Міністерство освіти і науки України

Ніжинський державний університет імені Миколи Гоголя

Факультет природничо-географічних і точних наук

Кафедра географії, туризму та спорту

Галузь знань: 12 Інформаційні технології

Спеціальність: 122 Комп’ютерні науки

**КВАЛІФІКАЦІЙНА РОБОТА**

на здобуття освітнього ступеня магістр

**Тема: «ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАДАЧІ АВТОМАТИЗАЦІЇ РОБОТИ БУДІВЕЛЬНОЇ ОРГАНІЗАЦІЇ»**

Студента Гірса Нікіти Олександровича

Науковий керівник: Фетісов В. С.

кандидат економічних наук, доцент

Рецензенти:

Доктор технічних наук, професор І. В. Казачков.

доцент, кандидат технічних наук Т. А. Кресан

Допущено до захисту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Завідувач кафедри \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Ніжин – 2024**

**АНОТАЦІЯ**

Метою кваліфікаційної роботи є розробка програмного забезпечення для автоматизації бізнес-процесів будівельної організації, включаючи управління проєктами, ресурсами, документообігом і моніторингом. Об’єктом дослідження є процес автоматизації роботи будівельної організації.
Предметом дослідження є програмна реалізація автоматизованої системи управління бізнес-процесами.

Магістерська робота складається зі вступу, основної частини, яка включає три розділи, висновків, списку використаних джерел.

У першому розділі проведено аналіз сучасного стану автоматизації будівельної галузі, визначено основні проблеми та потреби підприємств у впровадженні програмних рішень. Розглянуто існуючі системи автоматизації та їх обмеження. У другому розділі описано технічні аспекти розробки програми, вибір технологій і методів, архітектуру системи, а також функціонал ключових модулів. Особлива увага приділена структурованій реалізації коду та забезпеченню інтеграції з базою даних. У третьому розділі представлено результати тестування розробленої системи, аналіз її продуктивності та практичну значущість. Розглянуто можливі напрями розвитку програми для покращення продуктивності та масштабованості.

За результатами роботи зроблено висновки щодо ефективності автоматизації для оптимізації роботи будівельних компаній. Запропонована система дозволяє знизити витрати, підвищити точність виконання завдань і покращити прозорість процесів.

Ключові слова: автоматизація, програмне забезпечення, управління проектами, документообіг, будівельна організація.

**ANNOTATION**

The purpose of the qualification work is to develop software for automating the business processes of a construction organization, including project management, resource accounting, document flow, and monitoring. The object of the study is the process of automating the work of a construction organization. The subject of the study is the software implementation of an automated business process management system.

The master's thesis consists of an introduction, the main part, which includes three sections, conclusions, and a list of sources used.

The first section analyzes the current state of automation in the construction industry, identifies the main problems and needs of enterprises in implementing software solutions, and reviews existing automation systems and their limitations. The second section describes the technical aspects of software development, the choice of technologies and methods, system architecture, and the functionality of key modules. Particular attention is paid to the structured implementation of code and ensuring database integration. The third section presents the results of testing the developed system, analyzes its performance, and evaluates its practical significance. Possible directions for program development to improve performance and scalability are discussed.

Based on the results, conclusions were made regarding the effectiveness of automation for optimizing the work of construction companies. The proposed system allows reducing costs, increasing the accuracy of task execution, and improving process transparency.

Keywords: automation, software, project management, document flow, construction organization.

Зміст

[**ВСТУП** 5](#_Toc183984776)

[**РОЗДІЛ 1. АНАЛІЗ ДІЯЛЬНОСТІ БУДІВЕЛЬНОЇ ОРГАНІЗАЦІЇ ТА ПОТРЕБИ В АВТОМАТИЗАЦІЇ** 7](#_Toc183984777)

[**1.1.** **Огляд діяльності будівельної організації** 7](#_Toc183984778)

[**1.2. Визначення проблем, які можуть бути вирішені автоматизацією** 10](#_Toc183984779)

[**1.3. Роль автоматизації в підвищенні ефективності роботи будівельних організацій** 13](#_Toc183984780)

[**Висновки до розділу 1** 17](#_Toc183984781)

[**РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ І МЕТОДІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ** 18](#_Toc183984782)

[**2.1.** **Визначення вимог до програмного забезпечення** 18](#_Toc183984783)

[**2.2.** **Огляд можливих технологій для розробки** 20](#_Toc183984784)

[**2.3. Опис методології розробки** 21](#_Toc183984785)

[**2.4. Архітектура програмного забезпечення** 23](#_Toc183984786)

[**Висновки до розділу 2** 25](#_Toc183984787)

[**РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ І ФУНКЦІОНАЛУ СИСТЕМИ** 26](#_Toc183984788)

[**3.1. Загальний огляд програми** 26](#_Toc183984789)

[**3.2. Опис функціоналу** 29](#_Toc183984790)

[**3.3. Технічні аспекти реалізації** 31](#_Toc183984791)

[**3.4. Тестування та результати** 34](#_Toc183984792)

[**3.5. Потенційні напрями розвитку програми** 36](#_Toc183984793)

[**Висновки до розділу 3** 38](#_Toc183984794)

[**ВИСНОВКИ** 40](#_Toc183984795)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 42](#_Toc183984796)

[**ДОДАТКИ** 44](#_Toc183984797)

# **ВСТУП**

Актуальність теми. Будівельна галузь є однією з ключових сфер економіки, яка має значний вплив на розвиток інфраструктури та соціально-економічний стан країни. В умовах сучасного ринку зростають вимоги до якості будівельних робіт, дотримання строків та раціонального використання ресурсів. Для будівельних компаній це створює потребу в удосконаленні бізнес-процесів та впровадженні ефективних методів управління.

Одним із найбільш перспективних напрямків розвитку будівельної галузі є автоматизація ключових процесів управління. Інтеграція автоматизованих систем дозволяє значно покращити продуктивність праці, знизити витрати, мінімізувати помилки та забезпечити прозорість діяльності. Автоматизація управління проєктами, документообігом, фінансами та логістикою відкриває можливості для підвищення конкурентоспроможності та довгострокової стабільності будівельних організацій.

Проте процес впровадження автоматизованих систем стикається з певними труднощами. Серед них – брак фінансових ресурсів, недостатнє розуміння переваг автоматизації, а також потреба в адаптації рішень до специфіки діяльності компаній. Це робить дослідження в цій галузі особливо актуальними.

Об’єктом дослідження є процес автоматизації роботи будівельної організації.

Предметом дослідження є програмна реалізація автоматизованої системи управління ключовими бізнес-процесами.

Метою дипломної роботи є розробка програмного забезпечення, яке дозволить оптимізувати управління будівельними проєктами, ресурсами та документацією, підвищуючи ефективність діяльності будівельної організації.

Завдання дослідження

Для досягнення поставленої мети в дипломній роботі вирішуються наступні завдання:

* аналіз сучасних програмних рішень для автоматизації роботи будівельних компаній;
* визначення основних проблем і потреб, які виникають у процесі автоматизації;
* проектування архітектури програмного забезпечення;
* розробка системи для автоматизації управління проєктами, ресурсами та документообігом;
* тестування програмного забезпечення у реальних умовах;
* оцінка ефективності розробленого рішення.

Наукова новизна

Наукова новизна роботи полягає у створенні програмного продукту, який інтегрує інструменти для управління будівельними об’єктами, документами та ресурсами. Особлива увага приділяється адаптації рішення до специфіки діяльності будівельних компаній, що дозволяє мінімізувати витрати на впровадження автоматизації.

Практичне значення

Практична цінність роботи полягає у створенні ефективного інструменту, який дозволяє автоматизувати основні бізнес-процеси будівельної організації. Результати дослідження можуть бути використані для впровадження подібних рішень у малих та середніх будівельних компаніях, що дозволить підвищити їх конкурентоспроможність у сучасних ринкових умовах.

Структура роботи

Робота складається з аналізу проблем автоматизації будівельних процесів, опису архітектури розробленої системи, її реалізації та тестування. У заключній частині розглядаються результати впровадження програмного забезпечення та пропозиції щодо подальшого розвитку.

Апробація результатів дослідження. Опубліковано статтю «ІНОВАЦІЙНІ ПІДХОДИ ДО ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ЗАДАЧ АВТОМАТИЗАЦІЇ РОБОТИ БУДІВЕЛЬНОЇ ОРГАНІЗАЦІЇ» у Віснику студентського наукового товариства випуск 30 [13]. Хід дослідження висвітлювався у доповіді на Х міжнародній науково-практичній інтернет-конференції «Наука і молодь у ХХІ сторіччі». (Полтава, Україна, 28 листопада, 2024 р.) [14].

# **РОЗДІЛ 1. АНАЛІЗ ДІЯЛЬНОСТІ БУДІВЕЛЬНОЇ ОРГАНІЗАЦІЇ ТА ПОТРЕБИ В АВТОМАТИЗАЦІЇ**

* 1. **Огляд діяльності будівельної організації**

Будівельні організації займають важливе місце в економіці, оскільки забезпечують зведення нових об’єктів інфраструктури, житлового та комерційного будівництва, а також промислових споруд. Їх діяльність зазвичай характеризується високою складністю, потребою в узгодженні багатьох процесів та управлінні ресурсами. З огляду на специфіку роботи, будівельні компанії стикаються з низкою організаційних, управлінських та технічних викликів, які потребують ефективного управління для мінімізації витрат, скорочення термінів будівництва та підвищення якості виконання робіт.

Структура управління

Загальна структура управління будівельної організації, як правило, включає наступні підрозділи:

* Відділ проєктування – відповідає за створення проєктів і планів будівництва. В його обов’язки входить аналіз архітектурних та інженерних рішень, підготовка креслень та технічної документації, а також оцінка економічної доцільності проєкту.
* Відділ закупівель та постачання – займається забезпеченням будівельного процесу необхідними матеріалами, обладнанням та інструментами. Цей відділ часто працює з постачальниками, планує обсяги матеріалів, контролює поставки та оптимізує витрати на закупівлю.
* Виробничий відділ – основна ланка, що безпосередньо здійснює будівельні роботи. Він контролює всі етапи зведення об’єктів, організовує роботу будівельних бригад, виконує технічний нагляд і забезпечує відповідність якості виконання робіт.
* Відділ охорони праці та техніки безпеки – відповідальний за дотримання вимог з охорони праці, техніки безпеки та екологічних стандартів. Відділ здійснює контроль за умовами праці, забезпечує персонал захисними засобами та проводить інструктажі.
* Фінансовий відділ – займається фінансовим плануванням та контролем. Включає підрозділи для управління бюджетом, контролю витрат, бухгалтерії та звітності.
* Відділ управління проєктами – відповідає за координацію всіх робочих процесів на кожному етапі будівництва, визначає пріоритети та відповідає за дотримання графіків і термінів.

Такі підрозділи допомагають розділити обов’язки та підвищити ефективність, проте їхня робота вимагає постійної координації та злагодженого обміну інформацією. Ефективна взаємодія між відділами є важливою умовою для успішного виконання проєкту, особливо в умовах великих обсягів будівельних робіт.

Основні бізнес-процеси

Будівельні організації виконують безліч бізнес-процесів, які можна поділити на кілька ключових категорій:

* Планування та підготовка проєкту – включає проєктування, складання бюджету, підготовку технічної документації та визначення матеріально-технічних потреб. На цьому етапі часто залучаються архітектори, інженери та фахівці з кошторису.
* Управління ресурсами та матеріалами – завдання включають закупівлю матеріалів, інструментів, техніки, а також контроль за їх використанням. На цьому етапі важливо забезпечити своєчасне надходження ресурсів, щоб уникнути простоїв та перевитрат.
* Розподіл і контроль роботи персоналу – включає організацію роботи будівельних бригад, складання графіків роботи, розподіл завдань та моніторинг виконання робіт.
* Технічний нагляд і контроль якості – здійснюється на кожному етапі будівництва для забезпечення відповідності робіт технічним нормам та стандартам якості.
* Фінансове планування та облік витрат – відстеження витрат та їх співвідношення з бюджетом, управління заробітною платою працівників та складання фінансової звітності.
* Документообіг та звітність – ведення технічної, фінансової та юридичної документації, складання звітів та підготовка документів для передання об’єкта замовникові.

Кожен з цих процесів є критичним для ефективної роботи організації, однак без належної автоматизації вони можуть стати джерелом помилок, затримок та неефективності. У будівельній галузі надзвичайно важливою є точність даних та швидкість обробки інформації, оскільки навіть незначні затримки або неточності в управлінні ресурсами можуть суттєво збільшити витрати або затягнути виконання проєкту.

Потенційні точки автоматизації

Автоматизація в будівельних організаціях допомагає підвищити ефективність роботи, забезпечити злагоджену взаємодію між підрозділами та знизити ризик помилок. Основні точки автоматизації в будівельній організації включають:

* Управління ресурсами та логістикою – автоматизація процесів закупівлі матеріалів, інвентаризації ресурсів та логістики дозволяє зменшити витрати на закупівлю та уникнути простоїв, пов'язаних з дефіцитом матеріалів.
* Управління персоналом – автоматизація розподілу задач, моніторинг часу роботи та контролю ефективності працівників значно спрощує управління робочою силою.
* Контроль якості та технічний огляд – автоматизовані системи допомагають зберігати записи про технічні параметри, ведення журналів якості та контролювати відповідність будівельних норм.
* Документообіг – автоматизована система документообігу спрощує створення, зберігання, доступ і обмін важливою документацією між відділами, що дозволяє скоротити час на оформлення та узгодження.
* Фінансовий облік та звітність – автоматизація бухгалтерських процесів і фінансового обліку забезпечує точне ведення витрат, зменшує ймовірність помилок у розрахунках та полегшує складання звітності.

Загалом автоматизація у будівельній організації спрямована на підвищення ефективності роботи, зниження ризику помилок, забезпечення швидкості обробки даних і поліпшення контролю за кожним етапом проєкту. Використання сучасних програмних рішень дозволяє більш ефективно координувати взаємодію між відділами та контролювати всі аспекти будівельного процесу, забезпечуючи високу якість та своєчасність виконання робіт.

## **1.2. Визначення проблем, які можуть бути вирішені автоматизацією**

Будівельні організації стикаються з численними проблемами, обумовленими складністю проектів, високими вимогами до точності та ефективності, а також необхідністю обробки великого обсягу інформації. У сучасних умовах успішне функціонування будівельної компанії значною мірою залежить від рівня автоматизації її бізнес-процесів. Основними проблемами, що потребують автоматизованих рішень, є: трудомісткість ручних процесів, висока ймовірність помилок при обробці даних, затримки в ухваленні рішень через тривалий документообіг та недостатня ефективність комунікацій між підрозділами.

Трудомісткість ручних процесів та їх обмеження

У багатьох будівельних організаціях велика частина роботи з документами, обліком ресурсів, складанням звітності та плануванням залишається ручною. Використання паперової документації та традиційних таблиць призводить до збільшення витрат часу на обробку даних, дублювання інформації та її втрати. Наприклад, значний час витрачається на:

* пошук і перегляд документів;
* копіювання та розподіл документів між відділами;
* оновлення інформації у звітах і графіках робіт.

Ці процеси не лише займають значний обсяг робочого часу співробітників, але й потребують чіткої координації між відділами. При цьому людський фактор ускладнює гарантування точності і своєчасності роботи з документами, що може призводити до пропущених термінів, затримок у виконанні проектів та додаткових витрат. Автоматизація дозволяє значно зменшити ці ризики, забезпечуючи швидкий доступ до актуальної інформації, мінімізуючи дублювання процесів і звільняючи співробітників від трудомістких повторюваних завдань.

Висока ймовірність помилок при обробці даних

Обробка великих обсягів інформації вручну створює ризики помилок через людський фактор. У будівництві, де точність є критично важливою, навіть невелика помилка може призвести до серйозних наслідків, включаючи перевищення бюджету, збої в термінах та порушення стандартів якості. Наприклад, некоректне ведення обліку матеріалів може призвести до неправильного прогнозування витрат, а помилки в кошторисах можуть викликати затримки через необхідність перегляду бюджету або додаткових закупівель.

Автоматизація обробки даних і документообігу забезпечує високу точність у підрахунках, дозволяє уникати повторних помилок за рахунок централізованої системи контролю і скорочує ймовірність виникнення неточностей. Використання цифрових інструментів дозволяє налаштовувати вбудовані перевірки, системи нагадувань, а також зв'язок між відділами для швидкого виправлення можливих помилок.

Затримки у прийнятті рішень через тривалий документообіг

Процеси документообігу в будівельних організаціях часто є складними та багатоступеневими, адже для ухвалення рішень потрібно отримувати погодження від керівників різних підрозділів. Це призводить до затримок в управлінні проєктами, оскільки документи проходять кілька етапів погодження, особливо коли документообіг відбувається у паперовому форматі або за допомогою електронної пошти без системного обліку. Наприклад, оформлення договорів із постачальниками, розрахунки із субпідрядниками або затвердження змін у проєкті можуть займати більше часу, ніж необхідно, що створює затримки на всіх етапах будівництва.

Автоматизована система документообігу забезпечує зручне і швидке погодження документів через цифрову платформу, де кожен користувач може миттєво отримувати документи для перегляду, коментування або підпису. Це не тільки скорочує час на документообіг, але й робить процес більш прозорим і контрольованим, оскільки відстежується кожен етап погодження та всі зміни фіксуються в системі.

Недостатня ефективність комунікацій між підрозділами

Однією з найбільших проблем для будівельних компаній є розірваність зв’язків між різними підрозділами, особливо у випадках, коли компанія одночасно веде кілька проектів. Неузгодженість дій між проєктним відділом, закупівлями, будівельними підрозділами і бухгалтерією може призводити до затримок, перевищення бюджету та неефективного використання ресурсів. Наприклад, відсутність оперативного зв'язку між відділом закупівель та будівельною бригадою може спричинити простої на майданчику через нестачу матеріалів.

Автоматизація дозволяє створити єдину інформаційну систему, де кожен підрозділ має доступ до актуальної інформації щодо проєкту, змін і потреб, що дає можливість швидше вирішувати проблеми та приймати рішення. Така система дозволяє всім учасникам бачити дані у режимі реального часу, отримувати сповіщення про важливі зміни та миттєво взаємодіяти один з одним.

Низька прозорість та контроль над витратами

Фінансове управління є критично важливим для будівельних організацій, оскільки проєкти потребують значних інвестицій у ресурси, обладнання, оплату праці та послуги субпідрядників. Традиційні підходи до обліку витрат часто призводять до непрозорості, що ускладнює контроль бюджету та виявлення причин перевитрат. Відсутність автоматизованої системи фінансового обліку ускладнює відстеження всіх витрат, оскільки деякі операції можуть бути не внесені до системи або оброблятися із затримкою, що перешкоджає правильному плануванню бюджету.

Автоматизована система обліку дозволяє будівельним компаніям більш точно контролювати витрати, що сприяє виявленню причин перевитрат і ефективнішому управлінню ресурсами. Така система забезпечує миттєвий доступ до фінансових даних і дозволяє прогнозувати фінансовий стан кожного проекту на підставі актуальної інформації.

Складність управління великими обсягами даних

Будівельні проєкти створюють великий обсяг даних, зокрема фінансових звітів, кошторисів, графіків робіт, документів щодо обліку матеріалів та устаткування. Управління цими даними вручну або з використанням базових електронних таблиць є надзвичайно трудомістким і неефективним, оскільки велика кількість різних файлів ускладнює пошук інформації та створює ризик її втрати.

Використання систем управління даними дозволяє автоматизувати процеси збору, зберігання і аналізу даних, що сприяє кращому обміну інформацією між співробітниками, швидшому прийняттю рішень та забезпеченню доступу до потрібних даних у будь-який момент. Централізовані системи даних дозволяють компанії зберігати всі документи в одному місці, де вони доступні для перегляду, що забезпечує стабільність і надійність інформації.

## **1.3. Роль автоматизації в підвищенні ефективності роботи будівельних організацій**

Автоматизація є критично важливим елементом, який дозволяє будівельним організаціям покращувати продуктивність, скорочувати витрати та знижувати ймовірність помилок у своїй діяльності. Завдяки автоматизації ключових бізнес-процесів, таких як управління проєктами, документообіг, логістика та фінансовий контроль, будівельні організації можуть більш ефективно використовувати ресурси, мінімізувати простої і забезпечити вищу якість виконання проєктів. Розглянемо, як автоматизація сприяє покращенню роботи кожного з основних напрямів діяльності будівельної організації.

Поліпшення управління проєктами

Управління проєктами в будівельній галузі є надзвичайно складним завданням, яке вимагає точного планування, моніторингу виконання робіт і ефективного управління ресурсами. Застосування автоматизованих систем управління проєктами дозволяє значно полегшити цей процес. За допомогою таких систем можна:

* створювати та відстежувати графіки виконання робіт, що дає змогу краще контролювати терміни;
* координувати роботу різних підрозділів, забезпечуючи прозорість процесу виконання проєктів;
* своєчасно реагувати на відхилення від початкового плану завдяки можливості відстежувати зміни в режимі реального часу.

Крім того, автоматизовані системи управління проєктами можуть автоматично створювати звіти про виконання робіт, що дає можливість керівництву швидко отримувати актуальну інформацію про стан проєкту. Це дозволяє приймати обґрунтовані рішення, які мінімізують ризики та забезпечують своєчасне виконання всіх завдань.

Оптимізація документообігу

У більшості будівельних організацій документообіг є ключовим елементом, оскільки включає численні документи: від договорів і кошторисів до актів виконаних робіт і звітів. Традиційний документообіг, що базується на паперових носіях або неструктурованих електронних документах, має низку недоліків, включаючи тривалі затримки в узгодженні документів, втрату інформації та дублювання даних. Автоматизація документообігу дозволяє:

* забезпечити централізоване зберігання документів, що дозволяє швидко знаходити необхідну інформацію;
* підвищити прозорість процесу погодження документів та пришвидшити його завдяки цифровому підпису та системі оповіщень;
* забезпечити контроль над версіями документів і відслідковувати всі зміни, що дозволяє уникати помилок через використання застарілих даних.

Автоматизований документообіг робить процеси управління даними більш ефективними та зручними, що дозволяє мінімізувати затримки, знизити витрати на обробку документів і зменшити ймовірність виникнення помилок.

Підвищення ефективності управління ресурсами та логістики

Логістика і управління ресурсами в будівництві є критично важливими для забезпечення безперервності роботи на об’єктах. Нестача матеріалів або затримка постачання обладнання можуть спричинити значні втрати через простої та зрив термінів виконання робіт. Автоматизація управління ресурсами дозволяє:

* точно планувати потребу в матеріалах і техніці відповідно до графіку будівельних робіт;
* контролювати залишки матеріалів на складах, своєчасно плануючи їх поповнення;
* ефективно розподіляти робочу силу та техніку між об’єктами.

Крім того, автоматизована система управління логістикою забезпечує відстеження руху матеріалів і ресурсів у реальному часі, що дозволяє уникати надлишкових запасів і забезпечувати їх наявність у потрібний момент. Таким чином, будівельна компанія може значно скоротити витрати на утримання складів, уникати простоїв і забезпечувати дотримання графіків робіт.

Підвищення прозорості та контролю над фінансами

Фінансове управління в будівельній організації включає контроль витрат, формування кошторисів, управління бюджетами проєктів та складання фінансової звітності. Автоматизація фінансового обліку дозволяє більш точно контролювати всі витрати, оперативно отримувати дані про фінансовий стан кожного проєкту та прогнозувати подальші витрати. Зокрема, автоматизація допомагає:

* вести детальний облік витрат за кожним проєктом, що дозволяє виявляти можливі перевитрати і швидко реагувати на них;
* прогнозувати витрати на підставі даних про поточні і завершені проєкти, що сприяє точнішому плануванню бюджету;
* формувати фінансові звіти для керівництва та замовників у режимі реального часу, підвищуючи прозорість фінансових операцій.

Системи фінансової автоматизації також дозволяють вести облік взаєморозрахунків із субпідрядниками та постачальниками, що є важливим аспектом для забезпечення своєчасних розрахунків і уникнення штрафних санкцій за затримки платежів.

Підвищення продуктивності працівників і зниження навантаження на персонал

Ручні процеси, що включають обробку документів, звітів і даних, займають значний обсяг часу, який співробітники могли б використовувати на інші, більш продуктивні завдання. Автоматизація рутинних операцій дозволяє знизити навантаження на персонал, вивільнивши його для більш складних і творчих задач, таких як аналіз проєктних даних або прийняття стратегічних рішень. Зокрема, автоматизація забезпечує:

* швидке і точне виконання завдань, що знижує ймовірність помилок і потребу в додатковій перевірці результатів;
* можливість обробки великої кількості інформації без додаткових витрат часу;
* підвищення загальної продуктивності завдяки скороченню часу на повторювані операції.

Це не тільки підвищує задоволеність працівників і знижує рівень стресу, але й дозволяє компанії ефективніше використовувати свої трудові ресурси.

Забезпечення безперервного моніторингу та підвищення якості виконання проєктів

Будівельні проєкти потребують постійного моніторингу якості, щоб відповідати технічним вимогам, стандартам і термінам виконання. Автоматизовані системи дозволяють створювати механізми для постійного контролю якості на всіх етапах робіт:

* проводити інспекції якості у режимі реального часу;
* автоматично генерувати звіти та сповіщення про можливі відхилення;
* документувати всі процеси контролю, що сприяє підвищенню відповідальності та знижує ризик виникнення дефектів.

Завдяки цьому забезпечується висока якість робіт та дотримання встановлених стандартів на кожному етапі проєкту, що сприяє зміцненню репутації компанії і підвищенню довіри з боку замовників.

Зниження витрат та оптимізація використання ресурсів

Автоматизація дозволяє будівельним організаціям оптимізувати використання своїх ресурсів, що, в свою чергу, знижує загальні витрати. Наприклад, автоматизоване планування дозволяє знизити надлишкові запаси матеріалів на складах, які можуть виявитися непотрібними на поточний момент, або більш точно розподіляти робочу силу між проєктами. Оптимізація витрат дозволяє компанії підвищувати свою конкурентоспроможність на ринку, оскільки це забезпечує виконання проєктів в рамках бюджету і зі зниженою вартістю для клієнтів.

## **Висновки до розділу 1**

У цьому розділі було детально розглянуто структуру управління будівельною організацією та її основні бізнес-процеси. Побудова ефективної організаційної структури вимагає чіткої координації між підрозділами, що здійснюють проєктування, закупівлю матеріалів, виробництво, фінансовий облік та управління проектами. Ці процеси забезпечують виконання будівельних робіт у відповідності до графіків та вимог, проте їхня ефективність часто знижена через відсутність достатньої автоматизації.

Узгоджена робота підрозділів є важливою умовою для успішного завершення будівельних проектів. Однак, навіть при наявності структурованої організації, традиційні методи управління, зокрема ручне оброблення даних та документообіг, часто призводять до помилок, затримок та додаткових витрат. Зважаючи на ці виклики, автоматизація процесів, таких як управління ресурсами, персоналом, фінансами та документообігом, стає необхідністю для підвищення ефективності будівельних організацій.

Автоматизація дозволяє значно покращити координацію між підрозділами, зменшити кількість помилок, забезпечити прозорість фінансових витрат та прискорити документообіг, що в кінцевому підсумку веде до скорочення термінів будівництва, зниження витрат і поліпшення якості виконаних робіт. У наступному розділі буде розглянуто конкретні проблеми, які можуть бути вирішені через впровадження автоматизованих систем в діяльність будівельних організацій.

# **РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ І МЕТОДІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

1. **Визначення вимог до програмного забезпечення**

Визначення вимог є одним із ключових етапів розробки програмного забезпечення, оскільки саме на цьому етапі формується бачення майбутньої системи, яке повинно відповідати як очікуванням користувачів, так і технічним можливостям.

Програмне забезпечення повинно забезпечувати функціональність для роботи з обліковими записами користувачів різних ролей, підтримувати доступ до бази даних і забезпечувати виконання завдань у межах, визначених ролями. Наприклад, HR-менеджери повинні мати можливість переглядати, редагувати і додавати інформацію про вакансії та співробітників. У той же час працівники складу зможуть працювати лише з даними про наявність товарів і їхній рух між об’єктами. Важливим аспектом є також інтеграція з існуючими бізнес-процесами, що дозволить уникнути значних змін у поточній роботі підприємства.

Загалом, вимоги можна розділити на дві основні категорії: функціональні та нефункціональні.

Функціональні вимоги

Ці вимоги визначають, які задачі повинна виконувати система. У нашому випадку програмне забезпечення має:

* + забезпечувати можливість аутентифікації користувачів за логіном і паролем, враховуючи різні ролі;
	+ надавати кожному користувачу доступ тільки до тих даних і функцій, які відповідають його ролі;
	+ забезпечувати взаємодію з базою даних для перегляду, додавання, редагування і видалення записів;
	+ підтримувати створення звітів і проведення аналітики на основі інформації, що зберігається в базі.

Також важливою є реалізація автоматизації рутинних операцій, таких як оновлення статусу об’єктів чи синхронізація змін у таблицях бази даних.

Нефункціональні вимоги

Окрім того, як система має працювати, важливими є вимоги до її якості. Вони стосуються продуктивності, надійності, безпеки, масштабованості та зручності використання.

Система повинна працювати швидко, навіть за високого навантаження, із часом відповіді не більше двох секунд для більшості операцій. Вона має бути надійною, тобто захищеною від втрат даних або збоїв у роботі. Надійність досягається завдяки резервуванню бази даних і впровадженню механізмів автоматичного відновлення після збою.

Безпека також є критично важливою складовою. Доступ до системи має бути захищеним, із використанням шифрованих з’єднань, а система повинна бути захищена від поширених атак, таких як SQL-ін’єкції. Усі ці аспекти забезпечать користувачам впевненість у стабільній і безпечній роботі.

Користувацький інтерфейс повинен бути зрозумілим і зручним. Навіть ті користувачі, які не мають великого досвіду роботи з програмами, повинні легко орієнтуватися в системі. Адаптація інтерфейсу для різних типів пристроїв (настільні комп’ютери, планшети, смартфони) сприятиме підвищенню продуктивності та задоволеності користувачів.

Проблеми інтеграції з існуючими системами

Одним із важливих аспектів, які слід врахувати під час вибору технологій, є проблема інтеграції нового програмного забезпечення з уже існуючими системами та процесами на підприємстві.

Крім того, потрібно врахувати поточні бізнес-процеси. Наприклад, автоматизація оновлення даних може потребувати змін у звичних для співробітників сценаріях роботи. Це викликає необхідність у навчанні персоналу та поступовій адаптації до нового програмного забезпечення.

Ще одним викликом є забезпечення сумісності з іншими програмними продуктами, які вже використовуються на підприємстві, наприклад, з ERP-системами або спеціалізованими інструментами для обліку. Система має коректно взаємодіяти з цими продуктами, щоб забезпечити безперервність бізнес-процесів.

Отже, успішний вибір технологій і методів розробки вимагає глибокого аналізу вимог, продуманого підходу до інтеграції і врахування специфіки роботи підприємства. Це створить основу для розробки надійного, зручного і функціонального продукту.

1. **Огляд можливих технологій для розробки**

Ефективна розробка програмного забезпечення значною мірою залежить від правильного вибору технологій. Це стосується як інструментів для створення коду, так і систем для зберігання й обробки даних. Основною метою є забезпечення балансу між продуктивністю, масштабованістю та простотою підтримки майбутньої системи.

Вибір мови програмування, фреймворків і бібліотек

Одним із перших рішень є вибір мови програмування. Залежно від поставлених задач, варіанти можуть включати Java, Python, C#, JavaScript або інші популярні мови. У цьому проєкті було вирішено використовувати Java, що зумовлено її універсальністю, стабільністю та широкими можливостями для інтеграції з базами даних. Крім того, Java добре підтримується і має велику кількість фреймворків, таких як Spring, які значно спрощують розробку складних систем.

Фреймворки та бібліотеки допомагають зменшити час розробки, пропонуючи готові модулі для розв’язання поширених задач. Наприклад, Swing у Java використовується для створення графічного інтерфейсу, який є зручним і зрозумілим навіть для користувачів без технічного досвіду. Для роботи з базою даних обрано бібліотеку JDBC (Java Database Connectivity), яка забезпечує ефективну взаємодію між програмним кодом і реляційною базою даних.

Крім того, для безпеки та перевірки даних будуть використані додаткові бібліотеки, які допомагають захистити систему від SQL-ін’єкцій, забезпечити шифрування паролів і обмежити доступ до критичних частин системи.

Вибір бази даних, інструментів для зберігання та обробки даних

Іншим важливим аспектом є вибір бази даних, яка має відповідати обсягу даних і типу операцій, що виконуються. У даному випадку використовується MySQL, оскільки це популярна, надійна і легка в адмініструванні система управління реляційними базами даних. MySQL має широкий набір інструментів для налаштування та резервного копіювання, що робить її ідеальним вибором для таких проєктів.

Для взаємодії з базою даних розроблено структуру, яка включає таблиці для обліку користувачів, вакансій, складів і будівельників. Такий підхід дозволяє організувати дані у вигляді зрозумілих і логічних зв’язків між різними сутностями. Крім того, база підтримує тригери, які автоматично виконують необхідні дії, наприклад, додають або видаляють записи при зміні статусу об’єкта.

Для зберігання великих обсягів даних (у разі масштабування проєкту) можливо розглянути варіант використання NoSQL-баз, таких як MongoDB або Cassandra, які ефективні для зберігання неструктурованих даних. Проте в рамках поточного проєкту ці системи не є пріоритетними через специфіку роботи з чітко визначеними реляційними структурами.

Завдяки поєднанню сучасних мов програмування, перевірених фреймворків і потужних систем управління базами даних, можливо створити надійне програмне забезпечення, яке задовольнить усі вимоги замовника та кінцевих користувачів.

## **2.3. Опис методології розробки**

Методологія розробки програмного забезпечення визначає підхід до організації робочих процесів і виконання основних етапів створення програми. У цьому проєкті для розробки було вибрано класичний підхід водоспаду (Waterfall), який передбачає чітку послідовність етапів, кожен з яких завершується перед початком наступного. Однак для гнучкості в розробці в окремих випадках може застосовуватись елемент Agile для адаптації до змінних вимог.

Аналіз вимог і планування

Першим етапом є збирання вимог до програмного забезпечення, яке включає в себе спілкування з замовником та кінцевими користувачами. На цьому етапі визначаються як функціональні, так і нефункціональні вимоги, що визначають, що саме система повинна робити і як вона повинна працювати. Важливим є також оцінка ризиків та створення чіткої документації для подальшої роботи.

Проектування системи

На основі зібраних вимог розробляється архітектура програмного забезпечення. Це включає проектування бази даних, визначення основних компонентів і їх взаємодії, а також розробку інтерфейсу користувача. Архітектура системи повинна бути такою, щоб забезпечити масштабованість, надійність і безпеку.

Розробка

На етапі розробки починається безпосереднє написання коду. Всі компоненти програми створюються на основі проекту. Це включає реалізацію серверної частини, розробку інтерфейсу користувача та інтеграцію з базою даних. Окрім цього, на цьому етапі застосовуються автоматизовані інструменти тестування для виявлення помилок ще на стадії розробки.

Тестування

Тестування системи є важливою частиною розробки, що допомагає виявити помилки до впровадження програмного забезпечення. Включає різні типи тестів: юніт-тести, інтеграційні тести, тестування інтерфейсу та безпеки. Тестування дозволяє переконатися, що програма відповідає вимогам і працює стабільно у реальних умовах.

Впровадження

Після успішного тестування програмне забезпечення впроваджується на робочих серверах. Цей етап також включає налаштування середовища, навчання користувачів та проведення остаточного тестування у реальних умовах. Важливо забезпечити правильну інтеграцію з існуючими системами.

Підтримка та оновлення

Після впровадження програми важливо забезпечити постійну підтримку. Це включає виправлення помилок, оптимізацію роботи програми та оновлення функцій згідно з новими вимогами користувачів. Підтримка забезпечує стабільну роботу програми протягом її життєвого циклу.

Методологія водоспаду надає чітке і структуроване управління проєктом, а гнучкість підходу дає змогу оперативно реагувати на зміни під час розробки. Кожен етап логічно переходить у наступний, забезпечуючи послідовність і контроль якості на всіх етапах розробки.

## **2.4. Архітектура програмного забезпечення**

Архітектура програмного забезпечення визначає структуру і взаємодію основних компонентів системи, а також спосіб організації зберігання і обробки даних. У цьому проєкті архітектура охоплює як логічну, так і фізичну структуру, і визначає, як програма буде працювати на реальному обладнанні, а також як компоненти будуть взаємодіяти між собою.

Логічна архітектура

Логічна архітектура описує основні компоненти системи та їх взаємодію. Вона розділяється на кілька рівнів, кожен з яких виконує певну роль у забезпеченні роботи системи.

1. Клієнтська частина (UI): Це інтерфейс, через який користувачі взаємодіють із системою. Кожна роль користувача (HR, Builder, Storage) має доступ до свого власного інтерфейсу, який реалізований за допомогою Java Swing. Інтерфейс є графічним і дозволяє зручно переглядати, додавати або редагувати інформацію. Користувач взаємодіє з програмою через елементи інтерфейсу, які відправляють запити на сервер для обробки.
2. Бізнес-логіка (Business Logic): Цей компонент обробляє основні функції програми, такі як аутентифікація користувачів, перевірка їх прав доступу, обробка запитів на взаємодію з базою даних і обробка даних. Бізнес-логіка визначає основні правила роботи з даними та їх структуру. Вона реалізована на сервері з використанням Java та JDBC, що забезпечує зв'язок між програмою та базою даних.
3. База даних (MySQL): Для зберігання даних про користувачів, об'єкти, склади, будівельників та інші сутності використовується MySQL. База даних зберігає всю важливу інформацію в структурованому вигляді, включаючи таблиці, зв'язки між ними, а також індекси для швидкого доступу. Доступ до бази даних здійснюється через JDBC, що дозволяє бізнес-логіці взаємодіяти з даними.
4. Система безпеки: Окремо варто виділити систему безпеки, яка складається з механізмів аутентифікації і авторизації. Вона гарантує, що лише уповноважені користувачі можуть здійснювати доступ до чутливих даних. За допомогою цієї системи кожен користувач має доступ тільки до тих функцій і даних, які йому дозволено згідно з його роллю.

Фізична архітектура

Фізична архітектура визначає, як компоненти програми реалізуються на реальному обладнанні та як вони взаємодіють між собою. Вона включає сервери, на яких працює бізнес-логіка та база даних, а також клієнтські пристрої, з яких користувачі отримують доступ до системи.

1. Сервер додатків: Основна логіка програми, яка обробляє запити користувачів і взаємодіє з базою даних, працює на сервері додатків. Це може бути фізичний сервер або віртуальна машина, що підтримує запуск програми, обробку запитів і зберігання тимчасових даних.
2. База даних: База даних також розміщується на окремому сервері, що забезпечує її високу доступність і стабільну роботу. Сервер бази даних забезпечує швидкий доступ до даних, а також підтримує транзакційність і резервне копіювання для безпеки інформації.
3. Клієнтські пристрої: Користувачі можуть взаємодіяти з програмою через клієнтські пристрої, такі як персональні комп'ютери, планшети або смартфони. Взаємодія здійснюється через веб-клієнт або локальний додаток, що відправляє запити на сервер і отримує необхідні дані для відображення у вигляді графічного інтерфейсу.

Схема взаємодії компонентів

Взаємодія компонентів системи відбувається через чітко визначену структуру, де кожен компонент виконує свою роль і відповідає за конкретну частину процесу. Клієнтська частина (UI) взаємодіє із сервером додатків, який у свою чергу здійснює запити до бази даних, обробляє їх і повертає результати.


**Рис. 2.1.** Структура взаємодія компонентів системи.

1. Клієнт відправляє запит до сервера додатків, наприклад, на перегляд або зміну даних.
2. Сервер додатків обробляє запит і, при необхідності, відправляє запит до бази даних для отримання або зміни даних.
3. База даних повертає результат на сервер, де він обробляється і відправляється назад до клієнта для відображення.

## **Висновки до розділу 2**

У другому розділі розглянуто ключові аспекти вибору технологій та методів для розробки програмного забезпечення. Детальний аналіз вимог до системи дозволив визначити функціональні та нефункціональні характеристики, необхідні для забезпечення її ефективної роботи. Було обрано мову програмування Java, яка забезпечує стабільність, масштабованість і широкий вибір інструментів для реалізації бізнес-логіки, інтерфейсу користувача та взаємодії з базою даних.

Для зберігання та обробки даних використовується реляційна база даних MySQL, яка відповідає сучасним вимогам продуктивності, надійності та легкості в адмініструванні. Інтеграція бізнес-логіки з базою даних здійснюється за допомогою JDBC, що забезпечує гнучкість і швидкість виконання запитів.

Ретельно продуманий вибір інструментів дозволив створити архітектуру, яка забезпечує розмежування функцій, захист даних, а також простоту майбутнього масштабування. Такий підхід гарантує ефективну роботу програми як в умовах базового використання, так і при розширенні функціоналу в майбутньому.

# **РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ І ФУНКЦІОНАЛУ СИСТЕМИ**

## **3.1. Загальний огляд програми**

Система створена для автоматизації процесів управління персоналом, об’єктами будівництва та складськими ресурсами. Її головна мета — оптимізувати обробку даних, мінімізувати рутинну роботу користувачів і створити зручну платформу для ефективного контролю за ключовими процесами підприємства.

Програма підтримує розмежування доступу між користувачами залежно від їхньої ролі, що забезпечує зручність у використанні та підвищує безпеку системи. Наприклад, HR-менеджери отримують доступ до функцій управління співробітниками, тоді як складські працівники працюють лише з даними про матеріали та запаси.

Графічний інтерфейс програми розроблено з використанням Java Swing. Він має зручну, інтуїтивно зрозумілу структуру, яка адаптована до потреб кожного користувача. Основні екрани програми розроблені так, щоб забезпечити швидкий доступ до ключових функцій.

Основні екрани та компоненти інтерфейсу

Інтерфейс програми складається з кількох функціональних екранів, кожен з яких виконує специфічні завдання, залежно від ролі користувача.

1. Екран аутентифікації (Login)

Це стартовий екран системи, який забезпечує вхід користувачів за логіном і паролем. Він містить просту форму для введення облікових даних. Після успішної автентифікації користувач потрапляє до інтерфейсу, що відповідає його ролі. Реалізовані механізми перевірки прав доступу гарантують, що кожен користувач має доступ лише до функцій і даних, які відповідають його обов’язкам.


**Рис. 3.1. Екран аутентифікації**

1. Екран управління персоналом (HR Dashboard)

Цей екран призначений для роботи HR-менеджерів і надає можливості управління співробітниками. Він містить таблицю, яка відображає дані про всіх працівників, включаючи їхні посади, статус зайнятості, періоди відпусток та іншу важливу інформацію.

Інтерфейс включає функції пошуку і фільтрації даних, що спрощує роботу з великим обсягом записів. Також передбачені кнопки для додавання, редагування та видалення записів про співробітників, а також формування звітів про зайнятість персоналу.

**
Рис. 3.2. Екран управління персоналом**

1. Екран управління об’єктами (Builder Dashboard)

Цей екран забезпечує доступ до інформації про всі будівельні об’єкти. Тут відображаються основні параметри кожного об’єкта, такі як його назва, місцезнаходження, статус (активний, завершений, на етапі підготовки) і залучені до нього працівники.

Система дозволяє прив’язувати будівельників до конкретних об’єктів, а також відстежувати їхній прогрес у виконанні робіт. Крім того, користувачі можуть формувати звіти про стан кожного об’єкта для подальшого аналізу або обговорення.

**
Рис. 3.3. Екран управління персоналом**

1. Екран складів (Storage Dashboard)

Для складських працівників передбачено екран, який відображає інформацію про товари та матеріали, що зберігаються на складах. Інтерфейс дозволяє переглядати залишки ресурсів, реєструвати надходження нових матеріалів, а також контролювати їхній рух між об’єктами та складами.
Додатково передбачено функцію відображення історії змін, що дозволяє відстежувати дії, пов’язані з кожним товаром чи матеріалом. Це допомагає забезпечити прозорість процесів і швидко виявляти можливі розбіжності в даних.

**
Рис. 3.4. Екран складів**

Інтерфейс системи розроблено з урахуванням принципів мінімалізму: кожен екран містить лише ті функції, які потрібні для конкретної ролі користувача. Це забезпечує простоту навчання і високу продуктивність у роботі. Завдяки використанню Swing інтерфейс легко адаптується до різних розмірів екранів і дозволяє працювати на будь-якому пристрої, де підтримується Java.

## **3.2. Опис функціоналу**

Система надає широкий набір функцій, які спрощують і автоматизують управління проектами, облік ресурсів, документообіг та моніторинг діяльності. Функціонал орієнтований на підвищення ефективності роботи підприємства, мінімізацію ручної праці та забезпечення прозорості процесів.

Автоматизація управління проектами

Основна мета автоматизації управління проектами — полегшити координацію робіт на об’єктах будівництва та забезпечити своєчасне виконання завдань.

Система дозволяє створювати й управляти проектами, вказуючи їхні основні параметри: назву, місцезнаходження, терміни виконання, відповідальних осіб і статус (активний, завершений, у процесі підготовки). Користувачі можуть призначати виконавців на конкретні етапи проекту, а також вести облік виконаних робіт.
Автоматизація забезпечує синхронізацію даних між усіма користувачами, зокрема зміни в статусі проекту чи оновлення інформації про ресурси автоматично відображаються для всіх зацікавлених сторін. Це сприяє ефективній координації команд і зменшує ймовірність помилок, пов’язаних із застарілими даними.

Інструменти для обліку матеріалів і ресурсів

Функції обліку матеріалів і ресурсів спрямовані на поліпшення контролю за запасами і їх рухом. Система дозволяє складати список ресурсів, які використовуються на об’єктах, із вказанням їх кількості, одиниць вимірювання та місць зберігання.

Для складських працівників передбачені інструменти реєстрації надходження нових матеріалів, списання використаних і переміщення між об’єктами. Наприклад, якщо матеріал переміщується зі складу на об’єкт, система автоматично оновлює дані про залишки.

Інструменти також підтримують відстеження терміну придатності матеріалів (якщо це необхідно), історію змін і створення попереджень про критичні залишки. Це дозволяє мінімізувати перевитрати або недостачу матеріалів, що є ключовим для безперервного виконання проектів.

Можливості для ведення документообігу

Система забезпечує зберігання і управління документами, які супроводжують процеси управління проектами, обліку ресурсів та інших задач.

Користувачі можуть завантажувати, зберігати і організовувати документи, такі як контракти, технічні специфікації, акти виконаних робіт, рахунки-фактури та інші важливі файли. Документообіг організований таким чином, щоб забезпечити швидкий доступ до потрібної інформації та її редагування (залежно від прав доступу).

Інтеграція функції пошуку дозволяє швидко знаходити документи за ключовими параметрами (датою створення, типом документа, назвою проекту). Також передбачена можливість збереження історії змін у документах, що забезпечує прозорість та зручність у разі необхідності аналізу або перегляду попередніх версій.

Моніторинг і звітність

Для забезпечення контролю за станом проектів, ресурсів і персоналу система пропонує інструменти моніторингу та формування звітів.

Моніторинг дозволяє в реальному часі відстежувати стан активних проектів, запаси матеріалів на складах і залученість працівників. Усі зміни в системі відображаються миттєво, що дозволяє уникати простоїв і затримок.

Функція звітності дає змогу формувати детальні звіти за різними напрямками:

* статус виконання проектів;
* використання ресурсів;
* зайнятість персоналу;
* рух товарів між складами та об’єктами.

## **3.3. Технічні аспекти реалізації**

Технічна реалізація системи базується на використанні модульного підходу, який розділяє функціонал програми на логічні частини для полегшення розробки, тестування та підтримки. У цьому розділі описані основні модулі коду, наведені приклади реалізації ключових функцій та розглянуто особливості інтеграції з іншими системами.

Опис основних модулів коду

Модуль аутентифікації та авторизації

Модуль відповідає за перевірку користувачів, управління їхніми ролями та забезпечення доступу до функцій програми відповідно до прав. У ньому реалізовані функції перевірки логіна і пароля, а також механізми шифрування для безпечного зберігання паролів.
Ключові таблиці в базі даних:

* accounts: зберігає дані користувачів (логін, пароль, роль)

|  |
| --- |
|  private void authenticateUser() { String login = loginField.getText(); String password = new String(passwordField.getPassword()); try { // Перевіряємо чи з'єднання закрите, якщо так - створюємо нове if (connection == null || connection.isClosed()) { connection = jdbc.getConnection(); } String query = "SELECT \* FROM accounts WHERE UserName=? AND Password=?"; PreparedStatement statement = connection.prepareStatement(query); statement.setString(1, login); statement.setString(2, password); ResultSet resultSet = statement.executeQuery(); if (resultSet.next()) { // Успішний вхід currentLogin = login; // Зберігаємо успішний логін JOptionPane.showMessageDialog(frame,"Login successful. Welcome, "+currentLogin+"!"); // Виклик інших класів залежно від логіну openNextClass(currentLogin); frame.dispose(); // Закриваємо вікно логіну } else { // Помилка входу JOptionPane.showMessageDialog(frame, "Invalid login or password."); } } catch (SQLException ex) { ex.printStackTrace(); } } |

Табл. 3.1 – метод «authenticateUser», модуль аутентифікації та авторизації

Модуль управління об'єктами

Цей модуль обробляє інформацію про будівельні об’єкти, їх статус і зв’язок із залученими працівниками.

Таблиці в базі даних:

* involvedobject: зберігає дані про будівельників, залучених до конкретних об'єктів.
* allbuilders: містить інформацію про будівельників.

Модуль управління складом

Модуль відповідає за облік матеріалів, реєстрацію їхнього руху між складами та об’єктами, а також відображення поточних залишків. Він дозволяє працівникам складу фіксувати надходження, використання і переміщення ресурсів.

Таблиці в базі даних:

* materials: зберігає перелік матеріалів.
* storage: відображає залишки матеріалів.

Синхронізація через тригери

Використання тригерів у базі даних забезпечує автоматичне оновлення пов’язаних таблиць. Наприклад, додавання нового запису до таблиці employerslist з значенням Builder в колонці Department автоматично оновлює дані в таблиці allbuilders.
Приклад SQL тргерів для синхронізації даних на окремих базах об’єктів та таблиці БД відділу закупівель описаний в таблицях 3.2, 3.3, 3.4.

|  |
| --- |
| BEGIN -- Якщо DeliveredTo змінюється з 'Object1', видаляємо запис з таблиці itemsob1have IF OLD.DeliveredTo = 'Object1' THEN DELETE FROM itemsob1have WHERE ID = OLD.ID; END IF;END |

Табл. 3.2 – тригер для видалення даних з таблиці БД «itemsob1have» при зміні даних в колонці «DeliveredTo» з Object1 на будь яке інше значення в таблиці БД «purchasing\_list».

|  |
| --- |
| BEGIN -- Якщо Status = 'Delivered' і DeliveredTo = 'Object1', вставляємо дані в таблицю itemsob1have IF NEW.Status = 'Delivered' AND NEW.DeliveredTo = 'Object1' THEN INSERT INTO itemsob1have (ID, IName, Value, MUnit, ItemType, ObjectNum) VALUES (NEW.ID, NEW.IName, NEW.Value, NEW.MUnit, NEW.ItemType, 'Object1'); END IF;END |

Табл. 3.3 – тригер для перенесення даних з таблиці БД «itemsob1have» при введенні нових даних в колонці «DeliveredTo» на Object1 в таблиці БД «purchasing\_list»

|  |
| --- |
| BEGIN -- Якщо DeliveredTo змінюється на Object1, вставляємо в таблицю itemsob1have IF NEW.DeliveredTo = 'Object1' AND OLD.DeliveredTo != 'Object1' THEN INSERT INTO itemsob1have (ID, IName, Value, MUnit, ItemType, ObjectNum) VALUES (NEW.ID, NEW.IName, NEW.Value, NEW.MUnit, NEW.ItemType, 'Object1'); END IF; -- Якщо DeliveredTo змінюється з Object1 на інше значення, видаляємо з таблиці itemsob1have IF OLD.DeliveredTo = 'Object1' AND NEW.DeliveredTo != 'Object1' THEN DELETE FROM itemsob1have WHERE ID = OLD.ID; END IF;END |

Табл. 3.4 – тригер для перенесення даних з таблиці БД «itemsob1have» при введенні нових даних в колонці «DeliveredTo» на Object1 в таблиці БД «purchasing\_list»

## **3.4. Тестування та результати**

Тестування є невід’ємною частиною процесу розробки програмного забезпечення. Воно дозволяє перевірити коректність роботи функціоналу, знайти та виправити помилки, а також оцінити відповідність програми вимогам. У цьому розділі розглянуто проведені види тестування, виявлені помилки та їх виправлення, а також аналіз отриманих результатів.

**Проведення тестування функціоналу програми**

Для перевірки працездатності системи було проведено кілька типів тестування:

* Функціональне тестування
Перевірялася коректність роботи основних функцій, таких як:
	+ аутентифікація та авторизація користувачів;
	+ управління об'єктами, включаючи додавання, редагування та видалення записів;
	+ облік матеріалів та реєстрація їхнього переміщення;
	+ формування звітів.

Наприклад, під час тестування аутентифікації вводилися як правильні, так і неправильні логіни/паролі для перевірки механізмів верифікації даних.

* Тестування безпеки

Було перевірено захист від SQL-ін’єкцій шляхом введення некоректних даних у форми входу та пошуку. Крім того, тестувалися сценарії спроб доступу до функцій, які не входять до прав користувачів.

* Інтеграційне тестування

Перевірялася взаємодія між модулями системи та базою даних, зокрема коректність роботи тригерів, оновлення пов’язаних таблиць і синхронізація змін у різних компонентах програми.

* Тестування продуктивності

Було змодельовано високонавантажені сценарії, коли одночасно працюють кілька користувачів, які виконують дії, що вимагають значних ресурсів (генерація звітів, переміщення великих обсягів даних).

**Виявлені помилки та їх виправлення**

У процесі тестування були виявлені наступні помилки:

* Проблема з обробкою некоректних логінів/паролів

При введенні некоректних даних у форму входу програма некоректно обробляла помилку, через що користувач не отримував повідомлення.

Виправлення: Додано обробку винятків для виведення зрозумілого повідомлення "Неправильний логін або пароль".

* Неправильна робота тригера у таблиці involvedobject1

Тригер для автоматичного видалення будівельника з об'єкта після зміни статусу об'єкта працював некоректно, залишаючи записи в таблиці.

Виправлення: Внесено зміни до SQL-тригера, щоб він враховував оновлений статус об'єкта.

* Дублювання записів у звітах

Звіти про рух матеріалів іноді містили дубльовані записи через некоректні SQL-запити.

Виправлення: Оптимізовано SQL-запит, використавши DISTINCT для уникнення дублювань.

**Аналіз результатів тестування**

Після усунення виявлених помилок система пройшла повторне тестування, що показало значне покращення стабільності та продуктивності.

* Функціональність: Усі основні функції працюють відповідно до технічного завдання. Користувачі можуть успішно виконувати аутентифікацію, управляти об'єктами, переміщувати ресурси та формувати звіти.
* Безпека: Перевірка на SQL-ін’єкції та інші потенційні атаки показала, що система стійка до загроз.
* Інтеграція: Взаємодія з базою даних працює стабільно, тригери коректно обробляють зміни у пов'язаних таблицях.

У результаті проведених тестів підтверджено, що функціонал системи відповідає поставленим вимогам.

## **3.5. Потенційні напрями розвитку програми**

Для забезпечення гнучкості, масштабованості та підвищення зручності використання програми в майбутньому, можна впровадити додаткові функції, а також оптимізувати існуючу архітектуру. У цьому розділі наведено опис можливих нових функцій, а також пропозиції щодо покращення продуктивності та масштабування системи.

Опис можливих майбутніх функцій

* Додавання нових об’єктів однією клавішею

Можна реалізувати функцію швидкого додавання нових об'єктів через інтерактивну кнопку в інтерфейсі, яка створює об'єкт із типовими параметрами за замовчуванням (наприклад, "Новий об'єкт", статус "Активний"). Це прискорить процес для менеджерів, які працюють з великою кількістю однотипних проєктів.

* Створення ролей і автоматизація додавання нових користувачів через адміністратора

Реалізація інтерфейсу для адміністраторів, що дозволить створювати нові ролі з детальними правами доступу. Адміністратор зможе автоматично створювати нові облікові записи для співробітників з попередньо заданими параметрами доступу, спрощуючи масштабування системи в умовах зростання кількості користувачів.

* Реалізація мобільного додатка

Створення мобільної версії програми для смартфонів і планшетів дозволить користувачам, таким як будівельники або складські працівники, оперативно вводити дані, отримувати інформацію про ресурси та завдання безпосередньо на об’єктах.

* Нотіфікації про важливі події

Впровадження системи сповіщень, яка інформуватиме користувачів про важливі події, наприклад, про закінчення строку виконання проєкту, недостатню кількість ресурсів на складі або зміну статусу об’єкта. Сповіщення можуть надсилатися на електронну пошту або відображатися безпосередньо в інтерфейсі.

* Інтеграція з бухгалтерськими системами

Можливість експорту фінансових звітів у формати, які сумісні з бухгалтерськими програмами (наприклад, 1С або SAP). Це дозволить автоматизувати фінансовий облік матеріалів та витрат.

Пропозиції щодо оптимізації та масштабування програми

* Використання хмарного сервера БД або фізичного сервера з білою IP-адресою

Для підвищення доступності і швидкості роботи системи можна перенести базу даних на хмарний сервер (наприклад, Amazon RDS, Google Cloud SQL або Azure). Хмарні сервери забезпечують масштабованість і резервування даних. Альтернативним варіантом є використання фізичного сервера з білою IP-адресою для забезпечення швидкого доступу до БД з різних локацій.

* Оптимізація SQL-запитів

Аналіз частотних запитів до бази даних і створення додаткових індексів на полях, які часто використовуються у фільтрації чи пошуку, дозволить значно знизити час виконання запитів.

* Кешування запитів

Впровадження механізму кешування результатів найбільш поширених SQL-запитів (наприклад, за допомогою Redis або Memcached). Це дозволить знизити навантаження на сервер бази даних і прискорити відображення даних.

* Горизонтальне масштабування бази даних

У разі збільшення обсягів даних можна реалізувати шардінг бази даних, розділяючи таблиці між кількома серверами. Це забезпечить зростання продуктивності при роботі з великими обсягами інформації.

* Оптимізація графічного інтерфейсу

Перехід на сучасні UI-фреймворки (наприклад, JavaFX замість Swing) дозволить створити більш сучасний і адаптивний інтерфейс, який забезпечує підтримку різних типів пристроїв (комп’ютерів, планшетів, смартфонів).

* Розширення API для інтеграції з іншими системами

Впровадження REST або GraphQL API для взаємодії з іншими програмами. Це дозволить зовнішнім системам отримувати дані або інтегруватися з функціоналом програми безпосередньо.

* Модуль аналізу та прогнозування

Використання алгоритмів машинного навчання для аналізу даних і прогнозування, наприклад, майбутніх потреб у ресурсах або ймовірності затримки виконання проектів. Це допоможе приймати більш обґрунтовані управлінські рішення.

## **Висновки до розділу 3**

В цьому розділі детально описується програмна реалізація та функціонал системи, яка створена для автоматизації управління персоналом, будівельними об’єктами та складськими ресурсами. Система надає широкий набір функцій, орієнтованих на оптимізацію рутинних процесів і покращення ефективності роботи користувачів.

Інтерфейс програми, розроблений на основі Java Swing, забезпечує інтуїтивну взаємодію користувачів із системою. Чітке розмежування функцій між різними екранами — від управління персоналом і об'єктами до обліку ресурсів — дозволяє кожній категорії користувачів виконувати завдання відповідно до їхніх потреб. Додаткові інструменти, такі як звіти, пошук, фільтрація даних та документообіг, роблять систему універсальною та зручною у використанні.

Технічна реалізація базується на модульному підході, що включає аутентифікацію користувачів, управління ресурсами, інтеграцію з базою даних та автоматичне оновлення пов’язаних даних через SQL-тригери. Це забезпечує високу продуктивність системи та можливість її масштабування в майбутньому.

Таким чином, реалізована система успішно досягає поставлених цілей, надаючи користувачам інструменти для ефективного управління ключовими бізнес-процесами підприємства. Вона слугує основою для подальшого розширення функціоналу та впровадження нових технологій.

# **ВИСНОВКИ**

Дипломна робота була присвячена розробці програмного забезпечення для автоматизації ключових бізнес-процесів будівельної організації, зокрема управління проєктами, ресурсами, документообігом і моніторингом. У ході виконання роботи було досягнуто поставленої мети та реалізовано завдання, визначені у вступі.

У першому розділі проведено аналіз сучасного стану автоматизації в будівельній галузі, розглянуто переваги автоматизації бізнес-процесів та визначено основні проблеми, з якими стикаються підприємства під час впровадження автоматизованих систем. Це дозволило сформувати чітке розуміння вимог до програмного забезпечення, що розробляється.

Другий розділ був присвячений технічним аспектам реалізації системи. Було описано архітектуру програмного забезпечення, логіку роботи ключових модулів, а також вибір технологій, які забезпечують стабільну роботу системи. Реалізація бази даних із використанням MySQL, створення зручного інтерфейсу за допомогою Java Swing та впровадження тригерів для автоматизації операцій дозволили забезпечити продуктивність, масштабованість і простоту використання.

У третьому розділі було проведено тестування системи в умовах, наближених до реальних, що дозволило оцінити її функціональність і продуктивність. Виявлені помилки були виправлені, а результати тестування підтвердили відповідність системи поставленим вимогам. Також були визначені напрями для подальшого розвитку, зокрема інтеграція з мобільними платформами, впровадження хмарних рішень і розширення функціоналу.

Запропоноване програмне рішення показало високу ефективність у вирішенні завдань автоматизації роботи будівельної організації. Воно дозволяє:

* зменшити витрати часу на виконання рутинних завдань;
* оптимізувати використання ресурсів;
* підвищити точність і прозорість процесів;
* покращити комунікацію між підрозділами організації.

Результати роботи можуть бути корисними для малих і середніх будівельних компаній, які прагнуть підвищити свою конкурентоспроможність завдяки впровадженню сучасних інформаційних технологій. Запропонована система є універсальною та може бути адаптована до специфічних потреб інших організацій у будівельній галузі.

Таким чином, виконана дипломна робота доводить, що автоматизація є важливим інструментом для оптимізації бізнес-процесів і забезпечення ефективності діяльності будівельних компаній в умовах сучасного ринку.

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. 27.3.1Trigger Syntax and Examples URL: https://dev.mysql.com/doc/refman/8.4/en/trigger-syntax.html
2. Architecture and Engineering Services. URL: https://www.trade.gov/architecture-and-engineering-services
3. Establishing JDBC Connection in Java URL: https://www.geeksforgeeks.org/establishing-jdbc-connection-in-java/
4. Introduction to Java Swing URL: https://www.geeksforgeeks.org/introduction-to-java-swing/
5. Introduction to JDBC URL: https://www.baeldung.com/java-jdbc
6. JDBC або з чого все починається URL: https://javarush.com/ua/groups/posts/uk.2172.jdbc-abo-z-chogo-vse-pochinatjhsja
7. MySQL Triggers URL: https://dev.mysql.com/doc/refman/8.4/en/trigger-syntax.html
8. MySQL Triggers: How and why with examples URL: https://www.dolthub.com/blog/2023-06-09-writing-mysql-triggers/
9. SmarTTown, Організація будівельного процесу URL: https://gaz.rv.ua/statti/budivnitstvo/organizatsiya-budivelnogo-protsesu/ (дата звернення 15.10.2024)
10. StudFile, ОСНОВИ ТЕХНОЛОГІЇ БУДІВЕЛЬНИХ ПРОЦЕСІВ URL: https://studfile.net/preview/9686918/ (дата звернення 13.10.2024)
11. Swing Introduction URL: https://www.javatpoint.com/java-swing
12. Working with a connection URL: https://learn.microsoft.com/en-us/sql/connect/jdbc/working-with-a-connection?view=sql-server-ver16
13. Гірс Н. О. «Іноваційні підходи до програмної реалізації задач автоматизації роботи будівельної органіхації» Вісник студентського наукового товариства [електронне видання]: збірник наукових праць студентів, магістрантів і аспірантів / за заг. ред. О. В. Мельничука. Ніжин: НДУ ім. М. Гоголя, 2024. Вип. 30. 253 с. URL: http://www.ndu.edu.ua/storage/styd\_visnuk/visnuk\_stud\_tov\_30\_2024.pdf
14. Гірс Н. О. Роль автоматизації в підвищенні ефективності роботи будівельних організацій // Х міжнародній науково-практичній інтернет-конференції «Наука і молодь у ХХІ сторіччі».
15. Державні будівельні норми України управління, організація і технологія організація будівельного виробництва ДБН А.3.1-5:2016
16. Драйвер Microsoft JDBC для SQL Server URL: https://learn.microsoft.com/ru-ru/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server?view=sql-server-ver16
17. ДСТУ EN ISO 19650\_5. URL: https://uscc.ua/dstu-en-iso-196505
18. Корінєв В. Л. Роль реконфігурації в підвищенні ефективності підприємницької діяльності підприємства. – Кропивницький : Економічні науки, 2011. Вип. 19. – 45-50 с.
19. Осташевська, Г.Г. Технологія будівництва. Тексти лекцій для студентів рівня підготовки «Бакалавр» за напрямом 1201. – Харків : «Архітектура», 2009 – 84 с.
20. Що таке BIM і навіщо потрібні нові технології? - BIM partner. URL: https://bimpartner.com.ua/what-is-bim/
21. Що таке jdbc в Java? URL: https://itproger.com/ua/spravka/java/jdbc
22. Що таке MySQL? URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-mysql/
23. Я. Й. Коцій, І. Г. Іваник, С. І. Віхоть. Технологія будівельного виробництва. Курс лекцій – Львів: Національний університет «Львівська політехніка», 2006. –  94 с.

# **ДОДАТКИ**

Додаток А

Код класу StartPoint – аутентифікація та авторизація користувачів

import javax.swing.\*;
import java.awt.\*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class StartPoint {
 private JFrame frame;
 private JTextField loginField;
 private JPasswordField passwordField;
 private JLabel loginLabel;
 private JLabel passwordLabel;
 private JButton loginButton;
 private JDBC jdbc;
 private Connection connection;
 private String currentLogin; // Змінна для зберігання логіну

 public StartPoint() {
 // Ініціалізація вікна
 frame = new JFrame("Logon Page");
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);
 frame.setSize(300, 200);
 frame.setLayout(new GridBagLayout());
 GridBagConstraints gbc = new GridBagConstraints();

 // Поле для вводу логіну
 loginLabel = new JLabel("Login:");
 gbc.gridx = 0;
 gbc.gridy = 0;
 gbc.anchor = GridBagConstraints.*WEST*;
 frame.add(loginLabel, gbc);

 loginField = new JTextField(15);
 gbc.gridx = 1;
 gbc.gridy = 0;
 gbc.anchor = GridBagConstraints.*CENTER*;
 frame.add(loginField, gbc);

 // Поле для вводу пароля
 passwordLabel = new JLabel("Password:");
 gbc.gridx = 0;
 gbc.gridy = 1;
 gbc.anchor = GridBagConstraints.*WEST*;
 frame.add(passwordLabel, gbc);

**Продовження додатку А**

Код класу StartPoint – аутентифікація та авторизація користувачів

 passwordField = new JPasswordField(15);
 gbc.gridx = 1;
 gbc.gridy = 1;
 gbc.anchor = GridBagConstraints.*CENTER*;
 frame.add(passwordField, gbc);

 // Кнопка для входу
 loginButton = new JButton("Login");
 gbc.gridx = 1;
 gbc.gridy = 2;
 gbc.anchor = GridBagConstraints.*CENTER*;
 frame.add(loginButton, gbc);

 // Ініціалізація підключення до бази даних
 jdbc = new JDBC();

 // Дія при натисканні на кнопку
 loginButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 authenticateUser();
 }
 });

 frame.setVisible(true);
 }

 private void authenticateUser() {
 String login = loginField.getText();
 String password = new String(passwordField.getPassword());

 try {
 // Перевіряємо чи з'єднання закрите, якщо так - створюємо нове
 if (connection == null || connection.isClosed()) {
 connection = jdbc.getConnection();
 }

 String query = "SELECT \* FROM accounts WHERE UserName = ? AND Password = ?";
 PreparedStatement statement = connection.prepareStatement(query);
 statement.setString(1, login);
 statement.setString(2, password);

 ResultSet resultSet = statement.executeQuery();

 if (resultSet.next()) {
 // Успішний вхід
 currentLogin = login; // Зберігаємо успішний логін
 JOptionPane.*showMessageDialog*(frame, "Login successful. Welcome, " + currentLogin + "!");

**Продовження додатку А**

Код класу StartPoint – аутентифікація та авторизація користувачів

 // Виклик інших класів залежно від логіну
 openNextClass(currentLogin);
 frame.dispose(); // Закриваємо вікно логіну
 } else {
 // Помилка входу
 JOptionPane.*showMessageDialog*(frame, "Invalid login or password.");
 }
 } catch (SQLException ex) {
 ex.printStackTrace();
 }
 }

 private void openNextClass(String login) {
 if ("HR".equals(login)) {
 new HRWorkSpace(login); // Якщо логін HR, відкриваємо HRWorkSpace
 } else if ("Storage".equals(login)) {
 new StorageWorkSpace(login); // Якщо логін Storage, відкриваємо StorageWorkSpace
 } else if ("Purchasing".equals(login)) {
 new PurchasingWorkSpace(login); // Якщо логін Purchasing, відкриваємо PurchasingWorkSpace
 } else if ("MainBuilder".equals(login)|| "Builder1".equals(login)|| "Builder2".equals(login)||"Builder3".equals(login)) {
 new BuildingWorkSpace(login); // Якщо логін MainBuilder,Builder1,Builder2,Builder3, відкриваємо BuildingWorkSpace
 } else {
 JOptionPane.*showMessageDialog*(frame, "Unauthorized access.");
 }
 }

 // Геттер для отримання поточного логіну (якщо знадобиться)
 public String getCurrentLogin() {
 return currentLogin;
 }

 public static void main(String[] args) {
 new StartPoint();
 }

}

Додаток Б

Код класу HRWorkSpace – робочий простір користувачів HR

import javax.swing.\*;
import java.awt.\*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.\*;
import java.util.HashMap;
import java.util.Map;

public class HRWorkSpace {
 private JFrame frame;
 private String currentclass;
 private String currentLogin;
 private JComboBox<String> optionComboBox; // Випадаючий список
 private JButton signOutButton;
 private String selectedOption; // Змінна для зберігання вибраної опції
 private String tableName; // Змінна для назви таблиці

 public HRWorkSpace(String login) {
 this.currentLogin = login; // Зберігаємо логін
 createAndShowGUI();
 }

 private void createAndShowGUI() {
 frame = new JFrame("Welcome HR");
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);
 frame.setSize(800, 600);
 frame.setLayout(new GridBagLayout());
 GridBagConstraints gbc = new GridBagConstraints();
 String currentclass = "HRWorkSpace";

 // Випадаючий список
 String[] options = {"Vacancy", "Employers list"};
 optionComboBox = new JComboBox<>(options);
 gbc.gridx = 0;
 gbc.gridy = 0;
 gbc.gridwidth = 2; // займає 2 колонки
 gbc.anchor = GridBagConstraints.*CENTER*;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо
 frame.add(optionComboBox, gbc);

 Dimension buttonSize = new Dimension(120, 30); // Встановлення однакового розміру для кнопок

 // Кнопка List
 JButton listButton = new JButton("List");
 listButton.setPreferredSize(buttonSize);
 listButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {

**Продовження додатку Б**

Код класу HRWorkSpace – робочий простір користувачів HR

 Next();
 frame.dispose();
 new List(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridy = 1;
 gbc.gridwidth = 1; // займає одну колонку
 frame.add(listButton, gbc);
 // Кнопка Edit
 JButton editButton = new JButton("Edit");
 editButton.setPreferredSize(buttonSize);
 editButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Edit(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 1;
 frame.add(editButton, gbc);

 // Кнопка Add
 JButton addButton = new JButton("Add");
 addButton.setPreferredSize(buttonSize);
 addButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Add(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 0;
 gbc.gridy = 2;
 frame.add(addButton, gbc);

 // Кнопка Search
 JButton searchButton = new JButton("Search");
 searchButton.setPreferredSize(buttonSize);
 searchButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Search(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 1;
 frame.add(searchButton, gbc);

**Продовження додатку Б**

Код класу HRWorkSpace – робочий простір користувачів HR

 // Кнопка Sign Out
 signOutButton = new JButton("Sign Out");
 signOutButton.setPreferredSize(buttonSize);
 signOutButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 signOut();
 }
 });
 gbc.gridx = 0;
 gbc.gridy = 3;
 gbc.gridwidth = 2;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо кнопки
 frame.add(signOutButton, gbc);

 frame.setVisible(true);
 }

 private void signOut() {
 frame.dispose();
 new StartPoint();
 }

 private void Next() {
 selectedOption = (String) optionComboBox.getSelectedItem();
 if ("Vacancy".equals(selectedOption)) {
 tableName = "vacancy";
 updateVacancyEHave();
 } else if ("Employers list".equals(selectedOption)) {
 tableName = "employerslist";
 } else {
 tableName = null;
 }
 }

 private void updateVacancyEHave() {
 try (Connection connection = new JDBC().getConnection()) {
 // Отримуємо всі департаменти з таблиці vacancy
 String queryVacancy = "SELECT Departament FROM vacancy";
 Statement stmt = connection.createStatement();
 ResultSet rsVacancy = stmt.executeQuery(queryVacancy);

 // Зберігаємо департаменти та їхні повторення з employerslist
 Map<String, Integer> departmentCountMap = new HashMap<>();

 // Розміщуємо обробку результатів запиту до employerslist в окрему змінну
 String queryEmployers = "SELECT Department FROM employerslist";
 try (Statement stmtEmployers = connection.createStatement()) {
 ResultSet rsEmployers = stmtEmployers.executeQuery(queryEmployers);

**Продовження додатку Б**

Код класу HRWorkSpace – робочий простір користувачів HR

 while (rsEmployers.next()) {
 String department = rsEmployers.getString("Department");
 departmentCountMap.put(department, departmentCountMap.getOrDefault(department, 0) + 1);
 }
 }

 // Оновлюємо EHave для кожного департаменту в таблиці vacancy
 while (rsVacancy.next()) {
 String department = rsVacancy.getString("Departament");
 int count = departmentCountMap.getOrDefault(department, 0);

 String updateQuery = "UPDATE vacancy SET EHave = ? WHERE Departament = ?";
 try (PreparedStatement pstmt = connection.prepareStatement(updateQuery)) {
 pstmt.setInt(1, count);
 pstmt.setString(2, department);
 pstmt.executeUpdate();
 }
 }

 } catch (SQLException ex) {
 ex.printStackTrace();
 JOptionPane.*showMessageDialog*(frame, "Error updating EHave: " + ex.getMessage());
 }
 }
}

Додаток В

Код класу BuildingWorkSpace – робочий простір Прорабів на об’єктах та головного прораба

import javax.swing.\*;
import java.awt.\*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class BuildingWorkSpace {
 private JFrame frame;
 private String currentclass;
 private String currentLogin;
 private JComboBox<String> optionComboBox; // Випадаючий список
 private JButton addButton, editButton, signOutButton, acceptButton, listButton, searchButton;
 private String selectedOption; // Змінна для зберігання вибраної опції
 private String tableName; // Змінна для назви таблиці

 public BuildingWorkSpace(String login) {
 this.currentLogin = login; // Зберігаємо логін
 if ("MainBuilder".equals(currentLogin)) {
 createAndShowGUI();
 } else if ("Builder1".equals(currentLogin) || "Builder2".equals(currentLogin) || "Builder3".equals(currentLogin)) {
 createAndShowGUI2();
 }
 }
 private void createAndShowGUI() {
 frame = new JFrame("Welcome Main Builder");
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);
 frame.setSize(800, 600);
 frame.setLayout(new GridBagLayout());
 GridBagConstraints gbc = new GridBagConstraints();
 String currentclass = "BuildingWorkSpace";
 // Випадаючий список
 String[] options = {"Involved builders in Obj1", "Involved builders in Obj2", "Involved builders in Obj3",
 "Items status on Obj1","Items status on Obj1","Items status on Obj1",
 "Items on Obj1", "Items on Obj2", "Items on Obj3", "All objects status", "All Builders",
 "Involved Builders", "NOT Involved Builders"};
 optionComboBox = new JComboBox<>(options);
 gbc.gridx = 0;
 gbc.gridy = 0;
 gbc.gridwidth = 2; // займає 2 колонки
 gbc.anchor = GridBagConstraints.*CENTER*;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо
 frame.add(optionComboBox, gbc);
 // Кнопка Accept
 acceptButton = new JButton("Accept");
 acceptButton.setPreferredSize(new Dimension(120, 30));
 acceptButton.setEnabled(false); // Спочатку кнопка вимкнена
 acceptButton.addActionListener(new ActionListener() {

**Продовження додатку В**

Код класу BuildingWorkSpace – робочий простір Прорабів на об’єктах та

 @Override
 public void actionPerformed(ActionEvent e) {
 Next(); // Отримуємо вибрану опцію
 frame.dispose(); // Закриваємо поточне вікно
 new BuildersList(selectedOption, currentclass, currentLogin); // Переходимо до BuildersList
 }
 });
 gbc.gridy = 1;
 frame.add(acceptButton, gbc);

 // Решта кнопок (List, Edit, Add, Search, Sign Out)
 Dimension buttonSize = new Dimension(120, 30); // Встановлення однакового розміру для кнопок

 listButton = new JButton("List");
 listButton.setPreferredSize(buttonSize);
 listButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new List(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridy = 2;
 gbc.gridwidth = 1; // займає одну колонку
 frame.add(listButton, gbc);

 editButton = new JButton("Edit");
 editButton.setPreferredSize(buttonSize);
 editButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Edit(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 1;
 frame.add(editButton, gbc);

 addButton = new JButton("Add");
 addButton.setPreferredSize(buttonSize);
 addButton.addActionListener(new ActionListener() {

**Продовження додатку В**

Код класу BuildingWorkSpace – робочий простір Прорабів на об’єктах та

 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Add(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 0;
 gbc.gridy = 3;
 frame.add(addButton, gbc);

 searchButton = new JButton("Search");
 searchButton.setPreferredSize(buttonSize);
 searchButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Search(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 1;
 frame.add(searchButton, gbc);

 // Кнопка Sign Out
 signOutButton = new JButton("Sign Out");
 signOutButton.setPreferredSize(buttonSize);
 signOutButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 signOut();
 }
 });
 gbc.gridx = 0;
 gbc.gridy = 4;
 gbc.gridwidth = 2;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо кнопки
 frame.add(signOutButton, gbc);

 // Слухач для комбінованого списку
 optionComboBox.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 updateAcceptButtonState(); // Оновлюємо стан кнопки Accept
 updateButtonsState(); // Перевіряємо стан кнопок
 }
 });
 frame.setVisible(true);
 }

**Продовження додатку В**

Код класу BuildingWorkSpace – робочий простір Прорабів на об’єктах та

 private void createAndShowGUI2() {
 frame = new JFrame("Welcome Builder");
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);
 frame.setSize(800, 600);
 frame.setLayout(new GridBagLayout());
 GridBagConstraints gbc = new GridBagConstraints();
 String currentclass = "BuildingWorkSpace";

 if ("Builder1".equals(currentLogin))
 {
 // Випадаючий список
 String[] options = {"Involved builders in Obj1", "Items on Obj1", "Items status on Obj1", "All objects status"};
 optionComboBox = new JComboBox<>(options);
 gbc.gridx = 0;
 gbc.gridy = 0;
 gbc.gridwidth = 2; // займає 2 колонки
 gbc.anchor = GridBagConstraints.*CENTER*;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо
 frame.add(optionComboBox, gbc);
 }
 else if ("Builder2".equals(currentLogin))
 {
 // Випадаючий список
 String[] options = {"Involved builders in Obj2", "Items on Obj2", "Items status on Obj2", "All objects status"};
 optionComboBox = new JComboBox<>(options);
 gbc.gridx = 0;
 gbc.gridy = 0;
 gbc.gridwidth = 2; // займає 2 колонки
 gbc.anchor = GridBagConstraints.*CENTER*;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо
 frame.add(optionComboBox, gbc);
 }
 else if ("Builder3".equals(currentLogin))
 {
 // Випадаючий список
 String[] options = {"Involved builders in Obj3", "Items on Obj3", "Items status on Obj1", "All objects status"};
 optionComboBox = new JComboBox<>(options);
 gbc.gridx = 0;
 gbc.gridy = 0;
 gbc.gridwidth = 2; // займає 2 колонки
 gbc.anchor = GridBagConstraints.*CENTER*;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо
 frame.add(optionComboBox, gbc);
 }
 Dimension buttonSize = new Dimension(120, 30); // Встановлення однакового розміру для кнопок

**Продовження додатку В**

Код класу BuildingWorkSpace – робочий простір Прорабів на об’єктах та

 // Кнопка List
 JButton listButton = new JButton("List");
 listButton.setPreferredSize(buttonSize);
 listButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new List(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridy = 1;
 gbc.gridwidth = 1; // займає одну колонку
 frame.add(listButton, gbc);

 // Кнопка Edit
 editButton = new JButton("Edit");
 editButton.setPreferredSize(buttonSize);
 editButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Edit(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 1;
 frame.add(editButton, gbc);

 // Кнопка Add
 addButton = new JButton("Add");
 addButton.setPreferredSize(buttonSize);
 addButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Add(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 0;
 gbc.gridy = 2;
 frame.add(addButton, gbc);

 // Кнопка Search
 JButton searchButton = new JButton("Search");
 searchButton.setPreferredSize(buttonSize);
 searchButton.addActionListener(new ActionListener() {

**Продовження додатку В**

Код класу BuildingWorkSpace – робочий простір Прорабів на об’єктах та

 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Search(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 1;
 frame.add(searchButton, gbc);

 // Кнопка Sign Out
 signOutButton = new JButton("Sign Out");
 signOutButton.setPreferredSize(buttonSize);
 signOutButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 signOut();
 }
 });
 gbc.gridx = 0;
 gbc.gridy = 3;
 gbc.gridwidth = 2;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо кнопки
 frame.add(signOutButton, gbc);

 frame.setVisible(true);
 }

 private void updateAcceptButtonState() {
 selectedOption = (String) optionComboBox.getSelectedItem();
 acceptButton.setEnabled(
 "All Builders".equals(selectedOption) ||
 "Involved Builders".equals(selectedOption) ||
 "NOT Involved Builders".equals(selectedOption)
 );
 }

 private void updateButtonsState() {
 selectedOption = (String) optionComboBox.getSelectedItem();
 boolean enableButtons =
 "Involved builders in Obj1".equals(selectedOption) ||
 "Involved builders in Obj2".equals(selectedOption) ||
 "Involved builders in Obj3".equals(selectedOption) ||
 "Items on Obj1".equals(selectedOption) ||
 "Items on Obj2".equals(selectedOption) ||
 "Items on Obj3".equals(selectedOption) ||
 "All objects status".equals(selectedOption);

**Продовження додатку В**

Код класу BuildingWorkSpace – робочий простір Прорабів на об’єктах та

 listButton.setEnabled(enableButtons);
 editButton.setEnabled(enableButtons);
 addButton.setEnabled(enableButtons);
 searchButton.setEnabled(enableButtons);
 }
 private void signOut()
 {
 frame.dispose();
 new StartPoint();
 }

 private void Next() {
 selectedOption = (String) optionComboBox.getSelectedItem();
 if ("Involved builders in Obj1".equals(selectedOption)) {
 tableName = "involvedobject1";
 } else if ("Involved builders in Obj2".equals(selectedOption)) {
 tableName = "involvedobject2";
 } else if ("Involved builders in Obj3".equals(selectedOption)) {
 tableName = "involvedobject3";
 } else if ("Items on Obj1".equals(selectedOption)) {
 tableName = "itemsob1have";
 } else if ("Items on Obj2".equals(selectedOption)) {
 tableName = "itemsob2have";
 } else if ("Items on Obj3".equals(selectedOption)) {
 tableName = "itemsob3have";
 } else if ("All objects status".equals(selectedOption)) {
 tableName = "objectsstatus";
 } else if ("Items status on Obj1".equals(selectedOption)) {
 tableName = "itemsonobject№1";
 } else if ("Items status on Obj2".equals(selectedOption)) {
 tableName = "itemsonobject№2";
 } else if ("Items status on Obj3".equals(selectedOption)) {
 tableName = "itemsonobject№3";
 } else {
 tableName = null;
 }
 }
}

Додаток Г

Код класу PurchasingWorkSpace – робочий простір для працівників відділу

import javax.swing.\*;
import java.awt.\*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class PurchasingWorkSpace {
 private JFrame frame;
 private String currentclass;
 private String currentLogin;
 private JComboBox<String> optionComboBox; // Випадаючий список
 private JButton addButton, editButton, signOutButton;
 private String selectedOption; // Змінна для зберігання вибраної опції
 private String tableName; // Змінна для назви таблиці

 public PurchasingWorkSpace(String login) {
 this.currentLogin = login; // Зберігаємо логін
 createAndShowGUI();
 }

 private void createAndShowGUI() {
 frame = new JFrame("Welcome Purchasing department");
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);
 frame.setSize(800, 600);
 frame.setLayout(new GridBagLayout());
 GridBagConstraints gbc = new GridBagConstraints();
 String currentclass = "PurchasingWorkSpace";

 // Випадаючий список
 String[] options = {"In Storages","Storage1","Storage10","Storage11","Storages Status","Purchasing List"};
 optionComboBox = new JComboBox<>(options);
 optionComboBox.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 updateButtonState(); // Оновлення стану кнопок
 }
 });
 gbc.gridx = 0;
 gbc.gridy = 0;
 gbc.gridwidth = 2; // займає 2 колонки
 gbc.anchor = GridBagConstraints.*CENTER*;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо
 frame.add(optionComboBox, gbc);

 Dimension buttonSize = new Dimension(120, 30); // Встановлення однакового розміру для кнопок
 // Кнопка List
 JButton listButton = new JButton("List");
 listButton.setPreferredSize(buttonSize);
 listButton.addActionListener(new ActionListener() {

**Продовження додатку Г**

Код класу PurchasingWorkSpace – робочий простір для працівників відділу закупівель

 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new List(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridy = 1;
 gbc.gridwidth = 1; // займає одну колонку
 frame.add(listButton, gbc);

 // Кнопка Edit
 editButton = new JButton("Edit");
 editButton.setPreferredSize(buttonSize);
 editButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 {
 Next();
 frame.dispose();
 new Edit(tableName, selectedOption, currentclass, currentLogin);}
 }
 });
 gbc.gridx = 1;
 frame.add(editButton, gbc);

 // Кнопка Add
 addButton = new JButton("Add");
 addButton.setPreferredSize(buttonSize);
 addButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Add(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 0;
 gbc.gridy = 2;
 frame.add(addButton, gbc);

 // Кнопка Search
 JButton searchButton = new JButton("Search");
 searchButton.setPreferredSize(buttonSize);
 searchButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {

**Продовження додатку Г**

Код класу PurchasingWorkSpace – робочий простір для працівників відділу закупівель

 Next();
 frame.dispose();
 new Search(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 1;
 frame.add(searchButton, gbc);

 // Кнопка Sign Out
 signOutButton = new JButton("Sign Out");
 signOutButton.setPreferredSize(buttonSize);
 signOutButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 signOut();
 }
 });
 gbc.gridx = 0;
 gbc.gridy = 3;
 gbc.gridwidth = 2;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо кнопки
 frame.add(signOutButton, gbc);

 updateButtonState(); // Оновлення стану кнопок при ініціалізації
 frame.setVisible(true);
 }

 private void signOut() {
 frame.dispose();
 new StartPoint();
 }

 private void Next() {
 selectedOption = (String) optionComboBox.getSelectedItem();
 if ("In Storages".equals(selectedOption)) {
 tableName = "instorage";
 } else if ("Purchasing List".equals(selectedOption)) {
 tableName = "purchasing\_list";
 } else {
 tableName = null;
 }
 }
 private void updateButtonState() {
 selectedOption = (String) optionComboBox.getSelectedItem();
 boolean enabled = "Purchasing List".equals(selectedOption);
 addButton.setEnabled(enabled);
 editButton.setEnabled(enabled);
 }
}

Додаток Ґ

Код класу StorageWorkSpace – робочий простір для працівників складу

import javax.swing.\*;
import java.awt.\*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class StorageWorkSpace {
 private JFrame frame;
 private String currentclass;
 private String currentLogin;
 private JComboBox<String> optionComboBox; // Випадковий список
 private JButton signOutButton;
 private String selectedOption; // Змінна для зберігання вибраної опції
 private String tableName; // Змінна для назви таблиці

 public StorageWorkSpace(String login) {
 this.currentLogin = login; // Зберігаємо логін
 createAndShowGUI();
 }

 private void createAndShowGUI() {
 frame = new JFrame("Welcome Storage");
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);
 frame.setSize(800, 600);
 frame.setLayout(new GridBagLayout());
 GridBagConstraints gbc = new GridBagConstraints();
 String currentclass = "StorageWorkSpace";

 // Випадковий список
 String[] options = {"In Storages", "Storage1","Storage10","Storage11","Storages Status"};
 optionComboBox = new JComboBox<>(options);
 gbc.gridx = 0;
 gbc.gridy = 0;
 gbc.gridwidth = 2; // займає 2 колонки
 gbc.anchor = GridBagConstraints.*CENTER*;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо
 frame.add(optionComboBox, gbc);

 Dimension buttonSize = new Dimension(100, 30); // Встановлення однакового розміру для кнопок

 // Кнопка List
 JButton listButton = new JButton("List");
 listButton.setPreferredSize(buttonSize);
 listButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new List(tableName, selectedOption, currentclass, currentLogin);
 }
 });

**Продовження додатку Ґ**

Код класу StorageWorkSpace – робочий простір для працівників складу

 gbc.gridy = 1;
 gbc.gridwidth = 1;
 gbc.insets = new Insets(5, 5, 5, 5); // Відступи навколо кнопки
 frame.add(listButton, gbc);

 // Кнопка Edit
 JButton editButton = new JButton("Edit");
 editButton.setPreferredSize(buttonSize);
 editButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Edit(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 1;
 frame.add(editButton, gbc);

 // Кнопка Add
 JButton addButton = new JButton("Add");
 addButton.setPreferredSize(buttonSize);
 addButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Add(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 0;
 gbc.gridy = 2;
 frame.add(addButton, gbc);

 // Кнопка Search
 JButton searchButton = new JButton("Search");
 searchButton.setPreferredSize(buttonSize);
 searchButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 Next();
 frame.dispose();
 new Search(tableName, selectedOption, currentclass, currentLogin);
 }
 });
 gbc.gridx = 1;
 frame.add(searchButton, gbc);

**Продовження додатку Ґ**

Код класу StorageWorkSpace – робочий простір для працівників складу

 // Кнопка Sign Out
 signOutButton = new JButton("Sign Out");
 signOutButton.setPreferredSize(buttonSize);
 signOutButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 signOut();
 }
 });
 gbc.gridx = 0;
 gbc.gridy = 3;
 gbc.gridwidth = 2;
 gbc.insets = new Insets(10, 10, 10, 10); // Відступи навколо кнопки
 frame.add(signOutButton, gbc);

 frame.setVisible(true);
 }

 private void signOut() {
 frame.dispose(); // Закриваємо вікно Storage
 new StartPoint(); // Повертаємося до Logon Page
 }

 private void Next() {
 selectedOption = (String) optionComboBox.getSelectedItem();
 if ("In Storages".equals(selectedOption)) {
 tableName = "instorage";
 } else if ("Storage1".equals(selectedOption)) {
 tableName = "sn01\_storage";
 } else if ("Storage10".equals(selectedOption)) {
 tableName = "sn10\_storage";
 } else if ("Storage11".equals(selectedOption)) {
 tableName = "sn11\_storage";
 } else if ("Storages Status".equals(selectedOption)) {
 tableName = "storagesstatus";
 } else {
 tableName = null;
 }
 }
}

Додаток Д

Код класу JDBC – Java DataBase Connectivit (з'єднання з базами даних на Java)

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JDBC {
 private Connection connection;

 public JDBC() {
 try {
 String url = "jdbc:mysql://localhost:3306/jdbc\_diplom";
 String user = "root";
 String password = "root";
 connection = DriverManager.*getConnection*(url, user, password);
 } catch (SQLException ex) {
 ex.printStackTrace();
 }
 }

 public Connection getConnection() {
 return connection;
 }

 public void closeConnection() {
 try {
 if (connection != null && !connection.isClosed()) {
 connection.close();
 }
 } catch (SQLException ex) {
 ex.printStackTrace();
 }
 }
}

Додаток E

Код класу List – клас для подання таблиці в вікні

import javax.swing.\*;
import javax.swing.table.DefaultTableModel;
import java.awt.\*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.table.TableRowSorter;

public class List {
 private JFrame frame;
 private String tableName;
 private String windowTitle;
 private String currentclass;
 private String currentlogin;

 public List(String tableName, String windowTitle, String currentclass, String currentlogin) {
 this.currentclass = currentclass;
 this.currentlogin = currentlogin;
 this.tableName = tableName;
 this.windowTitle = windowTitle;
 createAndShowGUI();
 }

 private void createAndShowGUI() {
 frame = new JFrame(windowTitle); // Назва вікна
 frame.setSize(600, 400);
 frame.setLayout(new BorderLayout());

 // Таблиця для відображення даних
 JTable table = new JTable();
 JScrollPane scrollPane = new JScrollPane(table);
 frame.add(scrollPane, BorderLayout.*CENTER*);

 // Кнопка Back
 JButton backButton = new JButton("Back");
 backButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 back();
 }
 });
 frame.add(backButton, BorderLayout.*SOUTH*);

 // Завантажуємо дані в таблицю
 loadDataIntoTable(table);

 frame.setVisible(true);
 }

**Продовження додатку E**

Код класу List – клас для подання таблиці в вікні

 private void back() {
 if ("HRWorkSpace".equals(currentclass)) {
 new HRWorkSpace(currentlogin);
 } else if ("StorageWorkSpace".equals(currentclass)) {
 new StorageWorkSpace(currentlogin);
 } else if ("PurchasingWorkSpace".equals(currentclass)) {
 new PurchasingWorkSpace(currentlogin);
 } else if ("BuildingWorkSpace".equals(currentclass)) {
 new BuildingWorkSpace(currentlogin);
 } else {
 JOptionPane.*showMessageDialog*(frame, "Invalid operation");
 }
 frame.dispose();
 }

 private void loadDataIntoTable(JTable table) {
 try (Connection connection = new JDBC().getConnection()) {
 String query = "SELECT \* FROM " + tableName;
 PreparedStatement statement = connection.prepareStatement(query);
 ResultSet resultSet = statement.executeQuery();

 // Отримуємо метадані для таблиці
 DefaultTableModel model = new DefaultTableModel();
 int columnCount = resultSet.getMetaData().getColumnCount();
 for (int i = 1; i <= columnCount; i++) {
 model.addColumn(resultSet.getMetaData().getColumnName(i)); // Назви стовпчиків
 }

 // Додаємо дані
 while (resultSet.next()) {
 Object[] rowData = new Object[columnCount];
 for (int i = 0; i < columnCount; i++) {
 rowData[i] = resultSet.getObject(i + 1);
 }
 model.addRow(rowData);
 }

 table.setModel(model); // Встановлюємо модель таблиці

 TableRowSorter<DefaultTableModel> sorter = new TableRowSorter<>(model);
 table.setRowSorter(sorter);

 } catch (SQLException ex) {
 ex.printStackTrace();
 JOptionPane.*showMessageDialog*(frame, "Error loading data: " + ex.getMessage());
 }
 }
}

Додаток Є

Код класу Add – клас для додавання даних в таблиці БД

import javax.swing.\*;
import javax.swing.table.DefaultTableModel;
import java.awt.\*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Add {
 private JFrame frame;
 private String tableName;
 private String currentclass;
 private String currentlogin;
 private String windowTitle;
 private JTable table;
 private DefaultTableModel model;

 public Add(String tableName, String windowTitle, String currentclass, String currentlogin) {
 this.currentclass = currentclass;
 this.currentlogin = currentlogin;
 this.tableName = tableName;
 this.windowTitle = windowTitle + " ADD"; // Назва вікна
 createAndShowGUI();
 }

 private void createAndShowGUI() {
 frame = new JFrame(windowTitle);
 frame.setSize(600, 400);
 frame.setLayout(new BorderLayout());

 // Таблиця для введення нових даних
 model = new DefaultTableModel();
 table = new JTable(model);
 JScrollPane scrollPane = new JScrollPane(table);
 frame.add(scrollPane, BorderLayout.*CENTER*);

 // Кнопка Accept
 JButton acceptButton = new JButton("Accept");
 acceptButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 saveData(); // Зберігаємо нові дані
 }
 });
 frame.add(acceptButton, BorderLayout.*SOUTH*);

 // Кнопка Back
 JButton backButton = new JButton("Back");
 backButton.addActionListener(new ActionListener() {

**Продовження додатку Є**

Код класу Add – клас для додавання даних в таблиці БД

 @Override
 public void actionPerformed(ActionEvent e) {
 back();
 }
 });
 frame.add(backButton, BorderLayout.*NORTH*);

 // Завантажуємо структуру таблиці
 loadDataIntoTable();

 frame.setVisible(true);
 }

 private void back() {
 if ("HRWorkSpace".equals(currentclass)) {
 new HRWorkSpace(currentlogin);
 } else if ("StorageWorkSpace".equals(currentclass)) {
 new StorageWorkSpace(currentlogin);
 } else if ("PurchasingWorkSpace".equals(currentclass)) {
 new PurchasingWorkSpace(currentlogin);
 } else if ("BuildingWorkSpace".equals(currentclass)) {
 new PurchasingWorkSpace(currentlogin);
 } else {
 JOptionPane.*showMessageDialog*(frame, "Invalid operation");
 }
 frame.dispose();
 }

 private void loadDataIntoTable() {
 try (Connection connection = new JDBC().getConnection()) {
 String query = "SELECT \* FROM " + tableName + " LIMIT 1"; // Отримуємо лише перший рядок для структури
 PreparedStatement statement = connection.prepareStatement(query);
 ResultSet resultSet = statement.executeQuery();

 // Отримуємо метадані для таблиці (назви стовпців)
 int columnCount = resultSet.getMetaData().getColumnCount();
 for (int i = 2; i <= columnCount; i++) { // Починаємо з другого стовпця (пропускаємо ID)
 model.addColumn(resultSet.getMetaData().getColumnName(i)); // Назви стовпців
 }

 // Додаємо порожній рядок для введення нових даних
 Object[] emptyRow = new Object[columnCount - 1]; // Мінус один, бо пропустили ID
 model.addRow(emptyRow);
 } catch (SQLException ex) {
 ex.printStackTrace();
 JOptionPane.*showMessageDialog*(frame, "Error loading data: " + ex.getMessage());
 }
 }

**Продовження додатку Є**

Код класу Add – клас для додавання даних в таблиці БД

 private void saveData() {
 try (Connection connection = new JDBC().getConnection()) {
 // Підготовка SQL запиту для додавання нових даних
 StringBuilder queryBuilder = new StringBuilder("INSERT INTO " + tableName + " (");

 // Додаємо назви стовпців (окрім ID)
 int columnCount = model.getColumnCount();
 for (int i = 0; i < columnCount; i++) {
 queryBuilder.append(model.getColumnName(i)).append(", ");
 }
 queryBuilder.setLength(queryBuilder.length() - 2); // Видаляємо останню кому
 queryBuilder.append(") VALUES (");

 // Додаємо місце для значень (параметри ?)
 for (int i = 0; i < columnCount; i++) {
 queryBuilder.append("?, ");
 }
 queryBuilder.setLength(queryBuilder.length() - 2); // Видаляємо останню кому
 queryBuilder.append(")");

 // Підготовлений запит
 PreparedStatement insertStatement = connection.prepareStatement(queryBuilder.toString());

 // Встановлюємо значення для кожного стовпця (окрім ID)
 for (int col = 0; col < columnCount; col++) {
 insertStatement.setObject(col + 1, model.getValueAt(0, col)); // Беремо значення з першого рядка
 }

 // Виконуємо запит на додавання
 insertStatement.executeUpdate();

 JOptionPane.*showMessageDialog*(frame, "Data added successfully!");
 } catch (SQLException ex) {
 ex.printStackTrace();
 JOptionPane.*showMessageDialog*(frame, "Error saving data: " + ex.getMessage());
 }
 }
}

Додаток Ж

Код класу Edit – клас для додавання даних в таблиці БД

import javax.swing.\*;
import javax.swing.table.DefaultTableModel;
import java.awt.\*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Edit {
 private JFrame frame;
 private String tableName;
 private String windowTitle;
 private String currentclass;
 private String currentlogin;
 private JTable table;
 private DefaultTableModel model;

 public Edit(String tableName, String windowTitle, String currentclass, String currentlogin) {
 this.currentclass = currentclass;
 this.currentlogin = currentlogin;
 this.tableName = tableName;
 this.windowTitle = windowTitle + " edit"; // Назва вікна
 createAndShowGUI();
 }

 private void createAndShowGUI() {
 frame = new JFrame(windowTitle);
 frame.setSize(600, 400);
 frame.setLayout(new BorderLayout());

 // Таблиця для відображення даних
 model = new DefaultTableModel();
 table = new JTable(model);
 JScrollPane scrollPane = new JScrollPane(table);
 frame.add(scrollPane, BorderLayout.*CENTER*);

 // Кнопка Accept
 JButton acceptButton = new JButton("Accept");
 acceptButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 saveData(); // Зберігаємо відредаговані дані
 }
 });
 frame.add(acceptButton, BorderLayout.*SOUTH*);

 // Кнопка Back
 JButton backButton = new JButton("Back");
 backButton.addActionListener(new ActionListener() {

**Продовження додатку Ж**

Код класу Edit – клас для додавання даних в таблиці БД

 @Override
 public void actionPerformed(ActionEvent e) {
 back();
 }
 });
 frame.add(backButton, BorderLayout.*NORTH*);

 // Завантажуємо дані в таблицю
 loadDataIntoTable();
 frame.setVisible(true);
 }

 private void back() {
 if ("HRWorkSpace".equals(currentclass)) {
 new HRWorkSpace(currentlogin);
 } else if ("StorageWorkSpace".equals(currentclass)) {
 new StorageWorkSpace(currentlogin);
 } else if ("PurchasingWorkSpace".equals(currentclass)) {
 new PurchasingWorkSpace(currentlogin);
 } else if ("BuildingWorkSpace".equals(currentclass)) {
 new PurchasingWorkSpace(currentlogin);
 } else {
 JOptionPane.*showMessageDialog*(frame, "Invalid operation");
 }
 frame.dispose();
 }

 private void loadDataIntoTable() {
 try (Connection connection = new JDBC().getConnection()) {
 String query = "SELECT \* FROM " + tableName;
 PreparedStatement statement = connection.prepareStatement(query);
 ResultSet resultSet = statement.executeQuery();

 // Отримуємо метадані для таблиці
 int columnCount = resultSet.getMetaData().getColumnCount();
 for (int i = 1; i <= columnCount; i++) {
 model.addColumn(resultSet.getMetaData().getColumnName(i)); // Назви стовпчиків
 }

 // Додаємо дані
 while (resultSet.next()) {
 Object[] rowData = new Object[columnCount];
 for (int i = 0; i < columnCount; i++) {
 rowData[i] = resultSet.getObject(i + 1);
 }
 model.addRow(rowData);
 }
 } catch (SQLException ex) {
 ex.printStackTrace();
 JOptionPane.*showMessageDialog*(frame, "Error loading data: " + ex.getMessage());

**Продовження додатку Ж**

Код класу Edit – клас для додавання даних в таблиці БД

 }
 }

 private void saveData() {
 try (Connection connection = new JDBC().getConnection()) {
 // Підготовка SQL запиту для оновлення даних
 String updateQuery = "UPDATE " + tableName + " SET ";

 // Додаємо стовпці до запиту (окрім ID, який вважається першим стовпцем)
 StringBuilder queryBuilder = new StringBuilder(updateQuery);
 int columnCount = model.getColumnCount();
 for (int i = 1; i < columnCount; i++) {
 queryBuilder.append(model.getColumnName(i)).append(" = ?, ");
 }
 queryBuilder.setLength(queryBuilder.length() - 2); // Видаляємо останню кому
 queryBuilder.append(" WHERE ").append(model.getColumnName(0)).append(" = ?");

 // Підготовлений запит для оновлення кожного рядка
 PreparedStatement updateStatement = connection.prepareStatement(queryBuilder.toString());

 // Проходимо по кожному рядку таблиці
 for (int row = 0; row < model.getRowCount(); row++) {
 // Встановлюємо значення для кожного стовпця (окрім ID)
 for (int col = 1; col < columnCount; col++) {
 updateStatement.setObject(col, model.getValueAt(row, col));
 }
 // Встановлюємо значення для ID в WHERE (перший стовпець)
 updateStatement.setObject(columnCount, model.getValueAt(row, 0));

 // Виконуємо запит для кожного рядка окремо
 updateStatement.executeUpdate();
 }

 JOptionPane.*showMessageDialog*(frame, "Data updated successfully!");
 } catch (SQLException ex) {
 ex.printStackTrace();
 JOptionPane.*showMessageDialog*(frame, "Error saving data: " + ex.getMessage());
 }
 }
}

Додаток З

Код класу Search – клас для пошуку даних в таблиці БД

import javax.swing.\*;
import javax.swing.table.DefaultTableModel;
import java.awt.\*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Search {
 private JFrame frame;
 private String tableName;
 private String windowTitle;
 private String currentlogin;
 private String currentclass;
 private JTable table;
 private DefaultTableModel model;
 private JTextField searchField;
 private JComboBox<String> columnComboBox;

 public Search(String tableName, String windowTitle, String currentclass, String currentlogin) {
 this.tableName = tableName;
 this.currentclass = currentclass;
 this.currentlogin = currentlogin;
 this.windowTitle = windowTitle + " Search"; // Назва вікна
 createAndShowGUI();
 }

 private void createAndShowGUI() {
 frame = new JFrame(windowTitle);
 frame.setSize(600, 400);
 frame.setLayout(new BorderLayout());

 // Панель для пошуку
 JPanel searchPanel = new JPanel();
 searchPanel.setLayout(new FlowLayout());

 // Поле для введення тексту
 searchPanel.add(new JLabel("Text:"));
 searchField = new JTextField(20);
 searchPanel.add(searchField);

 // Випадковий список колонок
 columnComboBox = new JComboBox<>();
 columnComboBox.addItem("All Fields"); // Перший пункт - всі поля
 loadColumnsIntoComboBox(); // Завантажуємо назви колонок
 searchPanel.add(columnComboBox);

 // Кнопка Search
 JButton searchButton = new JButton("Search");
 searchButton.addActionListener(e -> searchData());
 searchPanel.add(searchButton);

**Продовження додатку** З

Код класу Search – клас для пошуку даних в таблиці БД

 // Кнопка Back
 JButton backButton = new JButton("Back");
 backButton.addActionListener(new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 back();
 }
 });
 frame.add(backButton, BorderLayout.*SOUTH*);

 frame.add(searchPanel, BorderLayout.*NORTH*);

 // Таблиця для відображення результатів
 model = new DefaultTableModel();
 table = new JTable(model);
 JScrollPane scrollPane = new JScrollPane(table);
 frame.add(scrollPane, BorderLayout.*CENTER*);

 frame.setVisible(true);
 }
 private void back() {
 if ("HRWorkSpace".equals(currentclass)) {
 new HRWorkSpace(currentlogin);
 } else if ("StorageWorkSpace".equals(currentclass)) {
 new StorageWorkSpace(currentlogin);
 } else if ("PurchasingWorkSpace".equals(currentclass)) {
 new PurchasingWorkSpace(currentlogin);
 } else if ("BuildingWorkSpace".equals(currentclass)) {
 new BuildingWorkSpace(currentlogin);
 } else {
 JOptionPane.*showMessageDialog*(frame, "Invalid operation");
 }
 frame.dispose();
 }
 private void loadColumnsIntoComboBox() {
 try (Connection connection = new JDBC().getConnection()) {
 String query = "SELECT \* FROM " + tableName + " LIMIT 1"; // Отримуємо перший рядок для структури
 PreparedStatement statement = connection.prepareStatement(query);
 ResultSet resultSet = statement.executeQuery();

 // Отримуємо назви колонок (окрім ID)
 int columnCount = resultSet.getMetaData().getColumnCount();
 for (int i = 2; i <= columnCount; i++) { // Починаємо з другого стовпця
 columnComboBox.addItem(resultSet.getMetaData().getColumnName(i)); // Додаємо назви колонок до комбобоксу
 }
 } catch (SQLException ex) {
 ex.printStackTrace();
 JOptionPane.*showMessageDialog*(frame, "Error loading columns: " + ex.getMessage());

**Продовження додатку** З

Код класу Search – клас для пошуку даних в таблиці БД

 }
 }

 private void searchData() {
 String searchText = searchField.getText();
 String selectedColumn = (String) columnComboBox.getSelectedItem();

 // Формуємо запит на основі вибраної колонки
 String query;
 if ("All Fields".equals(selectedColumn)) {
 query = "SELECT \* FROM " + tableName + " WHERE ";
 query += "CONCAT\_WS(' ', " + getAllColumnNames() + ") LIKE ?";
 } else {
 query = "SELECT \* FROM " + tableName + " WHERE " + selectedColumn + " LIKE ?";
 }

 try (Connection connection = new JDBC().getConnection();
 PreparedStatement statement = connection.prepareStatement(query)) {
 statement.setString(1, "%" + searchText + "%");
 ResultSet resultSet = statement.executeQuery();

 // Очищаємо таблицю перед новими даними
 model.setRowCount(0);
 model.setColumnCount(0); // Очищуємо старі колонки

 // Додаємо нові дані
 int columnCount = resultSet.getMetaData().getColumnCount();
 for (int i = 1; i <= columnCount; i++) {
 model.addColumn(resultSet.getMetaData().getColumnName(i)); // Додаємо назви стовпців
 }

 // Додаємо рядки
 while (resultSet.next()) {
 Object[] rowData = new Object[columnCount];
 for (int i = 0; i < columnCount; i++) {
 rowData[i] = resultSet.getObject(i + 1);
 }
 model.addRow(rowData);
 }
 } catch (SQLException ex) {
 ex.printStackTrace();
 JOptionPane.*showMessageDialog*(frame, "Error searching data: " + ex.getMessage());
 }
 }

**Продовження додатку** З

Код класу Search – клас для пошуку даних в таблиці БД

 private String getAllColumnNames() {
 StringBuilder columnNames = new StringBuilder();
 try (Connection connection = new JDBC().getConnection()) {
 String query = "SELECT \* FROM " + tableName + " LIMIT 1";
 PreparedStatement statement = connection.prepareStatement(query);
 ResultSet resultSet = statement.executeQuery();

 int columnCount = resultSet.getMetaData().getColumnCount();
 for (int i = 2; i <= columnCount; i++) { // Пропускаємо ID
 columnNames.append(resultSet.getMetaData().getColumnName(i)).append(", ");
 }
 columnNames.setLength(columnNames.length() - 2); // Видаляємо останню кому
 } catch (SQLException ex) {
 ex.printStackTrace();
 JOptionPane.*showMessageDialog*(frame, "Error getting column names: " + ex.getMessage());
 }
 return columnNames.toString();
 }
}

Додаток И

Код класу BuildersList – клас для роботи з персоналом (Будівельниками)

import javax.swing.\*;
import javax.swing.table.DefaultTableModel;
import java.awt.\*;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

public class BuildersList {

 private JFrame frame;
 private JTable table;
 private DefaultTableModel model;
 private String selectedOption;
 private String currentClass;
 private String currentLogin;

 public BuildersList(String selectedOption, String currentClass, String currentLogin) {
 this.selectedOption = selectedOption;
 this.currentClass = currentClass;
 this.currentLogin = currentLogin;
 createAndShowGUI();
 }

 private void createAndShowGUI() {
 frame = new JFrame("Builders List");
 frame.setSize(800, 500);
 frame.setLayout(new BorderLayout());

 // Створюємо модель таблиці з чекбоксами
 model = new DefaultTableModel() {
 @Override
 public Class<?> getColumnClass(int column) {
 return column == 0 ? Boolean.class : String.class;
 }
 };

 // Таблиця для відображення даних
 table = new JTable(model);
 JScrollPane scrollPane = new JScrollPane(table);
 frame.add(scrollPane, BorderLayout.CENTER);

 // Панель управління під таблицею
 JPanel controlPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));
 JLabel moveToLabel = new JLabel("Move to:");
 JComboBox<String> comboBox = new JComboBox<>(new String[]{"Object1", "Object2", "Object3", "notinvolved"});
 JButton acceptButton = new JButton("Accept");

**Продовження додатку** И

Код класу BuildersList – клас для роботи з персоналом (Будівельниками)

 controlPanel.add(moveToLabel);
 controlPanel.add(comboBox);
 controlPanel.add(acceptButton);
 frame.add(controlPanel, BorderLayout.SOUTH);

 // Кнопка Back
 JButton backButton = new JButton("Back");
 backButton.addActionListener(e -> back());
 frame.add(backButton, BorderLayout.NORTH);

 // Викликаємо відповідний метод залежно від вибраної опції
 switch (selectedOption) {
 case "NOT Involved Builders":
 NOTInvolvedBuilders();
 break;
 case "Involved Builders":
 InvolvedBuilders();
 break;
 case "All Builders":
 ALLBuilders();
 break;
 default:
 JOptionPane.showMessageDialog(frame, "Invalid operation");
 break;
 }

 // Додаємо дію до кнопки Accept
 acceptButton.addActionListener(e -> {
 String selectedValue = (String) comboBox.getSelectedItem();
 moveSelectedBuilders(selectedValue);
 });

 frame.setVisible(true);
 }
 private void back() {

 // Логіка для повернення до попереднього вікна
 if ("HRWorkSpace".equals(currentClass)) {
 new HRWorkSpace(currentLogin);
 } else if ("StorageWorkSpace".equals(currentClass)) {
 new StorageWorkSpace(currentLogin);
 } else if ("PurchasingWorkSpace".equals(currentClass)) {
 new PurchasingWorkSpace(currentLogin);
 } else if ("BuildingWorkSpace".equals(currentClass)) {
 new BuildingWorkSpace(currentLogin);
 } else {
 JOptionPane.showMessageDialog(frame, "Invalid operation");
 }
 frame.dispose();
 }

**Продовження додатку** И

Код класу BuildersList – клас для роботи з персоналом (Будівельниками)

 // Метод для виведення всіх будівельників
 public void ALLBuilders() {
 loadDataIntoTable("SELECT \* FROM allbuilders");
 }

 // Метод для виведення будівельників, які залучені
 public void InvolvedBuilders() {
 loadDataIntoTable("SELECT \* FROM allbuilders WHERE involvedin != 'notinvolved'");
 }

 // Метод для виведення будівельників, які не залучені
 public void NOTInvolvedBuilders() {
 loadDataIntoTable("SELECT \* FROM allbuilders WHERE involvedin = 'notinvolved'");
 }

 private void loadDataIntoTable(String query) {
 try (Connection connection = new JDBC().getConnection()) {
 PreparedStatement statement = connection.prepareStatement(query);
 ResultSet resultSet = statement.executeQuery();

 // Очищаємо попередні дані
 model.setRowCount(0);
 model.setColumnCount(0);

 // Додаємо колонку для чекбоксів
 model.addColumn("Select");

 // Додаємо колонки з метаданих
 int columnCount = resultSet.getMetaData().getColumnCount();
 for (int i = 1; i <= columnCount; i++) {
 model.addColumn(resultSet.getMetaData().getColumnName(i));
 }

 // Додаємо дані
 while (resultSet.next()) {
 Object[] rowData = new Object[columnCount + 1];
 rowData[0] = false; // Чекбокс не обраний за замовчуванням
 for (int i = 1; i <= columnCount; i++) {
 rowData[i] = resultSet.getObject(i);
 }
 model.addRow(rowData);
 }
 } catch (SQLException ex) {
 ex.printStackTrace();
 JOptionPane.showMessageDialog(frame, "Error loading data: " + ex.getMessage());
 }
 }

 private void moveSelectedBuilders(String newInvolvedValue) {
 ArrayList<Integer> selectedRows = new ArrayList<>();

**Продовження додатку** И

Код класу BuildersList – клас для роботи з персоналом (Будівельниками)

 // Отримуємо вибрані рядки
 for (int i = 0; i < table.getRowCount(); i++) {
 if ((Boolean) table.getValueAt(i, 0)) {
 selectedRows.add((Integer) table.getValueAt(i, 1)); // Передбачається, що ID у другій колонці
 }
 }

 if (selectedRows.isEmpty()) {
 JOptionPane.showMessageDialog(frame, "No builders selected.");
 return;
 }

 try (Connection connection = new JDBC().getConnection()) {
 String updateQuery = "UPDATE allbuilders SET involvedin = ? WHERE id = ?";
 PreparedStatement statement = connection.prepareStatement(updateQuery);

 for (int builderId : selectedRows) {
 statement.setString(1, newInvolvedValue);
 statement.setInt(2, builderId);
 statement.executeUpdate();
 }

 JOptionPane.showMessageDialog(frame, "Successfully moved!");
 loadDataIntoTable("SELECT \* FROM allbuilders"); // Оновлюємо дані після змін
 } catch (SQLException ex) {
 ex.printStackTrace();
 JOptionPane.showMessageDialog(frame, "Error updating data: " + ex.getMessage());
 }
 }
}