndex >= 0 && PlantingsGridView[0, e.RowIndex].Value.ToString() !=
_PlantingsList[0].Message) {
      _selectedRowIndex = e.RowIndex;
      UpdatePlantingsForm updatePlantingsForm = new
UpdatePlantingsForm(Convert.ToInt32(PlantingsGridView[0, e.RowIndex].Value.ToString()));
      updatePlantingsForm.ShowDialog();
      DataLoad();
    }
  }
 }
}
```

Лістинг 3. Клас «UpdateLandForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Forestry.Forms.Create {
  public partial class UpdateLandForm : Form {
    private int _LandId = 0;
    private Land _selectedLand = new Land();
    private LandProvider _LandProvider = new LandProvider();
    private ValidationMy _validation = new ValidationMy();
    private WorkerProvider _WorkerProvider = new WorkerProvider();
    private List<Worker> _WorkerList = new List<Worker>();
    private PlantingsProvider _PlantingsProvider = new PlantingsProvider();
    private List<Plantings> _PlantingsList = new List<Plantings>();
    private JobsProvider _JobsProvider = new JobsProvider();
    private List<Jobs> _JobsList = new List<Jobs>();


    public UpdateLandForm(int LandId) {
      InitializeComponent();
      _LandId = LandId;
      LoadAllDate();
    }

    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _LandProvider.UpdateLand(LandNameTBox.Text, double.Parse(SquareTBox.Text,
CultureInfo.InvariantCulture), Convert.ToInt32(PlantingsCBox.SelectedValue),
          Convert.ToInt32(WorkerCBox.SelectedValue),
Convert.ToInt32(JobsCBox.SelectedValue), StartDateDTP.Value, EndDateDTP.Value,
_LandId);
        this.Close();
      }
    }

    private void DeleteBtn_Click(object sender, EventArgs e) {
      if (MessageBox.Show("Ви дійсно хочете видалити цей елемент?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
        _LandProvider.DeleteLandByLandId(_LandId);
        this.Close();
```

```
    }
  }

  private void ExitBtn_Click(object sender, EventArgs e) {
    this.Close();
  }

  private void LoadAllDate() {
    _WorkerList = _WorkerProvider.GetAllWorker();
    WorkerCBox.DataSource = _WorkerList;
    WorkerCBox.ValueMember = "WorkerId";
    WorkerCBox.DisplayMember = "FIO";

    _PlantingsList = _PlantingsProvider.GetAllPlantings();
    PlantingsCBox.DataSource = _PlantingsList;
    PlantingsCBox.ValueMember = "PlantingsId";
    PlantingsCBox.DisplayMember = "TreeSpecies";

    _JobsList = _JobsProvider.GetAllJobs();
    JobsCBox.DataSource = _JobsList;
    JobsCBox.ValueMember = "JobsId";
    JobsCBox.DisplayMember = "JobsName";

    _selectedLand = _LandProvider.SelectedLandByLandId(_LandId);
    LandNameTBox.Text = _selectedLand.LandName;
    PlantingsCBox.SelectedValue = _selectedLand.PlantingsId;
    WorkerCBox.SelectedValue = _selectedLand.WorkerId;
    JobsCBox.SelectedValue = _selectedLand.JobsId;
    SquareTBox.Text = _selectedLand.Square.ToString();
    StartDateDTP.Value = _selectedLand.StartDate;
    EndDateDTP.Value = _selectedLand.EndDate;
  }

  private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (_validation.IsDataEntering(LandNameTBox.Text)) {
      LandNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      LandNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_validation.IsDataConvertToDouble(SquareTBox.Text)) {
      SquareValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      SquareValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    return isCorrect;
  }

 }
}
```

Лістинг 4. Клас «UpdatePlantingsForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Forestry.Forms.Create {
  public partial class UpdatePlantingsForm : Form {
    private int _PlantingsId = 0;
    private Plantings _selectedPlantings = new Plantings();
    private PlantingsProvider _PlantingsProvider = new PlantingsProvider();
    private ValidationMy _validation = new ValidationMy();

    public UpdatePlantingsForm(int PlantingsId) {
      InitializeComponent();
      _PlantingsId = PlantingsId;
      LoadAllDate();
    }

    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _PlantingsProvider.UpdatePlantings(TreeSpeciesTBox.Text, DescriptionTBox.Text,
LandingDateDTP.Value,
         Convert.ToInt32(NumberOfPlantingsTBox.Text),
Convert.ToDouble(PlantingAreaTBox.Text), _PlantingsId);
        this.Close();
      }
    }

    private void DeleteBtn_Click(object sender, EventArgs e) {
      if (MessageBox.Show("Ви дійсно хочете видалити цей елемент?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
        _PlantingsProvider.DeletePlantingsByPlantingsId(_PlantingsId);
        this.Close();
      }
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void LoadAllDate() {
      _selectedPlantings = _PlantingsProvider.SelectedPlantingsByPlantingsId(_PlantingsId);
```

```
      TreeSpeciesTBox.Text = _selectedPlantings.TreeSpecies;
      LandingDateDTP.Value = _selectedPlantings.LandingDate;
      NumberOfPlantingsTBox.Text = _selectedPlantings.NumberOfPlantings.ToString();
      PlantingAreaTBox.Text = _selectedPlantings.PlantingArea.ToString();
      DescriptionTBox.Text = _selectedPlantings.Description;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(TreeSpeciesTBox.Text)) {
        TreeSpeciesValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        TreeSpeciesValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataConvertToInt(NumberOfPlantingsTBox.Text)) {
        NumberOfPlantingsValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        NumberOfPlantingsValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataConvertToInt(PlantingAreaTBox.Text)) {
        PlantingAreaValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        PlantingAreaValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

  }
}
```

Лістинг 5. Клас «JobsForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Forestry.Forms {
  public partial class JobsForm : Form {
    private int _selectedRowIndex = 0;
```

```
    private ValidationMy _validation = new ValidationMy();
    private JobsProvider _JobsProvider = new JobsProvider();
    private List<Jobs> _JobsList = new List<Jobs>();
    public JobsForm() {
      InitializeComponent();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _JobsProvider.InsertJobs(JobsNameTBox.Text, DescriptionTBox.Text,
JobsDateDTP.Value);
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void DataLoad() {
      int firstRowIndex = 0;
      if (JobsGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = JobsGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _JobsList = _JobsProvider.GetAllJobs();
        LoadDataInJobsGridView(_JobsList);
        if (_selectedRowIndex == JobsGridView.Rows.Count) {
          _selectedRowIndex = JobsGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          JobsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          JobsGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
    }

    private void LoadDataInJobsGridView(List<Jobs> JobsList) {
      JobsGridView.DataSource = null;
      JobsGridView.Columns.Clear();
      JobsGridView.AutoGenerateColumns = false;
      JobsGridView.RowHeadersVisible = false;

      JobsGridView.DataSource = JobsList;

      if (JobsList.Count > 0) {
```

```
        if (JobsList[0].Message == NamesMy.NoDataNames.NoDataInJobs) {
          DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
          messageColumn.DataPropertyName = "Message";
          messageColumn.Width = JobsGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
          JobsGridView.Columns.Add(messageColumn);
        } else {
          DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
          DetailIdColumn.DataPropertyName = "JobsId";
          JobsGridView.Columns.Add(DetailIdColumn);
          JobsGridView.Columns[0].Visible = false;

          DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
          numberColumn.HeaderText = "№ ";
          numberColumn.DataPropertyName = "Number";
          numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
          numberColumn.Width = NamesMy.SizeOptins.NumberSize;
          JobsGridView.Columns.Add(numberColumn);

          DataGridViewColumn JobsNameColumn = new DataGridViewTextBoxColumn();
          JobsNameColumn.HeaderText = "Назва роботи";
          JobsNameColumn.DataPropertyName = "JobsName";
          JobsNameColumn.Width = NamesMy.SizeOptins.NameSize;
          JobsGridView.Columns.Add(JobsNameColumn);



        }
        for (int i = 0; i < JobsGridView.Columns.Count; i++) {
          JobsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
        }
      }
    }

    private void ClearAllControls() {
      JobsNameTBox.Text = String.Empty;
      DescriptionTBox.Text = String.Empty;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(JobsNameTBox.Text)) {
        JobsNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        JobsNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

    private void JobsGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
      if (e.RowIndex >= 0 && JobsGridView[0, e.RowIndex].Value.ToString() !=
_JobsList[0].Message) {
```

```
        _selectedRowIndex = e.RowIndex;
        UpdateJobsForm updateJobsForm = new
UpdateJobsForm(Convert.ToInt32(JobsGridView[0, e.RowIndex].Value.ToString()));
        updateJobsForm.ShowDialog();
        DataLoad();
      }
    }
  }
}
```

Лістинг 6. Клас «PositionForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Forestry.Forms {
  public partial class PositionForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    private PositionProvider _PositionProvider = new PositionProvider();
    private List<Position> _PositionList = new List<Position>();
    public PositionForm() {
      InitializeComponent();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _PositionProvider.InsertPosition(JobsNameTBox.Text, DescriptionTBox.Text);
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();

    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }
```

```
  private void DataLoad() {
    int firstRowIndex = 0;
    if (PositionGridView.FirstDisplayedScrollingRowIndex > 0) {
      firstRowIndex = PositionGridView.FirstDisplayedScrollingRowIndex;
    }
    try {
      _PositionList = _PositionProvider.GetAllPosition();
      LoadDataInPositionGridView(_PositionList);
      if (_selectedRowIndex == PositionGridView.Rows.Count) {
        _selectedRowIndex = PositionGridView.Rows.Count - 1;
      }
      if (_selectedRowIndex >= 0) {
        PositionGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
        PositionGridView.Rows[_selectedRowIndex].Selected = true;
      }
    } catch { }
  }

  private void LoadDataInPositionGridView(List<Position> PositionList) {
    PositionGridView.DataSource = null;
    PositionGridView.Columns.Clear();
    PositionGridView.AutoGenerateColumns = false;
    PositionGridView.RowHeadersVisible = false;

    PositionGridView.DataSource = PositionList;

    if (PositionList.Count > 0) {
      if (PositionList[0].Message == NamesMy.NoDataNames.NoDataInPosition) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = PositionGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
        PositionGridView.Columns.Add(messageColumn);
      } else {
        DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
        DetailIdColumn.DataPropertyName = "PositionId";
        PositionGridView.Columns.Add(DetailIdColumn);
        PositionGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
        numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        PositionGridView.Columns.Add(numberColumn);

        DataGridViewColumn PositionNameColumn = new DataGridViewTextBoxColumn();
        PositionNameColumn.HeaderText = "Назва посади";
        PositionNameColumn.DataPropertyName = "PositionName";
        PositionNameColumn.Width = NamesMy.SizeOptins.NameSize;
        PositionGridView.Columns.Add(PositionNameColumn);
```

```
        }
        for (int i = 0; i < PositionGridView.Columns.Count; i++) {
          PositionGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
        }
      }
    }

    private void ClearAllControls() {
      JobsNameTBox.Text = String.Empty;
      DescriptionTBox.Text = String.Empty;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(JobsNameTBox.Text)) {
        PositionNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        PositionNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

    private void PositionGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
      if (e.RowIndex >= 0 && PositionGridView[0, e.RowIndex].Value.ToString() !=
_PositionList[0].Message) {
        _selectedRowIndex = e.RowIndex;
        UpdatePositionForm updatePositionForm = new
UpdatePositionForm(Convert.ToInt32(PositionGridView[0, e.RowIndex].Value.ToString()));
        updatePositionForm.ShowDialog();
        DataLoad();
      }
    }
  }
}
```

Лістинг 7. Клас «UpdateJobsForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
namespace Forestry.Forms {
  public partial class UpdateJobsForm : Form {
    private int _JobsId = 0;
    private Jobs _selectedJobs = new Jobs();
    private JobsProvider _JobsProvider = new JobsProvider();
    private ValidationMy _Validation = new ValidationMy();
    public UpdateJobsForm(int JobsId) {
      InitializeComponent();
      _JobsId = JobsId;
      LoadAllDate();
    }

    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _JobsProvider.UpdateJobs(JobsNameTBox.Text, DescriptionTBox.Text,
JobsDateDTP.Value, _JobsId);
        this.Close();
      }
    }

    private void DeleteBtn_Click(object sender, EventArgs e) {
      if (MessageBox.Show("Вы действительно хотите удалить этот элемент?", "Удалить",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
        _JobsProvider.DeleteJobsByJobsId(_JobsId);
        this.Close();
      }
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }


    private void LoadAllDate() {
      _selectedJobs = _JobsProvider.SelectedJobsByJobsId(_JobsId);
      JobsNameTBox.Text = _selectedJobs.JobsName;
      DescriptionTBox.Text = _selectedJobs.Description;
      JobsDateDTP.Value = _selectedJobs.JobsDate;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_Validation.IsDataEntering(JobsNameTBox.Text)) {
        JobsNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        JobsNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

  }
```

}

Лістинг 8. Клас «UpdatePositionForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Forestry.Forms {
  public partial class UpdatePositionForm : Form {
    private int _PositionId = 0;
    private Position _selectedPosition = new Position();
    private PositionProvider _PositionProvider = new PositionProvider();
    private ValidationMy _Validation = new ValidationMy();
    public UpdatePositionForm(int PositionId) {
      InitializeComponent();
      _PositionId = PositionId;
      LoadAllDate();
    }

    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _PositionProvider.UpdatePosition(PositionNameTBox.Text, DescriptionTBox.Text,
_PositionId);
        this.Close();
      }
    }

    private void DeleteBtn_Click(object sender, EventArgs e) {
      if (MessageBox.Show("Вы действительно хотите удалить этот элемент?", "Удалить",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
        _PositionProvider.DeletePositionByPositionId(_PositionId);
        this.Close();
      }
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void LoadAllDate() {
      _selectedPosition = _PositionProvider.SelectedPositionByPositionId(_PositionId);
      PositionNameTBox.Text = _selectedPosition.PositionName;
      DescriptionTBox.Text = _selectedPosition.Description;
```

```
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_Validation.IsDataEntering(PositionNameTBox.Text)) {
        PositionNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        PositionNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

  }
}
```

Лістинг 9. Клас «UpdateWorkerForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Forestry.Forms {
  public partial class UpdateWorkerForm : Form {
    private int _WorkerId = 0;
    private Worker _selectedWorker = new Worker();
    private WorkerProvider _WorkerProvider = new WorkerProvider();
    private ValidationMy _validation = new ValidationMy();
    private PositionProvider _PositionProvider = new PositionProvider();
    private List<Position> _PositionList = new List<Position>();
    public UpdateWorkerForm(int WorkerId) {
      InitializeComponent();
      _WorkerId = WorkerId;
      LoadAllDate();
    }

    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _WorkerProvider.UpdateWorker(LastNameTBox.Text, FirstNameTBox.Text,
PhoneTBox.Text, AddressTBox.Text, EmailTBox.Text, BirthDateDTP.Value,
          Convert.ToInt32(PositionCBox.SelectedValue), _WorkerId);
        this.Close();
      }
    }
```

```
    private void DeleteBtn_Click(object sender, EventArgs e) {
      if (MessageBox.Show("Ви дійсно хочете видалити цей елемент?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
        _WorkerProvider.DeleteWorkerByWorkerId(_WorkerId);
        this.Close();
      }
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void LoadAllDate() {
      _PositionList = _PositionProvider.GetAllPosition();
      PositionCBox.DataSource = _PositionList;
      PositionCBox.ValueMember = "PositionId";
      PositionCBox.DisplayMember = "PositionName";

      _selectedWorker = _WorkerProvider.SelectedWorkerByWorkerId(_WorkerId);
      LastNameTBox.Text = _selectedWorker.LastName;
      FirstNameTBox.Text = _selectedWorker.FirstName;
      PhoneTBox.Text = _selectedWorker.Phone;
      AddressTBox.Text = _selectedWorker.Address;
      BirthDateDTP.Value = _selectedWorker.BirthDate;
      EmailTBox.Text = _selectedWorker.Email;
      PositionCBox.SelectedValue = _selectedWorker.PositionId;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(LastNameTBox.Text)) {
        LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(FirstNameTBox.Text)) {
        FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(PhoneTBox.Text)) {
        PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }
```

```
  }
}
```

Лістинг 10. Клас «WorkerForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Forestry.Forms {
  public partial class WorkerForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    WorkerProvider _WorkerProvider = new WorkerProvider();
    List<Worker> _WorkerList = new List<Worker>();
    private PositionProvider _PositionProvider = new PositionProvider();
    private List<Position> _PositionList = new List<Position>();
    public WorkerForm() {
      InitializeComponent();
      LoadAllDate();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _WorkerProvider.InsertWorker(LastNameTBox.Text, FirstNameTBox.Text,
PhoneTBox.Text, AddressTBox.Text, EmailTBox.Text, BirthDateDTP.Value,
Convert.ToInt32(PositionCBox.SelectedValue));
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void LoadAllDate() {
      _PositionList = _PositionProvider.GetAllPosition();
      PositionCBox.DataSource = _PositionList;
```

```
    PositionCBox.ValueMember = "PositionId";
    PositionCBox.DisplayMember = "PositionName";
   }

    private void DataLoad() {
    int firstRowIndex = 0;
    if (WorkerGridView.FirstDisplayedScrollingRowIndex > 0) {
      firstRowIndex = WorkerGridView.FirstDisplayedScrollingRowIndex;
    }
    try {
      _WorkerList = _WorkerProvider.GetAllWorker();
      LoadDataInWorkerGridView(_WorkerList);
      if (_selectedRowIndex == WorkerGridView.Rows.Count) {
        _selectedRowIndex = WorkerGridView.Rows.Count - 1;
      }
      if (_selectedRowIndex >= 0) {
        WorkerGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
        WorkerGridView.Rows[_selectedRowIndex].Selected = true;
      }
    } catch { }
   }

   private void LoadDataInWorkerGridView(List<Worker> WorkerList) {
    WorkerGridView.DataSource = null;
    WorkerGridView.Columns.Clear();
    WorkerGridView.AutoGenerateColumns = false;
    WorkerGridView.RowHeadersVisible = false;

    WorkerGridView.DataSource = WorkerList;

    if (WorkerList.Count > 0) {
      if (WorkerList[0].Message == NamesMy.NoDataNames.NoDataInWorker) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = WorkerGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
        WorkerGridView.Columns.Add(messageColumn);
      } else {
        DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
        DetailIdColumn.DataPropertyName = "WorkerId";
        WorkerGridView.Columns.Add(DetailIdColumn);
        WorkerGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
        numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        WorkerGridView.Columns.Add(numberColumn);

        DataGridViewColumn LastNameColumn = new DataGridViewTextBoxColumn();
```

```
      LastNameColumn.HeaderText = "Прізвище";
      LastNameColumn.DataPropertyName = "LastName";
      LastNameColumn.Width = NamesMy.SizeOptins.NameSize;
      WorkerGridView.Columns.Add(LastNameColumn);

      DataGridViewColumn FirstNameColumn = new DataGridViewTextBoxColumn();
      FirstNameColumn.HeaderText = "Ім'я";
      FirstNameColumn.DataPropertyName = "FirstName";
      FirstNameColumn.Width = NamesMy.SizeOptins.NameSize;
      WorkerGridView.Columns.Add(FirstNameColumn);

      DataGridViewColumn PhoneColumn = new DataGridViewTextBoxColumn();
      PhoneColumn.HeaderText = "№ телефону";
      PhoneColumn.DataPropertyName = "Phone";
      PhoneColumn.Width = NamesMy.SizeOptins.Phone;
      WorkerGridView.Columns.Add(PhoneColumn);

    }
    for (int i = 0; i < WorkerGridView.Columns.Count; i++) {
      WorkerGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
    }
  }
}

private void ClearAllControls() {
  LastNameTBox.Text = String.Empty;
  FirstNameTBox.Text = String.Empty;
  PhoneTBox.Text = String.Empty;
}

private bool IsDataEnteringCorrect() {
  bool isCorrect = true;
  if (_validation.IsDataEntering(LastNameTBox.Text)) {
    LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(FirstNameTBox.Text)) {
    FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(PhoneTBox.Text)) {
    PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    PhoneValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  return isCorrect;
}
```

```
    private void WorkerGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
      if (e.RowIndex >= 0 && WorkerGridView[0, e.RowIndex].Value.ToString() !=
_WorkerList[0].Message) {
        _selectedRowIndex = e.RowIndex;
        UpdateWorkerForm updateWorkerForm = new
UpdateWorkerForm(Convert.ToInt32(WorkerGridView[0, e.RowIndex].Value.ToString()));
        updateWorkerForm.ShowDialog();
        DataLoad();
      }
    }
  }
}
```

Лістинг 11. Клас «RaportJobsForm»

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

namespace Forestry.Forms.Raport {
  public partial class RaportJobsForm : Form {
    LandProvider _LandProvider = new LandProvider();
    List<Land> _LandList = new List<Land>();
    private JobsProvider _JobsProvider = new JobsProvider();
    private List<Jobs> _JobsList = new List<Jobs>();

    public RaportJobsForm() {
      InitializeComponent();
      LoadAllDate();
    }

    private void CreateBtn_Click(object sender, EventArgs e) {
      _LandList =
_LandProvider.GetAllLandByJobsId(Convert.ToInt32(JobsCBox.SelectedValue));
      GetDataRaport(_LandList);
    }

    private void LoadAllDate() {
      _JobsList = _JobsProvider.GetAllJobs();
      JobsCBox.DataSource = _JobsList;
      JobsCBox.ValueMember = "JobsId";
      JobsCBox.DisplayMember = "JobsName";
    }
```

```
    public void GetDataRaport(List<Land> AllLandList) {
      SearchTBox.Text = "";
      if (AllLandList.Count > 0) {
        for (int i = 0; i < AllLandList.Count; i++) {
          SearchTBox.Text += "Назва угіддя: " + AllLandList[i].LandName;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Робітник: " + AllLandList[i].WorkerFIO;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Початок робіт: " + AllLandList[i].StartDate.ToShortDateString();
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Кінець робіт: " + AllLandList[i].EndDate.ToShortDateString();
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Площа: " + AllLandList[i].Square;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "-----------------------------------------\r\n";
        }
      } else {
        SearchTBox.Text += "За заданими критеріями нічого не знайдено!";
      }
    }

  }
}
```

Лістинг 12. Клас «RaportPositionForm»

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

namespace Forestry.Forms.Raport {
  public partial class RaportPositionForm : Form {
    WorkerProvider _WorkerProvider = new WorkerProvider();
    List<Worker> _WorkerList = new List<Worker>();
    private PositionProvider _PositionProvider = new PositionProvider();
    private List<Position> _PositionList = new List<Position>();
    public RaportPositionForm() {
      InitializeComponent();
      LoadAllDate();
    }

    private void LoadAllDate() {
      _PositionList = _PositionProvider.GetAllPosition();
      PositionCBox.DataSource = _PositionList;
```

```
      PositionCBox.ValueMember = "PositionId";
      PositionCBox.DisplayMember = "PositionName";
    }

    private void CreateBtn_Click(object sender, EventArgs e) {
      _WorkerList =
_WorkerProvider.GetAllWorkerByPositionId(Convert.ToInt32(PositionCBox.SelectedValue));
      GetDataRaport(_WorkerList);
    }

    public void GetDataRaport(List<Worker> AllWorkerList) {
      SearchTBox.Text = "";
      if (AllWorkerList.Count > 0) {
        for (int i = 0; i < AllWorkerList.Count; i++) {
          SearchTBox.Text += AllWorkerList[i].FIO;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "День народження: " +
AllWorkerList[i].BirthDate.ToShortDateString();
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Email: " + AllWorkerList[i].Email;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Адреса: " + AllWorkerList[i].Address;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "-----------------------------------------\r\n";
        }
      } else {
        SearchTBox.Text += "За заданими критеріями нічого не знайдено!";
      }
    }
  }
}
```

Лістинг 13. Клас «LendSearchForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Forestry.Forms.Serch {
  public partial class LendSearchForm : Form {
    private ValidationMy _validation = new ValidationMy();
    private LandProvider _LandProvider = new LandProvider();
    private List<Land> _LandList = new List<Land>();
```

```
    public LendSearchForm() {
      InitializeComponent();
    }


    private void LandNameBtn_Click(object sender, EventArgs e) {
      _LandList = _LandProvider.GetAllLandByLandName(LandNameTBox.Text);
      GetDataSearch(_LandList);
    }

    private void SquareBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _LandList = _LandProvider.GetAllLandBySquare(Convert.ToDouble(SquareTBox.Text));
        GetDataSearch(_LandList);
      }
    }


    public void GetDataSearch(List<Land> AllLandList) {
      SearchTBox.Text = "";
      if (AllLandList.Count > 0) {
        for (int i = 0; i < AllLandList.Count; i++) {
          SearchTBox.Text += "Назва угіддя: " + AllLandList[i].LandName;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Робітник: " + AllLandList[i].WorkerFIO;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Початок робіт: " + AllLandList[i].StartDate.ToShortDateString();
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Кінець робіт: " + AllLandList[i].EndDate.ToShortDateString();
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Площа: " + AllLandList[i].Square;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "-----------------------------------------\r\n";
        }
      } else {
        SearchTBox.Text += "За заданими критеріями нічого не знайдено!";
      }
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataConvertToDouble(SquareTBox.Text)) {
        SquareValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        SquareValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }
  }
}
```

Лістинг 14. Клас «PlantingsSearchForm»

```
using Forestry.AppCode;
using Forestry.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Forestry.Forms.Serch {
  public partial class PlantingsSearchForm : Form {
    private ValidationMy _validation = new ValidationMy();
    private PlantingsProvider _PlantingsProvider = new PlantingsProvider();
    private List<Plantings> _PlantingsList = new List<Plantings>();

    public PlantingsSearchForm() {
      InitializeComponent();
    }

    private void TreeSpeciesBtn_Click(object sender, EventArgs e) {
      _PlantingsList = _PlantingsProvider.GetAllPlantingsByTreeSpecies(TreeSpeciesTBox.Text);
      GetDataSearch(_PlantingsList);
    }

    private void PlantingAreaBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _PlantingsList =
_PlantingsProvider.GetAllPlantingsByPlantingArea(Convert.ToDouble(PlantingAreaTBox.Text)
);
        GetDataSearch(_PlantingsList);
      }
    }


    public void GetDataSearch(List<Plantings> AllPlantingsList) {
      SearchTBox.Text = "";
      if (AllPlantingsList.Count > 0) {
        for (int i = 0; i < AllPlantingsList.Count; i++) {
          SearchTBox.Text += "Порода дерева: " + AllPlantingsList[i].TreeSpecies;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Дата висадки: " +
AllPlantingsList[i].LandingDate.ToShortDateString();
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Кількість насаджень: " + AllPlantingsList[i].NumberOfPlantings;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Площа насаджень: " + AllPlantingsList[i].PlantingArea;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "----------------------------------------\r\n";
```

```
      }
     } else {
      SearchTBox.Text += "За заданими критеріями нічого не знайдено!";
     }
    }

    private bool IsDataEnteringCorrect() {
     bool isCorrect = true;
     if (_validation.IsDataConvertToInt(PlantingAreaTBox.Text)) {
      PlantingAreaValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      PlantingAreaValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     return isCorrect;
    }



  }
}
```

Лістинг 15. Клас «WorkerSearchForm»

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

namespace Forestry.Forms.Serch {
  public partial class WorkerSearchForm : Form {
    WorkerProvider _WorkerProvider = new WorkerProvider();
    List<Worker> _WorkerList = new List<Worker>();
    public WorkerSearchForm() {
     InitializeComponent();

    }

    private void LastNameBtn_Click(object sender, EventArgs e) {
     _WorkerList = _WorkerProvider.GetAllWorkerByLastName(LastNameTBox.Text);
     GetDataSearch(_WorkerList);
    }

    private void FirstNameBtn_Click(object sender, EventArgs e) {
     _WorkerList = _WorkerProvider.GetAllWorkerByFirstName(FirstNameTBox.Text);
     GetDataSearch(_WorkerList);
```

```
    }

    public void GetDataSearch(List<Worker> AllWorkerList) {
      SearchTBox.Text = "";
      if (AllWorkerList.Count > 0) {
        for (int i = 0; i < AllWorkerList.Count; i++) {
          SearchTBox.Text += AllWorkerList[i].FIO;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "День народження: " +
AllWorkerList[i].BirthDate.ToShortDateString();
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Email: " + AllWorkerList[i].Email;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Посада: " + AllWorkerList[i].PositionId;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "Адреса: " + AllWorkerList[i].Address;
          SearchTBox.Text += "\r\n";
          SearchTBox.Text += "----------------------------------------\r\n";
        }
      } else {
        SearchTBox.Text += "За заданими критеріями нічого не знайдено!";
      }
    }

  }
}
```

Лістинг 16. Клас «JobsProvider»

```
using Forestry.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Forestry.Provider {
  class JobsProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertJobs(string JobsName, string Description, DateTime JobsDate) {
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        using (SqlCommand command = new SqlCommand()) {
          command.Connection = connection;
          command.CommandType = CommandType.Text;
          command.CommandText = "INSERT INTO Jobs (JobsName, Description, JobsDate) " +
            " VALUES (@JobsName, @Description, @JobsDate)";
          command.Parameters.AddWithValue("@JobsName", JobsName);
          command.Parameters.AddWithValue("@Description", Description);
```

```
      command.Parameters.AddWithValue("@JobsDate", JobsDate);
      try {
        connection.Open();
        int recordsAffected = command.ExecuteNonQuery();
      } catch (SqlException) {
        // error here
      } finally {
        connection.Close();
      }
    }
  }
}

public Jobs SelectedJobsByJobsId(int JobsId) {
  int i = 0;
  Jobs selectedJobs = new Jobs();
  string sqlExpression = "SELECT * FROM Jobs WHERE JobsId=" + JobsId.ToString();
  using (SqlConnection connection = new SqlConnection(_ConnString)) {
    connection.Open();
    SqlCommand command = new SqlCommand(sqlExpression, connection);
    SqlDataReader reader = command.ExecuteReader();

    if (reader.HasRows) // если есть данные
    {
      // выводим названия столбцов
      while (reader.Read()) // построчно считываем данные
        {

        selectedJobs.Number = ++i;
        selectedJobs.JobsId = Convert.ToInt32(reader["JobsId"]);
        selectedJobs.JobsName = reader["JobsName"].ToString();
        selectedJobs.Description = reader["Description"].ToString();
        selectedJobs.JobsDate = Convert.ToDateTime(reader["JobsDate"]);
      }
    }
    reader.Close();
  }
  return selectedJobs;
}

public List<Jobs> GetAllJobs() {
  int i = 0;
  List<Jobs> JobsList = new List<Jobs>();
  string sqlExpression = "SELECT * FROM Jobs";
  using (SqlConnection connection = new SqlConnection(_ConnString)) {
    connection.Open();
    SqlCommand command = new SqlCommand(sqlExpression, connection);
    SqlDataReader reader = command.ExecuteReader();

    if (reader.HasRows) // если есть данные
    {
      // выводим названия столбцов
```

```
        while (reader.Read()) // построчно считываем данные
          {
          Jobs selectedJobs = new Jobs();
          selectedJobs.Number = ++i;
          selectedJobs.JobsId = Convert.ToInt32(reader["JobsId"]);
          selectedJobs.JobsName = reader["JobsName"].ToString();
          selectedJobs.Description = reader["Description"].ToString();
          selectedJobs.JobsDate = Convert.ToDateTime(reader["JobsDate"]);
          JobsList.Add(selectedJobs);
         }
        }
        reader.Close();
      }
      if (JobsList.Count == 0) {
        Jobs noJobs = new Jobs();
        noJobs.JobsId = 0;
        noJobs.Message = NamesMy.NoDataNames.NoDataInJobs;
        JobsList.Add(noJobs);
      }

      return JobsList;
    }

    public void UpdateJobs(string JobsName, string Description, DateTime JobsDate, int JobsId)
{
      using (SqlConnection con = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand("UPDATE Jobs SET JobsName =
@JobsName, Description = @Description, JobsDate=@JobsDate " +
          " WHERE JobsId = @JobsId", con)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@JobsName", JobsName);
          cmd.Parameters.AddWithValue("@Description", Description);
          cmd.Parameters.AddWithValue("@JobsDate", JobsDate);
          cmd.Parameters.AddWithValue("@JobsId", JobsId);
          con.Open();
          int rowsAffected = cmd.ExecuteNonQuery();
          con.Close();
        }
      }
    }

    public void DeleteJobsByJobsId(int JobsId) {
      string sqlExpression = "DELETE  FROM Jobs WHERE JobsId=" + JobsId.ToString();
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        command.ExecuteNonQuery();
        connection.Close();
      }
    }

  }
```

```
}

public class Jobs {
  private int _Number;
  private int _JobsId;
  private string _JobsName;
  private string _Description;
  private DateTime _JobsDate;
  private string _Message;

  public Jobs() {
    _Number = 0;
    _JobsId = 0;
    _JobsName = String.Empty;
    _Description = String.Empty;
    _JobsDate = new DateTime();
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int JobsId {
    set { _JobsId = value; }
    get { return _JobsId; }
  }
  public string JobsName {
    set { _JobsName = value; }
    get { return _JobsName; }
  }
  public string Description {
    set { _Description = value; }
    get { return _Description; }
  }
  public DateTime JobsDate {
    set { _JobsDate = value; }
    get { return _JobsDate; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 17. Клас «LandProvider»

```
using Forestry.AppCode;

using System;
```

```
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Forestry.Provider {
  class LandProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertLand(string LandName, double Square, int PlantingsId, int WorkerId, int
JobsId, DateTime StartDate, DateTime EndDate) {
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        using (SqlCommand command = new SqlCommand()) {
          command.Connection = connection;
          command.CommandType = CommandType.Text;
          command.CommandText = "INSERT INTO Land (LandName, Square, PlantingsId,
WorkerId, JobsId, StartDate, EndDate) " +
            " VALUES (@LandName, @Square, @PlantingsId, @WorkerId, @JobsId, @StartDate,
@EndDate)";
          command.Parameters.AddWithValue("@LandName", LandName);
          command.Parameters.AddWithValue("@Square", Square);
          command.Parameters.AddWithValue("@PlantingsId", PlantingsId);
          command.Parameters.AddWithValue("@WorkerId", WorkerId);
          command.Parameters.AddWithValue("@JobsId", JobsId);
          command.Parameters.AddWithValue("@StartDate", StartDate);
          command.Parameters.AddWithValue("@EndDate", EndDate);
          try {
            connection.Open();
            int recordsAffected = command.ExecuteNonQuery();
          } catch (SqlException) {
```

```
    } finally {
      connection.Close();
    }
   }
  }
}

public Land SelectedLandByLandId(int LandId) {
  int i = 0;
  Land selectedLand = new Land();
  string sqlExpression = "SELECT * FROM Land WHERE LandId=" + LandId.ToString();
  using (SqlConnection connection = new SqlConnection(_ConnString)) {
   connection.Open();
   SqlCommand command = new SqlCommand(sqlExpression, connection);
   SqlDataReader reader = command.ExecuteReader();

   if (reader.HasRows) // если есть данные
   {
    // выводим названия столбцов
    while (reader.Read()) // построчно считываем данные
     {
      selectedLand.LandId = Convert.ToInt32(reader["LandId"]);
      selectedLand.LandName = reader["LandName"].ToString();
      selectedLand.Square = Convert.ToDouble(reader["Square"]);
      selectedLand.PlantingsId = Convert.ToInt32(reader["PlantingsId"]);
      selectedLand.WorkerId = Convert.ToInt32(reader["WorkerId"]);
      selectedLand.JobsId = Convert.ToInt32(reader["JobsId"]);
      selectedLand.StartDate = Convert.ToDateTime(reader["StartDate"]);
      selectedLand.EndDate = Convert.ToDateTime(reader["EndDate"]);
    }
   }
   reader.Close();
```

```
  }
  return selectedLand;
}

public List<Land> GetAllLand() {
  int i = 0;
  List<Land> LandList = new List<Land>();
  string sqlExpression = "SELECT * FROM Land";
  using (SqlConnection connection = new SqlConnection(_ConnString)) {
    connection.Open();
    SqlCommand command = new SqlCommand(sqlExpression, connection);
    SqlDataReader reader = command.ExecuteReader();

    if (reader.HasRows) // если есть данные
    {
      // выводим названия столбцов
      while (reader.Read()) // построчно считываем данные
        {
        Land selectedLand = new Land();
        selectedLand.Number = ++i;
        selectedLand.LandId = Convert.ToInt32(reader["LandId"]);
        selectedLand.LandName = reader["LandName"].ToString();
        selectedLand.Square = Convert.ToDouble(reader["Square"]);
        selectedLand.PlantingsId = Convert.ToInt32(reader["PlantingsId"]);
        selectedLand.WorkerId = Convert.ToInt32(reader["WorkerId"]);
        selectedLand.JobsId = Convert.ToInt32(reader["JobsId"]);
        selectedLand.StartDate = Convert.ToDateTime(reader["StartDate"]);
        selectedLand.EndDate = Convert.ToDateTime(reader["EndDate"]);
        LandList.Add(selectedLand);
      }
    }
    reader.Close();
```

```
  }
  if (LandList.Count == 0) {
    Land noLand = new Land();
    noLand.LandId = 0;
    noLand.Message = NamesMy.NoDataNames.NoDataInLand;
    LandList.Add(noLand);
  }

  return LandList;
}

public List<Land> GetAllLandByJobsId(int JobsId) {
  int i = 0;
  WorkerProvider WorkerProvider = new WorkerProvider();
  List<Worker> WorkerList = new List<Worker>();
  WorkerList = WorkerProvider.GetAllWorker();

  List<Land> LandList = new List<Land>();
  string sqlExpression = "SELECT * FROM Land  WHERE JobsId=" + JobsId.ToString();
  using (SqlConnection connection = new SqlConnection(_ConnString)) {
    connection.Open();
    SqlCommand command = new SqlCommand(sqlExpression, connection);
    SqlDataReader reader = command.ExecuteReader();

    if (reader.HasRows) // если есть данные
    {
      // выводим названия столбцов
      while (reader.Read()) // построчно считываем данные
        {
        Land selectedLand = new Land();
        selectedLand.Number = ++i;
        selectedLand.LandId = Convert.ToInt32(reader["LandId"]);
```

```
            selectedLand.LandName = reader["LandName"].ToString();
            selectedLand.Square = Convert.ToDouble(reader["Square"]);
            selectedLand.PlantingsId = Convert.ToInt32(reader["PlantingsId"]);
            selectedLand.WorkerId = Convert.ToInt32(reader["WorkerId"]);
            selectedLand.JobsId = Convert.ToInt32(reader["JobsId"]);
            selectedLand.StartDate = Convert.ToDateTime(reader["StartDate"]);
            selectedLand.EndDate = Convert.ToDateTime(reader["EndDate"]);
            LandList.Add(selectedLand);
          }
        }
        reader.Close();
      }

      for (int j = 0; j < LandList.Count; j++) {
        LandList[j].WorkerFIO = GetWorkerFIO(LandList[j].WorkerId, WorkerList);
      }
      return LandList;
    }

    public List<Land> GetAllLandByLandName(string LandName) {
      int i = 0;
      WorkerProvider WorkerProvider = new WorkerProvider();
      List<Worker> WorkerList = new List<Worker>();
      WorkerList = WorkerProvider.GetAllWorker();

      List<Land> LandList = new List<Land>();
      string sqlExpression = "SELECT * FROM Land  WHERE LandName LIKE N'%" +
LandName + "%'";
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        SqlDataReader reader = command.ExecuteReader();
```

```
    if (reader.HasRows) // если есть данные
    {
      // выводим названия столбцов
      while (reader.Read()) // построчно считываем данные
        {
        Land selectedLand = new Land();
        selectedLand.Number = ++i;
        selectedLand.LandId = Convert.ToInt32(reader["LandId"]);
        selectedLand.LandName = reader["LandName"].ToString();
        selectedLand.Square = Convert.ToDouble(reader["Square"]);
        selectedLand.PlantingsId = Convert.ToInt32(reader["PlantingsId"]);
        selectedLand.WorkerId = Convert.ToInt32(reader["WorkerId"]);
        selectedLand.JobsId = Convert.ToInt32(reader["JobsId"]);
        selectedLand.StartDate = Convert.ToDateTime(reader["StartDate"]);
        selectedLand.EndDate = Convert.ToDateTime(reader["EndDate"]);
        LandList.Add(selectedLand);
      }
    }
    reader.Close();
  }

  for (int j = 0; j < LandList.Count; j++) {
    LandList[j].WorkerFIO = GetWorkerFIO(LandList[j].WorkerId, WorkerList);
  }
  return LandList;
}

public List<Land> GetAllLandBySquare(double Square) {
  int i = 0;
  WorkerProvider WorkerProvider = new WorkerProvider();
  List<Worker> WorkerList = new List<Worker>();
```

```
WorkerList = WorkerProvider.GetAllWorker();

List<Land> LandList = new List<Land>();
string sqlExpression = "SELECT * FROM Land WHERE Square <=" + Square;
using (SqlConnection connection = new SqlConnection(_ConnString)) {
  connection.Open();
  SqlCommand command = new SqlCommand(sqlExpression, connection);
  SqlDataReader reader = command.ExecuteReader();

  if (reader.HasRows) // если есть данные
  {
    // выводим названия столбцов
    while (reader.Read()) // построчно считываем данные
      {
      Land selectedLand = new Land();
      selectedLand.Number = ++i;
      selectedLand.LandId = Convert.ToInt32(reader["LandId"]);
      selectedLand.LandName = reader["LandName"].ToString();
      selectedLand.Square = Convert.ToDouble(reader["Square"]);
      selectedLand.PlantingsId = Convert.ToInt32(reader["PlantingsId"]);
      selectedLand.WorkerId = Convert.ToInt32(reader["WorkerId"]);
      selectedLand.JobsId = Convert.ToInt32(reader["JobsId"]);
      selectedLand.StartDate = Convert.ToDateTime(reader["StartDate"]);
      selectedLand.EndDate = Convert.ToDateTime(reader["EndDate"]);
      LandList.Add(selectedLand);
    }
  }
  reader.Close();
}

for (int j = 0; j < LandList.Count; j++) {
  LandList[j].WorkerFIO = GetWorkerFIO(LandList[j].WorkerId, WorkerList);
```

```
    }

    return LandList;
  }

  private string GetWorkerFIO(int WorkerId, List<Worker> WorkerList) {
    for (int i = 0; i < WorkerList.Count; i++) {
      if (WorkerId == WorkerList[i].WorkerId) {
        return WorkerList[i].FIO;
      }
    }
    return "";
  }

  public void UpdateLand(string LandName, double Square, int PlantingsId, int WorkerId, int
JobsId, DateTime StartDate, DateTime EndDate, int LandId) {
    using (SqlConnection con = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand("UPDATE Land SET LandName =
@LandName, Square = @Square, PlantingsId=@PlantingsId, " +
        "WorkerId=@WorkerId, JobsId=@JobsId, StartDate=@StartDate, EndDate=@EndDate "
+
        " WHERE LandId = @LandId", con)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("@LandName", LandName);
        cmd.Parameters.AddWithValue("@Square", Square);
        cmd.Parameters.AddWithValue("@PlantingsId", PlantingsId);
        cmd.Parameters.AddWithValue("@WorkerId", WorkerId);
        cmd.Parameters.AddWithValue("@JobsId", JobsId);
        cmd.Parameters.AddWithValue("@StartDate", StartDate);
        cmd.Parameters.AddWithValue("@EndDate", EndDate);
        cmd.Parameters.AddWithValue("@LandId", LandId);
        con.Open();
        int rowsAffected = cmd.ExecuteNonQuery();
```

```
            con.Close();
          }
        }
      }

      public void DeleteLandByLandId(int LandId) {
        string sqlExpression = "DELETE  FROM Land WHERE LandId=" + LandId.ToString();
        using (SqlConnection connection = new SqlConnection(_ConnString)) {
          connection.Open();
          SqlCommand command = new SqlCommand(sqlExpression, connection);
          command.ExecuteNonQuery();
          connection.Close();
        }
      }

    }
  }

  public class Land {
    private int _Number;
    private int _LandId;
    private string _LandName;
    private double _Square;
    private int _PlantingsId;
    private int _WorkerId;
    private string _WorkerFIO;
    private int _JobsId;
    private DateTime _StartDate;
    private DateTime _EndDate;
    private string _Message;

    public Land() {
```

```
  _Number = 0;
  _LandId = 0;
  _LandName = String.Empty;
  _Square = 0.0;
  _PlantingsId = 0;
  _WorkerId = 0;
  _WorkerFIO = String.Empty;
  _JobsId = 0;
  _StartDate = new DateTime();
  _EndDate = new DateTime();
  _Message = String.Empty;
}

public int Number {
  set { _Number = value; }
  get { return _Number; }
}
public int LandId {
  set { _LandId = value; }
  get { return _LandId; }
}
public string LandName {
  set { _LandName = value; }
  get { return _LandName; }
}
public double Square {
  set { _Square = value; }
  get { return _Square; }
}
public int PlantingsId {
  set { _PlantingsId = value; }
  get { return _PlantingsId; }
```

```
  }
  public int WorkerId {
    set { _WorkerId = value; }
    get { return _WorkerId; }
  }
  public string WorkerFIO {
    set { _WorkerFIO = value; }
    get { return _WorkerFIO; }
  }
  public int JobsId {
    set{ _JobsId = value; }
    get { return _JobsId; }
  }
  public DateTime StartDate {
    set { _StartDate = value; }
    get { return _StartDate; }
  }
  public DateTime EndDate {
    set { _EndDate = value; }
    get { return _EndDate; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 18. Клас «PlantingsProvider»

```
using Forestry.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Forestry.Provider {
  class PlantingsProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertPlantings(string TreeSpecies, string Description, DateTime LandingDate, int
NumberOfPlantings, double PlantingArea) {
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        using (SqlCommand command = new SqlCommand()) {
          command.Connection = connection;
          command.CommandType = CommandType.Text;
          command.CommandText = "INSERT INTO Plantings (TreeSpecies, Description,
LandingDate, NumberOfPlantings, PlantingArea) " +
            " VALUES (@TreeSpecies, @Description, @LandingDate, @NumberOfPlantings,
@PlantingArea)";
          command.Parameters.AddWithValue("@TreeSpecies", TreeSpecies);
          command.Parameters.AddWithValue("@Description", Description);
          command.Parameters.AddWithValue("@LandingDate", LandingDate);
          command.Parameters.AddWithValue("@NumberOfPlantings", NumberOfPlantings);
          command.Parameters.AddWithValue("@PlantingArea", PlantingArea);
          try {
            connection.Open();
            int recordsAffected = command.ExecuteNonQuery();
          } catch (SqlException) {
          } finally {
            connection.Close();
          }
        }
      }
```

```
    }


    public Plantings SelectedPlantingsByPlantingsId(int PlantingsId) {
      int i = 0;
      Plantings selectedPlantings = new Plantings();
      string sqlExpression = "SELECT * FROM Plantings WHERE PlantingsId=" +
PlantingsId.ToString();
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        SqlDataReader reader = command.ExecuteReader();


        if (reader.HasRows) // если есть данные
        {
          // выводим названия столбцов
          while (reader.Read()) // построчно считываем данные
            {


            selectedPlantings.Number = ++i;
            selectedPlantings.PlantingsId = Convert.ToInt32(reader["PlantingsId"]);
            selectedPlantings.TreeSpecies = reader["TreeSpecies"].ToString();
            selectedPlantings.Description = reader["Description"].ToString();
            selectedPlantings.LandingDate = Convert.ToDateTime(reader["LandingDate"]);
            selectedPlantings.NumberOfPlantings =
Convert.ToInt32(reader["NumberOfPlantings"]);
            selectedPlantings.PlantingArea = Convert.ToDouble(reader["PlantingArea"]);
          }
        }
        reader.Close();
      }
      return selectedPlantings;
```

```
    }

    public List<Plantings> GetAllPlantings() {
      int i = 0;
      List<Plantings> PlantingsList = new List<Plantings>();
      string sqlExpression = "SELECT * FROM Plantings";
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        SqlDataReader reader = command.ExecuteReader();

        if (reader.HasRows) // если есть данные
        {
          // выводим названия столбцов
          while (reader.Read()) // построчно считываем данные
            {
            Plantings selectedPlantings = new Plantings();
            selectedPlantings.Number = ++i;
            selectedPlantings.PlantingsId = Convert.ToInt32(reader["PlantingsId"]);
            selectedPlantings.TreeSpecies = reader["TreeSpecies"].ToString();
            selectedPlantings.Description = reader["Description"].ToString();
            selectedPlantings.LandingDate = Convert.ToDateTime(reader["LandingDate"]);
            selectedPlantings.NumberOfPlantings =
Convert.ToInt32(reader["NumberOfPlantings"]);
            selectedPlantings.PlantingArea = Convert.ToDouble(reader["PlantingArea"]);
            PlantingsList.Add(selectedPlantings);
          }
        }
        reader.Close();
      }
      if (PlantingsList.Count == 0) {
        Plantings noPlantings = new Plantings();
```

```
        noPlantings.PlantingsId = 0;
        noPlantings.Message = NamesMy.NoDataNames.NoDataInPlantings;
        PlantingsList.Add(noPlantings);
      }

      return PlantingsList;
    }

    public List<Plantings> GetAllPlantingsByTreeSpecies(string TreeSpecies) {
      int i = 0;
      List<Plantings> PlantingsList = new List<Plantings>();
      string sqlExpression = "SELECT * FROM Plantings  WHERE TreeSpecies LIKE N'%" +
TreeSpecies + "%'";
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        SqlDataReader reader = command.ExecuteReader();

        if (reader.HasRows) // если есть данные
        {
          // выводим названия столбцов
          while (reader.Read()) // построчно считываем данные
            {
            Plantings selectedPlantings = new Plantings();
            selectedPlantings.Number = ++i;
            selectedPlantings.PlantingsId = Convert.ToInt32(reader["PlantingsId"]);
            selectedPlantings.TreeSpecies = reader["TreeSpecies"].ToString();
            selectedPlantings.Description = reader["Description"].ToString();
            selectedPlantings.LandingDate = Convert.ToDateTime(reader["LandingDate"]);
            selectedPlantings.NumberOfPlantings =
Convert.ToInt32(reader["NumberOfPlantings"]);
            selectedPlantings.PlantingArea = Convert.ToDouble(reader["PlantingArea"]);
```

```
            PlantingsList.Add(selectedPlantings);
          }
        }
        reader.Close();
      }
      return PlantingsList;
    }

    public List<Plantings> GetAllPlantingsByPlantingArea(double PlantingArea) {
      int i = 0;
      List<Plantings> PlantingsList = new List<Plantings>();
      string sqlExpression = "SELECT * FROM Plantings WHERE PlantingArea <=" +
PlantingArea; ;
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        SqlDataReader reader = command.ExecuteReader();

        if (reader.HasRows)
        {
          while (reader.Read())
            {
            Plantings selectedPlantings = new Plantings();
            selectedPlantings.Number = ++i;
            selectedPlantings.PlantingsId = Convert.ToInt32(reader["PlantingsId"]);
            selectedPlantings.TreeSpecies = reader["TreeSpecies"].ToString();
            selectedPlantings.Description = reader["Description"].ToString();
            selectedPlantings.LandingDate = Convert.ToDateTime(reader["LandingDate"]);
            selectedPlantings.NumberOfPlantings =
Convert.ToInt32(reader["NumberOfPlantings"]);
            selectedPlantings.PlantingArea = Convert.ToDouble(reader["PlantingArea"]);
            PlantingsList.Add(selectedPlantings);
```

```
        }
       }
        reader.Close();
      }

      return PlantingsList;
    }

    public void UpdatePlantings(string TreeSpecies, string Description, DateTime LandingDate,
int NumberOfPlantings, double PlantingArea, int PlantingsId) {
      using (SqlConnection con = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand("UPDATE Plantings SET TreeSpecies =
@TreeSpecies, Description = @Description, LandingDate=@LandingDate, " +
          "NumberOfPlantings=@NumberOfPlantings, PlantingArea=@PlantingArea " +
          " WHERE PlantingsId = @PlantingsId", con)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@TreeSpecies", TreeSpecies);
          cmd.Parameters.AddWithValue("@Description", Description);
          cmd.Parameters.AddWithValue("@LandingDate", LandingDate);
          cmd.Parameters.AddWithValue("@NumberOfPlantings", NumberOfPlantings);
          cmd.Parameters.AddWithValue("@PlantingArea", PlantingArea);
          cmd.Parameters.AddWithValue("@PlantingsId", PlantingsId);
          con.Open();
          int rowsAffected = cmd.ExecuteNonQuery();
          con.Close();
        }
      }
    }

    public void DeletePlantingsByPlantingsId(int PlantingsId) {
      string sqlExpression = "DELETE  FROM Plantings WHERE PlantingsId=" +
PlantingsId.ToString();
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
```

```
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        command.ExecuteNonQuery();
        connection.Close();
      }
    }

  }
}

public class Plantings {
  private int _Number;
  private int _PlantingsId;
  private string _TreeSpecies;
  private DateTime _LandingDate;
  private int _NumberOfPlantings;
  private double _PlantingArea;
  private string _Description;
  private string _Message;

  public Plantings() {
    _Number = 0;
    _PlantingsId = 0;
    _TreeSpecies = String.Empty;
    _LandingDate = new DateTime();
    _NumberOfPlantings = 0;
    _PlantingArea = 0.0;
    _Description = String.Empty;
    _Message = String.Empty;
  }

  public int Number {
```

```
  set { _Number = value; }
  get { return _Number; }
}
public int PlantingsId {
  set { _PlantingsId = value; }
  get { return _PlantingsId; }
}
public string TreeSpecies {
  set { _TreeSpecies = value; }
  get { return _TreeSpecies; }
}
public DateTime LandingDate {
  set { _LandingDate = value; }
  get { return _LandingDate; }
}
public int NumberOfPlantings {
  set { _NumberOfPlantings = value; }
  get { return _NumberOfPlantings; }
}
public double PlantingArea {
  set { _PlantingArea = value; }
  get{ return _PlantingArea; }
}
public string Description {
  set { _Description = value; }
  get { return _Description; }
}

public string Message {
  set { _Message = value; }
  get { return _Message; }
}
```

```
}
```

Лістинг 19. Клас «PositionProvider»

```
using Forestry.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Forestry.Provider {
  class PositionProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertPosition(string PositionName, string Description) {
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        using (SqlCommand command = new SqlCommand()) {
          command.Connection = connection;
          command.CommandType = CommandType.Text;
          command.CommandText = "INSERT INTO Position (PositionName, Description) " +
            " VALUES (@PositionName, @Description)";
          command.Parameters.AddWithValue("@PositionName", PositionName);
          command.Parameters.AddWithValue("@Description", Description);
          try {
            connection.Open();
            int recordsAffected = command.ExecuteNonQuery();
          } catch (SqlException) {
            // error here
          } finally {
            connection.Close();
          }
        }
      }
    }




    public Position SelectedPositionByPositionId(int PositionId) {
      int i = 0;
      Position selectedPosition = new Position();
      string sqlExpression = "SELECT * FROM Position WHERE PositionId=" +
PositionId.ToString();
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        SqlDataReader reader = command.ExecuteReader();
```

```
    if (reader.HasRows) // если есть данные
    {
      // выводим названия столбцов
      while (reader.Read()) // построчно считываем данные
        {

        selectedPosition.Number = ++i;
        selectedPosition.PositionId = Convert.ToInt32(reader["PositionId"]);
        selectedPosition.PositionName = reader["PositionName"].ToString();
        selectedPosition.Description = reader["Description"].ToString();
      }
    }
    reader.Close();
  }
  return selectedPosition;
}

public List<Position> GetAllPosition() {
  int i = 0;
  List<Position> PositionList = new List<Position>();
  string sqlExpression = "SELECT * FROM Position";
  using (SqlConnection connection = new SqlConnection(_ConnString)) {
    connection.Open();
    SqlCommand command = new SqlCommand(sqlExpression, connection);
    SqlDataReader reader = command.ExecuteReader();

    if (reader.HasRows) // если есть данные
    {
      // выводим названия столбцов
      while (reader.Read()) // построчно считываем данные
        {
        Position onePosition = new Position();
        onePosition.Number = ++i;
        onePosition.PositionId = Convert.ToInt32(reader["PositionId"]);
        onePosition.PositionName = reader["PositionName"].ToString();
        onePosition.Description = reader["Description"].ToString();
        PositionList.Add(onePosition);
      }
    }
    reader.Close();
  }
  if (PositionList.Count == 0) {
    Position noPosition = new Position();
    noPosition.PositionId = 0;
    noPosition.Message = NamesMy.NoDataNames.NoDataInPosition;
    PositionList.Add(noPosition);
  }

  return PositionList;
}
```

```
    public void UpdatePosition(string PositionName, string Description, int PositionId) {
      using (SqlConnection con = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand("UPDATE Position SET PositionName =
@PositionName, Description = @Description " +
          " WHERE PositionId = @PositionId", con)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@PositionName", PositionName);
          cmd.Parameters.AddWithValue("@Description", Description);
          cmd.Parameters.AddWithValue("@PositionId", PositionId);
          con.Open();
          int rowsAffected = cmd.ExecuteNonQuery();
          con.Close();
        }
      }
    }

    public void DeletePositionByPositionId(int PositionId) {
      string sqlExpression = "DELETE  FROM Position WHERE PositionId=" +
PositionId.ToString();
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        command.ExecuteNonQuery();
        connection.Close();
      }
    }

  }

  class CopyOfPositionProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertPosition(string PositionName, string Description) {
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        using (SqlCommand command = new SqlCommand()) {
          command.Connection = connection;
          command.CommandType = CommandType.Text;
          command.CommandText = "INSERT INTO Position (PositionName, Description) " +
            " VALUES (@PositionName, @Description)";
          command.Parameters.AddWithValue("@PositionName", PositionName);
          command.Parameters.AddWithValue("@Description", Description);
          try {
            connection.Open();
            int recordsAffected = command.ExecuteNonQuery();
          } catch (SqlException) {
            // error here
          } finally {
            connection.Close();
          }
        }
      }
```

```
    }



    public Position SelectedPositionByPositionId(int PositionId) {
      int i = 0;
      Position selectedPosition = new Position();
      string sqlExpression = "SELECT * FROM Position WHERE PositionId=" +
PositionId.ToString();
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        SqlDataReader reader = command.ExecuteReader();

        if (reader.HasRows) // если есть данные
        {
          // выводим названия столбцов
          while (reader.Read()) // построчно считываем данные
            {

            selectedPosition.Number = ++i;
            selectedPosition.PositionId = Convert.ToInt32(reader["PositionId"]);
            selectedPosition.PositionName = reader["PositionName"].ToString();
            selectedPosition.Description = reader["Description"].ToString();
          }
        }
        reader.Close();
      }
      return selectedPosition;
    }

    public List<Position> GetAllPosition() {
      int i = 0;
      List<Position> PositionList = new List<Position>();
      string sqlExpression = "SELECT * FROM Position";
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        SqlDataReader reader = command.ExecuteReader();

        if (reader.HasRows) // если есть данные
        {
          // выводим названия столбцов
          while (reader.Read()) // построчно считываем данные
            {
            Position onePosition = new Position();
            onePosition.Number = ++i;
            onePosition.PositionId = Convert.ToInt32(reader["PositionId"]);
            onePosition.PositionName = reader["PositionName"].ToString();
            onePosition.Description = reader["Description"].ToString();
            PositionList.Add(onePosition);
```

```
        }
       }
       reader.Close();
      }
      if (PositionList.Count == 0) {
       Position noPosition = new Position();
       noPosition.PositionId = 0;
       noPosition.Message = NamesMy.NoDataNames.NoDataInPosition;
       PositionList.Add(noPosition);
      }

      return PositionList;
    }

    public void UpdatePosition(string PositionName, string Description, int PositionId) {
      using (SqlConnection con = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand("UPDATE Position SET PositionName =
@PositionName, Description = @Description " +
          " WHERE PositionId = @PositionId", con)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@PositionName", PositionName);
          cmd.Parameters.AddWithValue("@Description", Description);
          cmd.Parameters.AddWithValue("@PositionId", PositionId);
          con.Open();
          int rowsAffected = cmd.ExecuteNonQuery();
          con.Close();
        }
      }
    }

    public void DeletePositionByPositionId(int PositionId) {
      string sqlExpression = "DELETE  FROM Position WHERE PositionId=" +
PositionId.ToString();
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        command.ExecuteNonQuery();
        connection.Close();
      }
    }

  }
}


public class Position {
  private int _Number;
  private int _PositionId;
  private string _PositionName;
  private string _Description;
  private string _Message;
```

```
  public Position() {
    _Number = 0;
    _PositionId = 0;
    _PositionName = String.Empty;
    _Description = String.Empty;
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int PositionId {
    set { _PositionId = value; }
    get { return _PositionId; }
  }
  public string PositionName {
    set { _PositionName = value; }
    get { return _PositionName; }
  }
  public string Description {
    set { _Description = value; }
    get { return _Description; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 20. Клас «WorkerProvider»

```
using Forestry.AppCode;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;


namespace Forestry.Provider {

  class WorkerProvider {

    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];
```

```
    public void InsertWorker(string LastName, string FirstName, string Phone, string Address, string Email, DateTime BirthDate, int PositionId) {
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        using (SqlCommand command = new SqlCommand()) {
          command.Connection = connection;
          command.CommandType = CommandType.Text;
          command.CommandText = "INSERT into Worker (LastName, FirstName, Phone, Address, Email, BirthDate, PositionId) " +
            " VALUES (@LastName, @FirstName, @Phone, @Address, @Email, @BirthDate, @PositionId)";
          command.Parameters.AddWithValue("@LastName", LastName);
          command.Parameters.AddWithValue("@FirstName", FirstName);
          command.Parameters.AddWithValue("@Phone", Phone);
          command.Parameters.AddWithValue("@Address", Address);
          command.Parameters.AddWithValue("@Email", Email);
          command.Parameters.AddWithValue("@BirthDate", BirthDate.ToString("yyyy-MM-dd HH:mm:ss"));
          command.Parameters.AddWithValue("@PositionId", PositionId);
          try {
            connection.Open();
            int recordsAffected = command.ExecuteNonQuery();
          } catch (SqlException) {
            // error here
          } finally {
            connection.Close();
          }
        }
      }
    }
    public Worker SelectedWorkerByWorkerId(int WorkerId) {
      int i = 0;
      Worker selectedWorker = new Worker();
```

```
    string sqlExpression = "SELECT * FROM Worker WHERE WorkerId=" +
WorkerId.ToString();
    using (SqlConnection connection = new SqlConnection(_ConnString)) {
      connection.Open();
      SqlCommand command = new SqlCommand(sqlExpression, connection);
      SqlDataReader reader = command.ExecuteReader();

      if (reader.HasRows) // если есть данные
      {
        // выводим названия столбцов
        while (reader.Read()) // построчно считываем данные
          {

          selectedWorker.Number = ++i;
          selectedWorker.WorkerId = Convert.ToInt32(reader["WorkerId"]);
          selectedWorker.LastName = reader["LastName"].ToString();
          selectedWorker.FirstName = reader["FirstName"].ToString();
          selectedWorker.Phone = reader["Phone"].ToString();
          selectedWorker.BirthDate = Convert.ToDateTime(reader["BirthDate"]);
          selectedWorker.Address = reader["Address"].ToString();
          selectedWorker.Email = reader["Email"].ToString();
          selectedWorker.PositionId = Convert.ToInt32(reader["PositionId"]);
        }
      }
      reader.Close();
    }
    return selectedWorker;
  }
  public List<Worker> GetAllWorker() {
    int i = 0;
    List<Worker> WorkerList = new List<Worker>();
    string sqlExpression = "SELECT * FROM Worker";
```

```
using (SqlConnection connection = new SqlConnection(_ConnString)) {
  connection.Open();
  SqlCommand command = new SqlCommand(sqlExpression, connection);
  SqlDataReader reader = command.ExecuteReader();

  if (reader.HasRows) // если есть данные
  {
    // выводим названия столбцов
    while (reader.Read()) // построчно считываем данные
      {
      Worker selectedWorker = new Worker();
      selectedWorker.Number = ++i;
      selectedWorker.WorkerId = Convert.ToInt32(reader["WorkerId"]);
      selectedWorker.LastName = reader["LastName"].ToString();
      selectedWorker.FirstName = reader["FirstName"].ToString();
      selectedWorker.FIO = selectedWorker.LastName + " " + selectedWorker.FirstName;
      selectedWorker.Phone = reader["Phone"].ToString();
      selectedWorker.BirthDate = Convert.ToDateTime(reader["BirthDate"]);
      selectedWorker.Address = reader["Address"].ToString();
      selectedWorker.Email = reader["Email"].ToString();
      selectedWorker.PositionId = Convert.ToInt32(reader["PositionId"]);
      WorkerList.Add(selectedWorker);
    }
  }
  reader.Close();
}
if (WorkerList.Count == 0) {
  Worker noWorker = new Worker();
  noWorker.WorkerId = 0;
  noWorker.Message = NamesMy.NoDataNames.NoDataInWorker;
  WorkerList.Add(noWorker);
}
```

```
    return WorkerList;
  }
  private string GetPositionName(int PositionId, List<Position> PositionList) {
    for (int i = 0; i < PositionList.Count; i++) {
      if (PositionId == PositionList[i].PositionId) {
        return PositionList[i].PositionName;
      }
    }
    return "";
  }

  public List<Worker> GetAllWorkerByFirstName(string FirstName) {
    PositionProvider positionProvider = new PositionProvider();
    List<Position> positionList = new List<Position>();
    positionList = positionProvider.GetAllPosition();
    int i = 0;
    List<Worker> WorkerList = new List<Worker>();
    string sqlExpression = "SELECT * FROM Worker WHERE FirstName LIKE N'%" +
FirstName + "%'";
    using (SqlConnection connection = new SqlConnection(_ConnString)) {
      connection.Open();
      SqlCommand command = new SqlCommand(sqlExpression, connection);
      SqlDataReader reader = command.ExecuteReader();

      if (reader.HasRows) // если есть данные
      {
        // выводим названия столбцов
        while (reader.Read()) // построчно считываем данные
          {
          Worker selectedWorker = new Worker();
          selectedWorker.Number = ++i;
```

```
            selectedWorker.WorkerId = Convert.ToInt32(reader["WorkerId"]);
            selectedWorker.LastName = reader["LastName"].ToString();
            selectedWorker.FirstName = reader["FirstName"].ToString();
            selectedWorker.FIO = selectedWorker.LastName + " " + selectedWorker.FirstName;
            selectedWorker.Phone = reader["Phone"].ToString();
            selectedWorker.BirthDate = Convert.ToDateTime(reader["BirthDate"]);
            selectedWorker.Address = reader["Address"].ToString();
            selectedWorker.Email = reader["Email"].ToString();
            selectedWorker.PositionId = Convert.ToInt32(reader["PositionId"]);
            WorkerList.Add(selectedWorker);
          }
        }
        reader.Close();
      }

      for (int j = 0; j < WorkerList.Count; j++) {
        WorkerList[j].PositionName = GetPositionName(WorkerList[j].PositionId, positionList);
      }

      return WorkerList;
    }

    public List<Worker> GetAllWorkerByLastName(string LastName) {
      PositionProvider positionProvider = new PositionProvider();
      List<Position> positionList = new List<Position>();
      positionList = positionProvider.GetAllPosition();
      int i = 0;
      List<Worker> WorkerList = new List<Worker>();
      string sqlExpression = "SELECT * FROM Worker WHERE LastName LIKE N'%" +
LastName + "%'";
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
```

```
    SqlCommand command = new SqlCommand(sqlExpression, connection);
    SqlDataReader reader = command.ExecuteReader();

    if (reader.HasRows) // если есть данные
    {
      // выводим названия столбцов
      while (reader.Read()) // построчно считываем данные
        {
        Worker selectedWorker = new Worker();
        selectedWorker.Number = ++i;
        selectedWorker.WorkerId = Convert.ToInt32(reader["WorkerId"]);
        selectedWorker.LastName = reader["LastName"].ToString();
        selectedWorker.FirstName = reader["FirstName"].ToString();
        selectedWorker.FIO = selectedWorker.LastName + " " + selectedWorker.FirstName;
        selectedWorker.Phone = reader["Phone"].ToString();
        selectedWorker.BirthDate = Convert.ToDateTime(reader["BirthDate"]);
        selectedWorker.Address = reader["Address"].ToString();
        selectedWorker.Email = reader["Email"].ToString();
        selectedWorker.PositionId = Convert.ToInt32(reader["PositionId"]);
        WorkerList.Add(selectedWorker);
      }
    }
    reader.Close();
  }
  for (int j = 0; j < WorkerList.Count; j++) {
    WorkerList[j].PositionName = GetPositionName(WorkerList[j].PositionId, positionList);
  }
  return WorkerList;
}

public List<Worker> GetAllWorkerByPositionId(int PositionId) {
  PositionProvider positionProvider = new PositionProvider();
```

```
    List<Position> positionList = new List<Position>();
    positionList = positionProvider.GetAllPosition();
    int i = 0;
    List<Worker> WorkerList = new List<Worker>();
    string sqlExpression = "SELECT * FROM Worker WHERE PositionId=" +
PositionId.ToString();
    using (SqlConnection connection = new SqlConnection(_ConnString)) {
      connection.Open();
      SqlCommand command = new SqlCommand(sqlExpression, connection);
      SqlDataReader reader = command.ExecuteReader();

      if (reader.HasRows) // если есть данные
      {
        // выводим названия столбцов
        while (reader.Read()) // построчно считываем данные
          {
          Worker selectedWorker = new Worker();
          selectedWorker.Number = ++i;
          selectedWorker.WorkerId = Convert.ToInt32(reader["WorkerId"]);
          selectedWorker.LastName = reader["LastName"].ToString();
          selectedWorker.FirstName = reader["FirstName"].ToString();
          selectedWorker.FIO = selectedWorker.LastName + " " + selectedWorker.FirstName;
          selectedWorker.Phone = reader["Phone"].ToString();
          selectedWorker.BirthDate = Convert.ToDateTime(reader["BirthDate"]);
          selectedWorker.Address = reader["Address"].ToString();
          selectedWorker.Email = reader["Email"].ToString();
          selectedWorker.PositionId = Convert.ToInt32(reader["PositionId"]);
          WorkerList.Add(selectedWorker);
        }
      }
      reader.Close();
    }
```

```
    for (int j = 0; j < WorkerList.Count; j++) {
      WorkerList[j].PositionName = GetPositionName(WorkerList[j].PositionId, positionList);
    }
    return WorkerList;
  }

  public void UpdateWorker(string LastName, string FirstName, string Phone, string Address, string Email, DateTime BirthDate, int PositionId, int WorkerId) {
    using (SqlConnection con = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand("UPDATE Worker SET LastName = @LastName, FirstName = @FirstName, " +
        "Phone = @Phone, Address=@Address, Email=@Email, BirthDate=@BirthDate, PositionId=@PositionId " +
        " WHERE WorkerId = @WorkerId", con)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("@LastName", LastName);
        cmd.Parameters.AddWithValue("@FirstName", FirstName);
        cmd.Parameters.AddWithValue("@Phone", Phone);
        cmd.Parameters.AddWithValue("@Address", Address);
        cmd.Parameters.AddWithValue("@Email", Email);
        cmd.Parameters.AddWithValue("@BirthDate", BirthDate);
        cmd.Parameters.AddWithValue("@PositionId", PositionId);
        cmd.Parameters.AddWithValue("@WorkerId", WorkerId);
        con.Open();
        int rowsAffected = cmd.ExecuteNonQuery();
        con.Close();
      }
    }
  }

  public void DeleteWorkerByWorkerId(int WorkerId) {
    string sqlExpression = "DELETE  FROM Worker WHERE WorkerId=" + WorkerId.ToString();
```

```
      using (SqlConnection connection = new SqlConnection(_ConnString)) {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        command.ExecuteNonQuery();
        connection.Close();
      }
    }

  }

  }

public class Worker {
  private int _Number;
  private int _WorkerId;
  private string _FirstName;
  private string _LastName;
  private string _FIO;
  private string _Phone;
  private string _Address;
  private string _Email;
  private DateTime _BirthDate;
  private int _PositionId;
  private string _PositionName;
  private string _Message;

  public Worker() {
    _Number = 0;
    _WorkerId = 0;
    _FirstName = String.Empty;
    _LastName = String.Empty;
    _FIO = String.Empty;
```

```
  _Phone = String.Empty;
  _Address = String.Empty;
  _Email = String.Empty;
  _BirthDate = new DateTime();
  _PositionId = 0;
  _PositionName = String.Empty;
  _Message = String.Empty;
}

public int Number {
  set { _Number = value; }
  get { return _Number; }
}
public int WorkerId {
  set { _WorkerId = value; }
  get { return _WorkerId; }
}
public string FirstName {
  set { _FirstName = value; }
  get { return _FirstName; }
}
public string LastName {
  set { _LastName = value; }
  get { return _LastName; }
}
public string FIO {
  set { _FIO = value; }
  get { return _FIO; }
}
public string Phone {
  set { _Phone = value; }
  get { return _Phone; }
```

```
  }
  public string Address {
    set { _Address = value; }
    get { return _Address; }
  }
  public string Email {
    set { _Email = value; }
    get { return _Email; }
  }
  public DateTime BirthDate {
    set { _BirthDate = value; }
    get { return _BirthDate; }
  }
  public int PositionId {
    set { _PositionId = value; }
    get { return _PositionId; }
  }
  public string PositionName {
    set { _PositionName = value; }
    get { return _PositionName; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```