Вінницький національний технічний університет

Факультет комп'ютерних систем і автоматики

Кафедра системного аналізу, комп'ютерного моніторингу

та інженерної графіки

ІНФОРМАЦІЙНА АВТОМАТИЗОВАНА СИСТЕМА РОБОЧИХ МІСЦЬ ДЛЯ СПІВРОБІТНИКІВ СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ

Пояснювальна записка до магістерської кваліфікаційної роботи

Виконав: студент 2 курсу, групи ІСТ-18м

спеціальності 126 – «Інформаційні системи та технології»

Дячук А.Ю.

Керівник: к.т.н., доц. Козачко О. М.

Рецензент: доц. Ковтун В.В.

Вінниця ВНТУ – 2019 року

РЕФЕРАТ

Магістерська кваліфікаційна робота: 147 ст., 12 табл., 35 рис., 20 джерел.

Об’єкт досліджень – процес автоматизації управління та процесу призначення задач під час роботи станції технічного обслуговування.

Мета роботи – пришвидшення та оптимізація робочого процесу на станціях технічного обслуговування за рахунок математичних моделей процесу призначення задач, які враховують складність задач та кваліфікацію робітників.

Здійснено порівняльний аналіз існуючих систем для автоматизації робочого процесу на станціях технічного обслуговування в Україні. Проведено порівняльний аналіз алгоритмів для вирішення задачі про доручення задач виконавцям та визначено оптимальні для інтеграції в систему автоматизованих робочих місць для співробітників станції технічного обслуговування. Визначено оптимальні технології для реалізації веб-системи (її бази даних, серверної та клієнтської частин). Змодельовано задачу про призначення для вирішення її Угорським та Генетичним алгоритмами.

Прогнозні припущення про розвиток об’єкта дослідження – розробка інформаційної моделі автоматизованої системи співробітників станції технічного обслуговування, яка на відмінну від існуючих, здійснює миттєву взаємодію між програмними модулями системи за рахунок розробленого API, яка забезпечує миттєве зчитування/запис інформації до бази даних.

Галузь застосування – сервіси технічного обслуговування та діагностичні центри для автомобілів по всій Україні.

ІНФОРМАЦІЙНА АВТОМАТИЗОВАНА СИСТЕМА РОБОЧИХ МІСЦЬ, ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, ЗАДАЧА ПРО ПРИЗНАЧЕННЯ, ГЕНЕТИЧНИЙ АЛГОРИТМ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, АВТОМАТИЗАЦІЯ СТАНЦІЙ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ.

ABSTRACT

Master's qualification work: 147 pages, 12 tables, 35 pictures, 20 sources.

The object of research – the process of automation of management and task assignment during the operation of a maintenance station.

The purpose of the work – to accelerate and optimize the work process at the service stations due to mathematical models of the task assignment process, which takes into account the complexity of the tasks and the qualification of the workers.

Comparative analysis of existing systems for workflow automation at service stations in Ukraine is carried out. A comparative analysis of the algorithms for solving the task of assigning tasks to the contractors was performed and the optimal ones for integration into the system of automated workplaces for the employees of the service station were determined. The optimal technologies for realization of the web system (its database, server and client parts) are determined. The assignment problem for solving its Hungarian and Genetic algorithms is modeled.

Estimated assumptions about the development of the object of study - the development of an information model of an automated system of employees of the service station, which, unlike the existing ones, instantaneously interacts with the program modules of the system due to the developed API, which provides instantaneous reading / writing of information to the database.

Scope of application – maintenance services and diagnostic centers for vehicles throughout Ukraine.

INFORMATIVE AUTOMATED SYSTEM OF WORKPLACES, INFORMATION TECHNOLOGY, TASK OF PURPOSE, GENETIC ALGORITHM, CLIENT-SERVER, ARCHITECTURE, ARCHITECTURE

ЗМІСТ

ВСТУП

В сучасному сьогоденні широко постає питання автоматизації управління та моніторингу усього робочого процесу не тільки на станціях технічного обслуговування а й на підприємствах в цілому.

Основною вимогою при розробці інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування є забезпечення можливості використовувати систему за допомогою будь якого пристрою з доступом в інтернет.

Об'єктом дослідження є процес автоматизації управління та процесу призначення завдань під час роботи станції технічного обслуговування.

Предметом дослідження є програмні засоби автоматизації управління та процесу призначення завдань під час роботи станції технічного обслуговування.

Метою дослідження є пришвидшення та оптимізація робочого процесу на станціях технічного обслуговування за рахунок математичних моделей процесу призначення задач, яка враховує складність задач та кваліфікацію робітників.

Існує кілька систем які надають певні можливості пришвидшити процес створення замовлення з подальшою можливість управління та призначення. Однак, жоден з них не можна назвати системою з повноцінними автоматизованими робочими місцями для кожного співробітника станцій технічного обслуговування.

Аналіз показав, що подібні системи мають такі основні недоліки: по-перше, відсутність так званих власних сторінок (робочих місць) для співробітників, де вони могли б займатись управлінням своїх задач, а їх директори – моніторингом та контролем виконання; по-друге, відсутність функції автоматичного призначення задач для виконавців (опираючись на кваліфікації співробітників та рівень складності задач), що б спростило робочий процес та стало важливим фактором в прискоренні виконання

замовлень; по-третє, є складнощі з простим формуванням замовлення та збереження усієї необхідної інформації в єдиній системі із сховищем даних.

Для усунення цих недоліків слід розробити алгоритм автоматизованого призначення роботи для співробітників СТО, використавши один із алгоритмів пошуку та оптимізації як базовий. Розроблений алгоритм інтегрувати в автоматизоване робоче місце співробітників СТО, де і буде здійснюватися менеджмент задач кожного робітника.

Отже, розробка автоматизованого робочого місця для співробітників з використанням алгоритму автоматичного призначення задач на СТО, є актуальною.

Наукова новизна одержаних результатів. Основні результати, які були отримані в процесі вирішення поставлених завдань та становлять наукову новизну дослідження, полягають в наступному:

1. Подальшого розвитку набули моделі автоматизованого процесу призначення задач для співробітників СТО на базі генетичного та угорського алгоритмів, які на відміну від існуючих, враховують одночасно, складність задачі, час виконання задач та кваліфікацію робітників.

2. Розроблено інформаційну модель автоматизованої системи співробітників станції технічного обслуговування, яка на відмінну від існуючих, здійснює миттєву взаємодію між програмними модулями системи за рахунок розробленого API, яка забезпечує миттєве зчитування/запис інформації до бази даних.

Практичне значення одержаних результатів полягає у наступному:

– розроблено та реалізовано алгоритм автоматизованого призначення задач для співробітників станції технічного обслуговування на базі генетичного та угорського алгоритмів;

– розроблено структуру та реалізовано базу даних для оптимального представлення усієї інформації предметної області;

– розроблено API з можливістю швидкого зчитування/запису інформації до бази даних та зручним обміном цими даними з клієнтською частиною;

– розроблено крос-платформну клієнтську частину зі зручним та зрозумілим інтерфейсом, для швидкої взаємодії користувача із системою та її базою даних в режимі реального часу.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням математичних методів під час доведення наукових положень, строгим виведенням аналітичних співвідношень, порівнянням результатів з відомими, та збіжністю результатів математичного моделювання з результатами, що отримані під час впровадження розробленого програмного засобу.

Апробація результатів роботи. Результати роботи були апробовані на всеукраїнській науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи 2019».

Публікації. За результатами магістерської кваліфікаційної роботи опубліковано на сайті всеукраїнської науково-практичної Інтернет-конференції ВНТУ [1].

1 ОГЛЯД ТЕХНІЧНИХ ТА ПРОГРАМНИХ ЗАСОБІВ РОЗРОБКИ ІНФОРМАЦІЙНОЇ АВТОМАТИЗОВАНОЇ СИСТЕМИ РОБОЧИХ МІСЦЬ ДЛЯ СПІВРОБІТНИКІВ СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ

1.1 Аналіз аналогічних рішень

Доволі популярним рішенням проблеми управління, моніторингу та систематизації робочого процесу на підприємстві є 1С: Підприємство.

Система "1С: Підприємство" – це автоматизована інформаційна система обліку на підприємстві з розширеними можливостями систем 3-го покоління. Вона складається з окремих модулів, які можуть працювати разом. Їх можливості викладені в наступних характеристиках:

- бухгалтерський облік дозволяє реалізувати будь-яку схему облік залежно від конфігурації;
- оперативний облік призначений для обліку наявності і руху матеріальних цінностей і коштів;
- розрахунок дозволяє реалізувати будь-яку схему розрахунку зарплатні залежно від конфігурації.

Головні модулі є технологічними блоками, що реалізують загальну функціональність, яка залежить від технічних особливостей апаратного та програмного забезпечення. Ці модулі здійснюють також підтримку так званих "конфігурацій" і є базою для виконання програм на вбудованій мові програмування. Конфігурація є множиною ключових змінних, таких, що статично характеризують підприємство та його діяльність (код реєстрації, ставка ПДВ, план рахунків, нормативні показники тощо), та "бізнес-правил" у вигляді програм на вбудованій мові високого рівня, які характеризують бізнес-процеси підприємства в динаміці.

Приклад вікна системи "1С: Підприємство" відображений на рисунку 1.1.

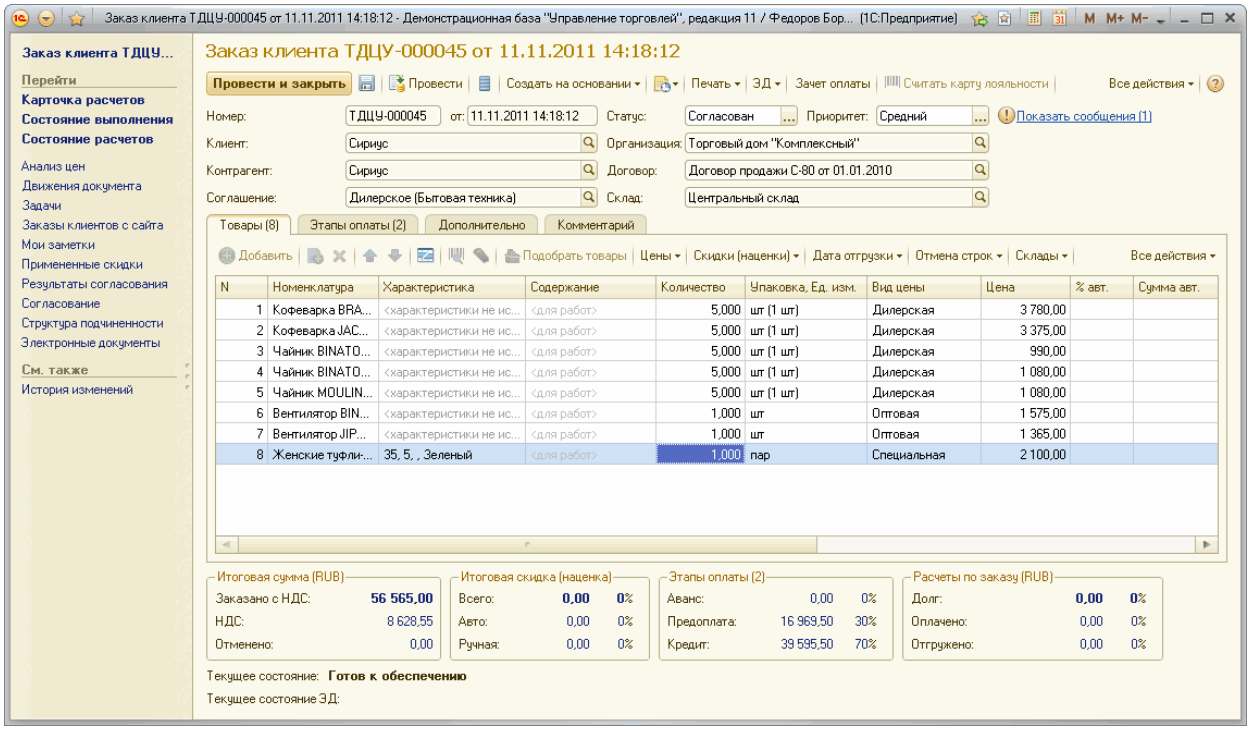


Рисунок 1.1 – Приклад вікна системи "1С: Підприємство"

Не зважаючи на усі перечисленні переваги системи, щоб використовувати такий сервіс підприємству необхідно мати персональні комп’ютери необхідної потужності для коректної роботи програми. Такий підхід суперечить основним вимогам до системи: доступність для використання за допомогою будь яких гаджетів з доступом в Інтернет [2].

Ще одним аналогом для вирішення даної проблеми є сервіс «autobooking.com».

Система “autobooking.com” – являє собою веб рішення, що є сервісом для запису до найкращих станції технічного обслуговування для автомобіля.

Можливості даної системи:

– Вибрати найкращий СТО по індивідуальним параметрам;
– Дізнатися про якість роботи СТО на основі відгуків реальних клієнтів;
– Подати заявку на СТО в зручний час.

Одне з вікон сервісу «autobooking.com» відображено на рисунку 1.2.

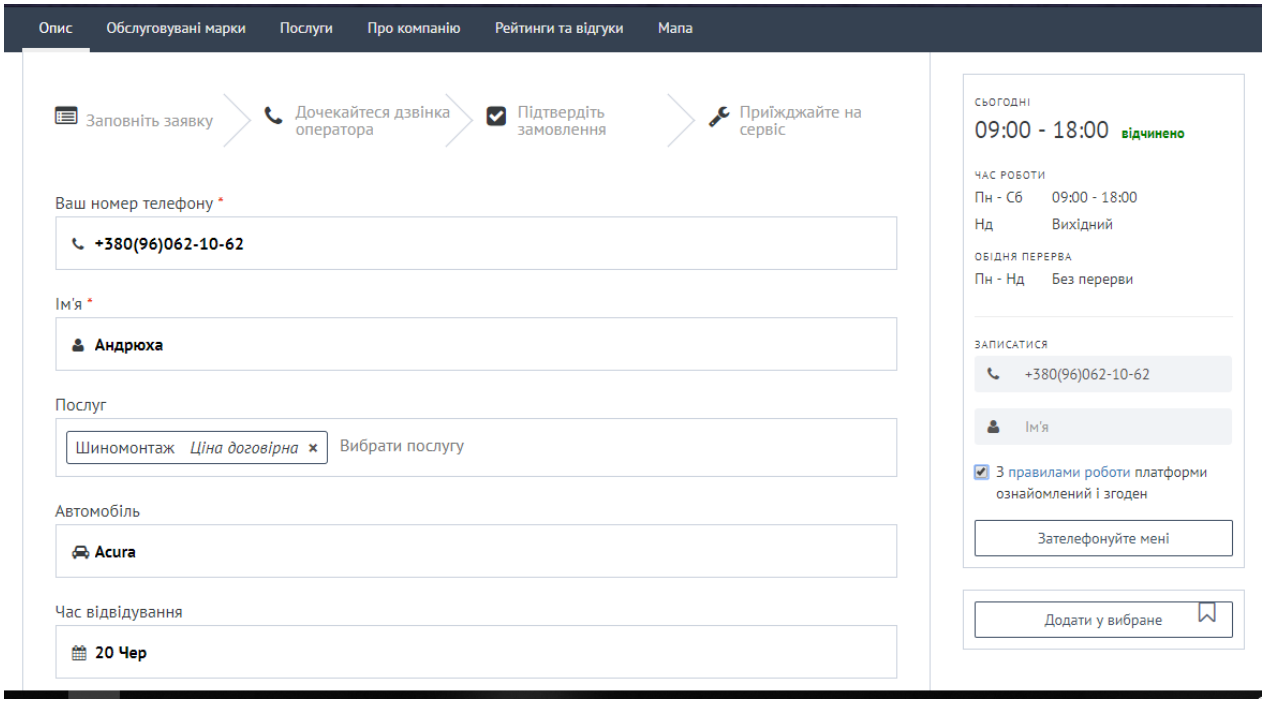


Рисунок 1.2. – Одне з вікон сервісу «autobooking.com»

Не дивлячись на зручність використання даного сервісу, його функціонал описує лише процес подачі замовлення без можливості відслідковування клієнтом процесу його виконання. Також, великим недоліком є відсутність будь якого управління станцією технічного обслуговування з точки зору менеджерів чи інших працівників сервісу.

1.2 Обґрунтування і вибір методів та інструментів

Інформаційна автоматизована система робочих місць буде включати в себе функціонал управління та моніторингу великої кількості різних моделей даних, таких як: користувачі, деталі автомобіля, замовлення, задачі, статистика, марки автомобіля, моделі автомобіля, та ін. Для реалізації даного сервісу найоптимальнішим підходом буде розробка веб-системи з клієнт-серверною архітектурою, що дасть клієнту свободу вибору пристрою для

роботи із системою і не вимагатиме певних жорстко визначених апаратних ресурсів.

Весь функціонал роботи з моделями даних, їх зберігання та обробка, а також процес проведення алгоритмічних розрахунків буде виконуватись на серверній частині. Для зберігання даних системи зручно буде використати реляційну базу даних, а взаємодію сервера з нею організувати із використанням ORM, що пришвидшить процес реалізації структури бази даних на мові JavaScript. Клієнтська частина буде відповідати тільки за зручне відображення, даних, реактивність системи та можливість взаємодії з нею з різних девайсів в режимі реального часу.

1.2.1 Вибір методу вирішення задачі про призначення

Основним ядро роботи інформаційної автоматизованої системи робочих місць для співробітників СТО буде розподіл задач різної складності та характеру між інженерами різної кваліфікації, швидкості виконання та рівня досвіду. Дана проблема є класичним прикладом задачі про призначення, вирішення якої і буде основним процесом розроблювальної системи. Задача про призначення є однією із базових задач комбінаторної оптимізації в галузі дослідження операцій в математиці. ЇЇ основною є пошук парування мінімальної (або максимальної) ваги між елементами двох скінченних множин. Для розв’язання даної задачі можуть бути застосовані різні методи – від загальних – із лінійного програмування, до більш складних – на основі графів [4].

Симплекс-метод – це один із методів розв’язання задачі лінійного програмування, в основі якого є проведення скорегованого руху по опорних планах до знаходження оптимального розв’язку [5].

Також, добре відомим методом вирішення задачі про призначення є алгоритм аукціону. Цей алгоритм використовується у бізнес рішеннях для

пошуку найкращих цін на певні продукти та товари, що пропонуються масиву покупців [6].

Одним із найпоширеніших методів розв’язання задачі про призначення є Угорський алгоритм. Це алгоритм оптимізації, який розв’язує задачу про призначення за поліноміальний час і який передує симплекс-методу [7].

Ядром роботи Угорського алгоритму є моделювання задачі про призначення та побудова матриці відповідності та основі цієї моделі. Кожна комірка відповідає певному значенню взаємодії між виконавцем та задачею. При чому, це значення може бути коефіцієнтом що є унікальним для кожної комбінації та враховувати ефективність робітника, час його роботи чи необхідну кваліфікацію, а також складність задачі та тривалість її виконання. За потреби, можна визначити який із критеріїв є важливішим, що робить такий підхід доволі зручним для застосування в різного роду задачах.

Аналізуючи методи для впровадження в інформаційну автоматизовану систему робочих місць, слід згадати також про Генетичний алгоритм.

Генетичним алгоритмом називають еволюційні алгоритм пошуку, що використовується для вирішення задач оптимізації і модулювання шляхом послідовного підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію. Особливістю такого алгоритму є використання операторів «схрещення» та «мутації» які виконують рекомбінацію і дія яких аналогічна ролі в живій природі [8].

Особливістю Генетичного алгоритму є його ефективність при великому розмірі задачі оптимізації, що добре підходить для системи в часи високої завантаженості, коли є велика кількість задач та виконавців. Угорський, на відмінну, ефективніший при малих розмірах початкової матриці – що буде оптимально використати в проміжки не великих навантажень на систему, наприклад в нічний час добри, коли клієнтів на СТО не так багато.

Опираючись на таку залежність, ефективним рішенням буде використовувати Угорський алгоритм поки інженерів та задач не велика

кількість. А в певний момент часу перемикати алгоритм автоматичного розподіл задач як це відображено на рисунку 1.3

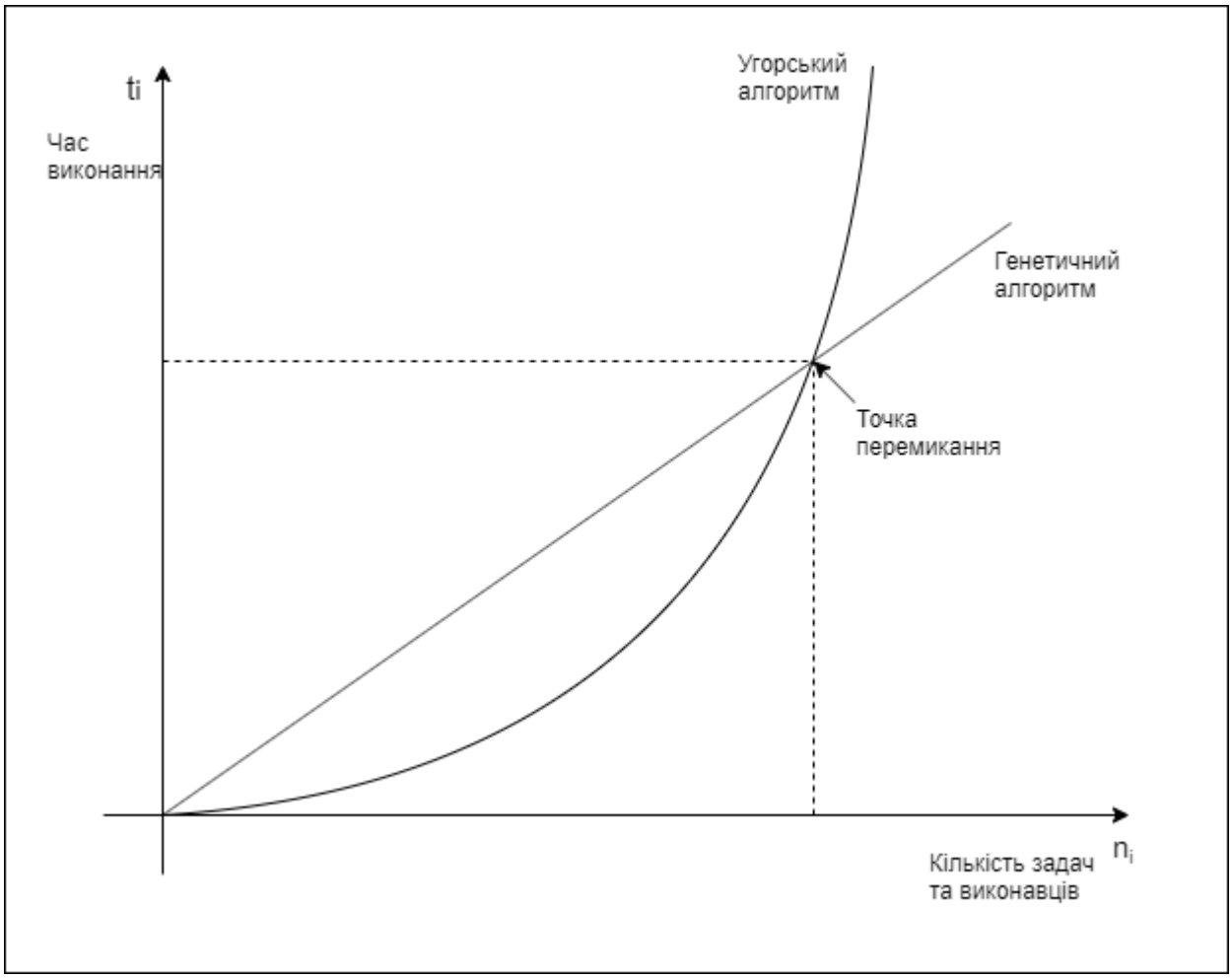


Рисунок 1.3 – Час роботи алгоритмів за умови зростання задач та виконавців

Отже, для оптимального призначення задач між інженерами на станції технічного обслуговування в інформаційній автоматизованій системі робочих місць доцільно буде використати Генетичний алгоритм в часи великих навантажень та Угорських в часи помірної завантаженості, зважаючи на їх наступні переваги над суміжними підходами:

- Відносна простота реалізації у порівняні з іншими алгоритмами;
- Швидкість отримання результату після спрацювання алгоритмів є однією найвищих з усіх проаналізованих;

- Відносно краща сумісність алгоритмів із контекстом даної проблеми розподілу задач між співробітниками.

Отже, задачу про призначення в контексті розподілу задач в інформаційній автоматизованій системі робочих місць доцільно буде вирішувати використовуючи Генетичний та Угорський алгоритми.

1.2.2 Вибір засобів реалізації серверної частини

Оскільки, однією із цілей магістерської кваліфікаційної роботи є – розробити API з можливістю швидкого зчитування/запису інформації до бази даних та зручним обміном цими даними з клієнтською частиною, доцільно використати Node.js для написання серверу.

Бібліотека Node.js є платформою з відкритим кодом для написання серверних додатків. В її основу лягло подієво-орієнтоване та асинхронне програмування з неблокуючим введенням/виведенням інформації.

Платформа призначена для виконання веб-додатків та мові програмування JS. Для виконання коду використовується двигун V8 від Google [9].

Зважаючи на те, що при розробці системи треба буде реалізувати аж 6 модулів (автоматизованих робочих місць), до складу яких будуть входити відповідні роутери, контролери та моделі, буде зручно використати фреймворк Express,js для пришвидшення написання серверу. Express.js – це пакет розробки веб-застосунків для Node.js, реалізований як вільне і відкрите програмне забезпечення. Він спроектований для створення веб-застосунків і API. Класично є стандартним пакетом для Node.js [10].

1.2.3 Вибір засобів організації та реалізації бази даних

Для забезпечення надійного, швидкого і зручного введення, редагування, видалення і маніпулювання інформацією в автоматизованій системі робочих місць для співробітників станції технічного обслуговування можна використовувати сучасну базу даних. Використання такої бази даних та системи управління нею значно скоротить час виконання відповідних операцій, що раніше виконувались вручну.

База даних – іменована сукупність даних, яка відображає стани об’єктів та їхніх співвідношень у предметній галузі, що розглядається. Альтернативне визначення бази даних — це набір взаємопов’язаних даних та їхніх описів, що використовуються спільно і призначені для задоволення інформаційних потреб користувачів [11].

Для реалізації бази даних модульної інформаційної системи для сервісу технічного обслуговування доцільно використовувати реляційну модель даних.

У реляційній моделі даних об'єкти і взаємозв'язки між ними представляються за допомогою таблиць. Взаємозв'язки також подаються як об'єкти. Кожна таблиця представляє один об'єкт і складається з рядків і стовпців. Перевагами реляційної моделі є незалежність від фізичного рівня представлення, зручність і розуміння організації даних користувачами, можливість розширення бази приєднанням нових елементів й записів без зміни при цьому існуючих підсхем та прикладних програм [12].

Для бази даних системи буде доцільно використати СУБД PostgreSQL. PostgreSQL – найбільш повнофункціональна, вільно поширювана об’єктно-реляційна СУБД з відкритим кодом [13]. Основні можливості СУБД:

- Об’єктно-реляційна модель;
- Простота рішення;
- Повноцінна підтримка стандарту SQL;
- Гнучкість API [14].

В нашій ситуації буде краще використати саме її на відмінну від, наприклад, MySQL, бо:

- PostgreSQL підтримує тип Boolean, який буде необхідний при проектуванні моделей;
- Хмарний сервіс Heroku, що буде використаний як хостинг, підтримує PosqtgreSQL як СУБД для клієнт-серверних додатків.

1.2.4 Вибір ORM

Одним з найпопулярніших рішень для взаємодії сервера з СУБД було б використати SQL. SQL – символізує собою структуровану мову запитів. Елегантність і незалежність від специфіки комп'ютерних технологій, а також його підтримка лідерами промисловості в області технології реляційних баз даних, зробило SQL основною стандартною мовою [15].

Проте, для пришвидшення написання запитів до бази даних та уникнення SQL ін’єкцій на серверній частині буде доцільно використовувати ORM.

Об'єктно-реляційна модель (ORM) є автоматизованим прозорим представленням-відношенням об'єктів до таблиць у реляційній базі даних. ORM, по суті, працює перетворюючи дані з одного представлення до іншого [16].

Зважаючи на те, що на сервері Node.js, в якості ORM буде використана Sequalize.js. Бібліотека Sequelize – це Node.js ORM система для роботи з такими базами даних, як PostgreSQL, MySQL, SQLite і MSSQL [17].

1.2.5 Вибір засобів реалізації інтерфейсу користувача

Для проектування структури та реалізації клієнтської частини буде використано HTML, CSS, JavaScript. Також, оскільки однією із цілей магістерської кваліфікаційної роботи є – розробити крос-платформну клієнтську частину зі зручним та зрозумілим інтерфейсом, для швидкої взаємодії користувача із системою та її базою даних в режимі реального часу, доцільним буде застосувати Bootstrap, jQuery та шаблонізатор Dust.js.

Dust.js – це шаблонізатор JavaScript, призначений для роботи в асинхронному режимі на сервері і в браузері [18].

Bootstrap (Бутстрап) – найпопулярніший CSS фреймворк від компанії Twitter для адаптивної верстки сайтів. Bootstrap — це набір готових CSS файлів, який можна підключити до вашого сайту, щоб швидко і якісно зробити його «мобільним» – тобто таким, щоб він коректно відображався на телефонах і планшетах.

Основа Bootstrap – модульна сітка з 12 колонок, яка вміє перелаштовуватися в залежності від розміру екрану пристрою відвідувача вашого сайту.

Крім того, Bootstrap включає в себе стилі оформлення безлічі елементів сучасних сайтів: кнопок, форм, спливаючих вікон, навігаційних панелей і т.п. Важлива перевага Bootstrap — в його масовості. Він використовується так давно і на такій великій кількості сайтів, що можна бути впевненим, що більшість проблем сумісності з браузерами та операційними системами в ньому вирішені [19].

jQuery – бібліотека Javascript, що фокусується на взаємодії Javascript і HTML. Для використання доступні всі можливості базового функціонала jquery, серед яких:

- функції ядра;
- робота із селекторами;
- робота з атрибутами;

- обхід дерева DOM;
- маніпуляції елементами;
- робота з CSS-властивостями елементів;
- робота з подіями;
- візуальні ефекти;
- взаємодія з ajax;
- утиліти.

1.3 Висновок

Отже, в процесі огляду технічних та програмних засобів створення інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування було проведено аналіз аналогічних рішень, обґрунтування і вибір методів та інструментів, вибір засобів реалізації серверної частини, вибір засобів організації та реалізації бази даних, вибір ORM (Об’єктно-Реляційна Проекція) та вибір засобів реалізації інтерфейсу користувача.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ АВТОМАТИЗОВАНОЇ СИСТЕМИ РОБОЧИХ МІСЦЬ ДЛЯ СПІВРОБІТНИКІВ СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ

2.1 Опис типів користувачів

Основним завданням інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування є забезпечення автоматизованим робочим місцем кожної ланки (ролі) робочого процесу:

- Адміністратор (Директор);
- Менеджер (Модератор);
- Інженер;
- Зав. складом;
- Клієнт;
- Бухгалтер.

2.2 Постановка та моделювання задачі про призначення

Здійснимо постановку задачі про призначення, тобто опишемо початкові умови.

В системі є інженери: A, B, C, D. Кожен інженер має набір характеристик { t, c, p }, де:

“t” – середній час його роботи (чим більше, тим гірше (для СТО))

“c” – його кваліфікація [1,2,3,4]. Чим більша - тим краща (для СТО)

“p” – його зарплата за умовну одиницю часу роботи. Чим більша - тим гірше для СТО.

$$A = \{t = 1, c = 4, p = 300\},$$

$$B = \{t = 2, c = 3, p = 200\},$$
$$C = \{t = 4, c = 2, p = 100\},$$
$$D = \{t = 8, c = 1, p = 80\}.$$

Також в системі є задачі: X, Y, Z, W. Кожна задача має характеристику “s” – складність [1,2,3,4]. Чим більша цифра – тим складніша задача (тим гірше для СТО).

$$X = \{s = 3\},$$
$$Y = \{s = 4\},$$
$$Z = \{s = 1\},$$
$$W = \{s = 2\}.$$

2.2.1 Моделювання задачі для Угорського алгоритму

Змоделюємо задачу про призначення завдань інженерам для вирішення її Угорським алгоритмом.

Першим кроком треба скласти матрицю відповідності де будуть пересікатись кожна задача з кожним інженером. Пуста матриця для заповнення відображена в таблиці 2.1.

Таблиця 2.1 – Матриця для заповнення.

	Інженер A	Інженер B	Інженер C	Інженер D
Задача X				
Задача Y				
Задача Z				
Задача W				

Коли дана матриця буде заповнена значеннями, можна буде вирішити задачу Угорським алгоритмом. За алгоритмом треба буде шукати найменші значення. Тому для кожної комірки, тобто для кожного варіанту вирішення кожної задачі кожним інженером треба знайти певний коефіцієнт (і чим меншим він буде – тим краще).

Формуємо його наступним чином: якщо якась характеристика краща для СТО коли вона зростає - кладу в знаменник; якщо якась характеристика гірша для СТО коли вона зростає - кладу в чисельник.

Даний коефіцієнт чим менший – тим кращий.

$$K = \frac{s*t*p}{c*100}. \quad (2.1)$$

Покладемо складність задачі, зарплату інженера та час виконання в чисельник бо чим вони вищі - тим гірше для СТО. А кваліфікацію інженера в знаменник, бо чим вона більша - тим краще для СТО.

Також можна додати степені важливості до чисельника і знаменника – a та b відповідно:

$$K = \frac{(s*t*p)^a}{(c*100)^b}. \quad (2.2)$$

І таким чином регулювати потреби підприємства в різні сезони роботи.
Тепер можна для кожної комбінації розрахувати цей коефіцієнт:

$$K_{AX} = \frac{3 * 1 * 300}{4 * 100} = 2.25,$$

$$K_{AY} = \frac{4 * 1 * 300}{4 * 100} = 3,$$

$$K_{Az} = \frac{1 * 1 * 300}{4 * 100} = 0.75,$$

$$K_{AW} = \frac{2 * 1 * 300}{4 * 100} = 1.5,$$

… … …

$$K_{DX} = \frac{3 * 8 * 80}{1 * 100} = 19.2,$$

$$K_{DY} = \frac{4 * 8 * 80}{1 * 100} = 1.5,$$

$$K_{DZ} = \frac{1 * 8 * 80}{1 * 100} = 6.4,$$

$$K_{DW} = \frac{2 * 8 * 80}{1 * 100} = 12.8.$$

Наступним кроком можна заповнити матрицю. Заповнена матриця відображена в таблиці 2.2.

Таблиця 2.2. – Матриця, заповнена значеннями коефіцієнтів.

	Інженер A	Інженер B	Інженер C	Інженер D
Задача X	2,25	2	6	19,2
Задача Y	3	5,3	8	25,6
Задача Z	0,75	1,3	2	6,4
Задача W	1,5	2,7	4	12,8

Матриця готова для вирішення Угорським алгоритмом (де найменше значення – там краще).

2.2.2 Моделювання задачі для Генетичного алгоритму

Тепер інтерпретуємо модель задачі для вирішення її Генетичним алгоритмом.

1) Визначення критеріїв цільової функції.

Першим х критерієм буде характеристика задачі, а саме – її складність s. А другим у критерієм буде характеристика інженера. Характеристику інженера визначимо наступним чином:

$$y = \frac{c*100}{t*p}. \quad (2.3)$$

Чим вища характеристика інженера, тим краще для СТО. Тому в чисельник я поклав кваліфікацію інженера, яка повинна прямувати до максимуму. А в знаменник – час виконання та його оплата – що повинні прямувати до мінімуму.

2) Визначення цільової функції.

Загалом, цільова функція буде прямувати до мінімуму, отже в чисельник необхідно покласти складність задачі – х. А в знаменник – характеристика інженера у яка має прямувати до максимуму. У висновку виходить що:

$$F = \sum \left(\frac{X_I}{Y_I}\right). \quad (2.4)$$

Таким чином, наша цільова функція готова. Наступним етапом необхідно обчислити характеристику х кожного інженера спираючись на початкові дані:

$$Y_A = \frac{4 * 100}{1 * 300} = 1.33,$$
$$Y_B = \frac{3 * 100}{2 * 200} = 0.75,$$
$$Y_C = \frac{2 * 100}{4 * 100} = 0.5,$$
$$Y_D = \frac{1 * 100}{8 * 80} = 0.16.$$

Тепер, можемо заповнити матрицю для Генетичного алгоритму. Для цього в будь якому порядку записуємо задачі в таблицю, в 4 рядка, тобто формуємо 4 хромосоми. Кожен стовпчик – це інженер, який може виконати лише 1 задачу. Матриця для Генетичного алгоритму відображена в Таблиці 2.3

Таблиця 2.3 – Матриця для Генетичного алгоритму

Інженер A	Інженер B	Інженер C	Інженер D
X	Y	W	Z
Z	X	Y	W
W	Z	X	Y
Y	Z	W	X

Наступним кроком починаємо розв’язувати задачу Генетичним алгоритмом. Робимо по одній мутації двох елементів в двох будь яких хромосомах.

Наприклад: в першій хромосомі міняємо 2 і 3 елементи місцями, в третій – 1 і 3 елементи. В результаті отримуємо 2 нові хромосоми з новим порядком (набором) елементів. Напишемо усі існуючі і отримані хромосоми в матрицю, що відображена в Таблиці 2.4.

Таблиця 2.4 – Матриця з існуючими і отриманими хромосомами.

A	**B**	**C**	**D**
X	Y	W	Z
Z	X	Y	W
W	Z	X	Y
Y	Z	W	X
X	W	Y	Z
X	Z	W	Y

Наступним етапом буде обрахунок цільової функції для кожної хромосоми.

$$F_1 = \frac{x_X}{y_A} + \frac{x_Y}{y_B} + \frac{x_W}{y_C} + \frac{x_Z}{y_D} = \frac{3}{1.33} + \frac{4}{0.75} + \frac{2}{0.5} + \frac{1}{0.16} = 17.84,$$

$$F_2 = \frac{x_Z}{y_A} + \frac{x_X}{y_B} + \frac{x_Y}{y_C} + \frac{x_W}{y_D} = \frac{1}{1.33} + \frac{3}{0.75} + \frac{4}{0.5} + \frac{2}{0.16} = 25.26,$$

$$F_3 = \frac{x_W}{y_A} + \frac{x_Z}{y_B} + \frac{x_X}{y_C} + \frac{x_Y}{y_D} = \frac{2}{1.33} + \frac{1}{0.75} + \frac{3}{0.5} + \frac{4}{0.16} = 33.83,$$

$$F_4 = \frac{x_Y}{y_A} + \frac{x_Z}{y_B} + \frac{x_W}{y_C} + \frac{x_X}{y_D} = \frac{4}{1.33} + \frac{1}{0.75} + \frac{2}{0.5} + \frac{3}{0.16} = 27.08,$$

$$F_5 = \frac{x_X}{y_A} + \frac{x_W}{y_B} + \frac{x_Y}{y_C} + \frac{x_Z}{y_D} = \frac{3}{1.33} + \frac{2}{0.75} + \frac{4}{0.5} + \frac{1}{0.16} = 37.17,$$

$$F_6 = \frac{x_X}{y_A} + \frac{x_Z}{y_B} + \frac{x_W}{y_C} + \frac{x_Y}{y_D} = \frac{3}{1.33} + \frac{1}{0.75} + \frac{2}{0.5} + \frac{4}{0.16} = 31.83.$$

Тепер, маючи набір значень цільових функцій, виберемо з нього 4 найкращі варіанти використовуючи фітнес функцію. Для нашого випадку, цільова функція прямує до мінімуму, тому необхідно буде обрати найменші результати. Отже: F_1, F_2, F_4 та F_6.

Наступним етапом необхідно взяти три хромосоми що відповідають цим цільовим функціям та повторити процедуру мутації та застосування фітнес функції знову. Такий цикл можна проробити 5 - 10 разів. Очевидно, що чим більше разів - тим точніший буде результат.

2.3 Формування процесів та методів роботи модуля «Менеджер»

Менеджмент робочого процесу на виробництві – одна з найважливіших функцій, яку повинна виконувати відповідальна особа – менеджер. Даній ролі буде надано можливість та засоби управління та контролю за усім життєвим циклом замовлення, починаючи від процесу його формування до видачі квитанції клієнту після його успішного виконання.

2.3.1 Формування методів роботи модуля «Менеджер»

Менеджер має доступ майже до усіх даних та процесів системи. Функціонал, що повинна забезпечувати інформаційна автоматизована система робочих місць для співробітників станції технічного обслуговування для ролі менеджер:

- Створити, переглянути та редагувати замовлення клієнта;
- Створити, переглянути та редагувати задачі інженера;
- Створити, переглянути та редагувати типи задач системи;
- Створити, переглянути та редагувати користувачів системи (менеджеру доступне створення та редагування лише користувачів типу клієнт);
- Створити, переглянути та редагувати деталі необхідні для виконання замовлення;
- Почати, завершити, анулювати, доопрацювати замовлення;
- Оновити, передоручити, анулювати задачі;

- Переглядати статистику замовлень, задач та фінансів системи;
- Надрукувати квитанцію із зазначенням таблиці виконаних задач та використаних деталей та відповідних їм сум грошей;
- Змінювати стан замовлення на будь-який з визначених системою станів, шляхом натискання відповідних кнопок:
  1) Кнопки «Почати» та «Доопрацювати» змінюють статус замовлення на «Замовлення виконується»;
  2) Кнопка «Завершити» змінює статус замовлення на «Замовлення виконано»;
  3) Кнопка «Анулювати» змінює статус замовлення на «Замовлення анульовано»;
  4) Кнопка «Видати» змінює статус замовлення на «Замовлення видано».
- Змінювати стан задачі на певні з визначених системою станів, шляхом натискання відповідних кнопок:
  1) Кнопки «Оновити задачу» змінює статус задачі на «Задача в очікуванні»;
  2) Кнопка «Завершити задачу» змінює статус задачі на «Задачу виконано»;
  3) Кнопка «Анулювати задачу» змінює статус задачі на «Задачу анульовано»;
- Переглядати списки замовлень у відфільтрованому вигляді по категоріях: «Усі», «Не початі», «Виконуються», «Очікують запчастини», «Анульовані», «Видані», «Виконані», та «Видаленні».

Отже, увесь функціонал, що повинна забезпечувати інформаційна автоматизована система робочих місць для співробітників станції технічного обслуговування для ролі менеджер описаний в use-case діаграмі. UML діаграма прецедентів модуля «Менеджер» на рисунку 2.1.

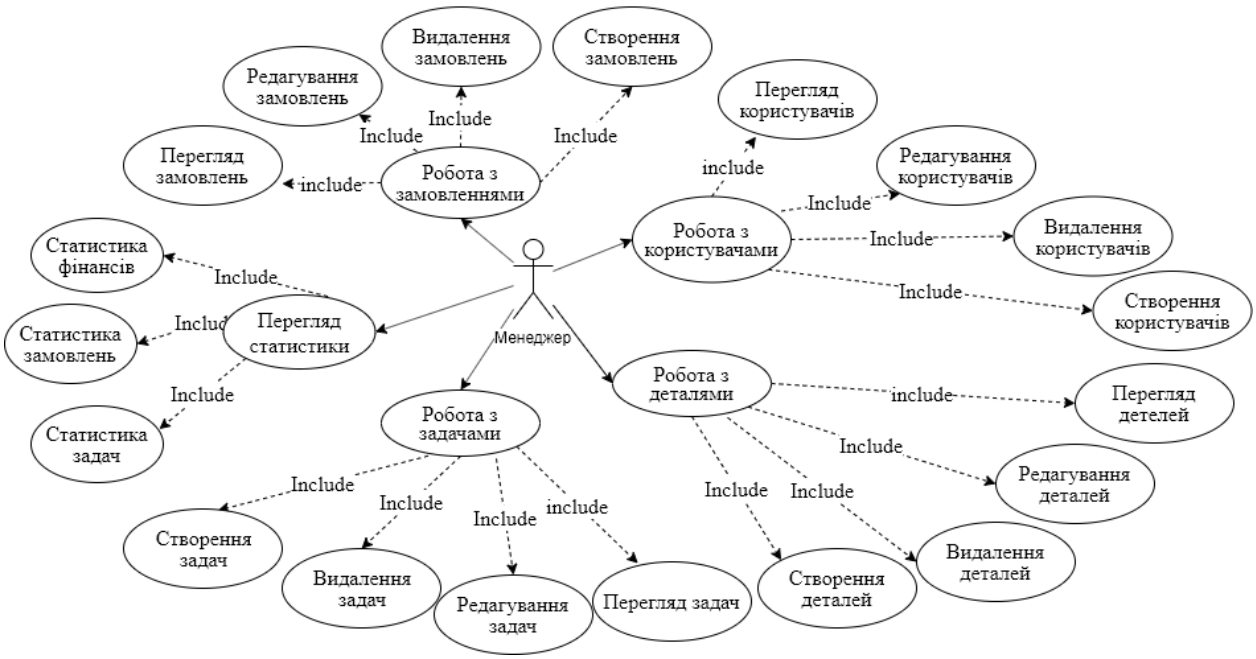


Рисунок 2.1 – UML діаграма прецедентів модуля «Менеджер»

2.3.2 Формування процесу створення та редагування користувача в модулі «Менеджер»

Задля забезпечення зручності та простоти створення та редагування користувача варто реалізувати цей процес у вигляді заповнення (зміни) усіх необхідних полів форми в модальному вікні із зазначенням обов’язковості на необов’язковості деяких з них. Варто зауважити що в обов’язки менеджера входить створення тільки користувачів з роллю «Клієнт». Доступ до створення усіх інших типів користувачів наявний у адміністратора. Для зберігання усієї необхідної інформації про користувача доцільно буде створити наступні поля для заповнення менеджером: «Ім’я», «Прізвище», «Компанія», «Адреса», «Контактний номер», «Логін», «Email» та «Пароль».
В логіці роботи форми з даними користувача необхідно також реалізувати перевірку на правильність введених даних та заповнення усіх обов’язкових полів та вивід відповідних повідомлень про помилки в діях менеджера.

Дані про усіх користувачів в рамках інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування зручно буде відображати у виді таблиці. Варто зазначити, що функція видалення користувача не буде доступна для менеджера. Дана частина менеджменту користувачами буде доступна тільки адміністратору.

Опишемо усі доступні процеси управління користувачами для модуля «Менеджер за допомогою UML діаграми діяльності. Діаграма діяльності модуля «Менеджер»: управління користувачами відображена на рисунку 2.2.

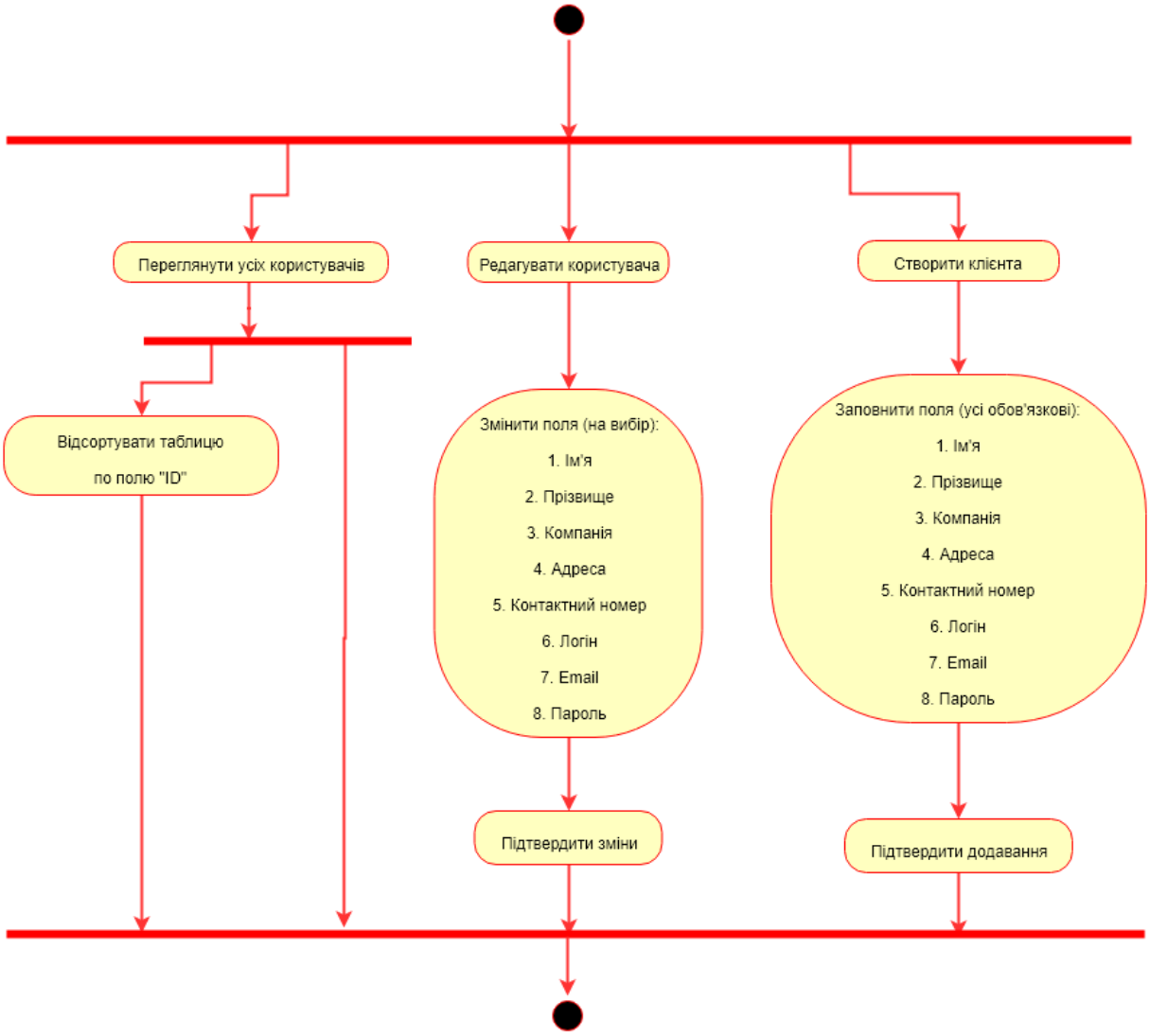


Рисунок 2.2 – Діаграма діяльності модуля «Менеджер»: управління користувачами

2.3.3 Формування процесів управління деталями в модулі «Менеджер»

Задля забезпечення зручності та простоти створення та редагування інформації про деталь варто реалізувати цей процес у вигляді заповнення (зміни) усіх необхідних полів форми в модальному вікні із зазначенням обов’язковості на необов’язковості деяких з них. Для зберігання усієї необхідної інформації про деталь доцільно буде створити наступні поля для заповнення менеджером: «Назва деталі», «Тип транспорту», «Марка автомобіля», «Модель автомобіля» та «Вартість».

В логіці роботи форми з даними деталі необхідно також реалізувати перевірку на правильність введених даних та заповнення усіх обов’язкових полів та вивід відповідних повідомлень про помилки в діях менеджера.

Дані про усі деталі в рамках інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування зручно буде відображати у виді таблиці. Варто зазначити, що функція видалення деталі не буде доступна для менеджера. Дана частина менеджменту деталями буде доступна тільки адміністратору.

Опишемо усі доступні процеси управління деталями для модуля «Менеджер за допомогою UML діаграми діяльності. Діаграма діяльності модуля «Менеджер»: управління деталями відображена на рисунку 2.3.

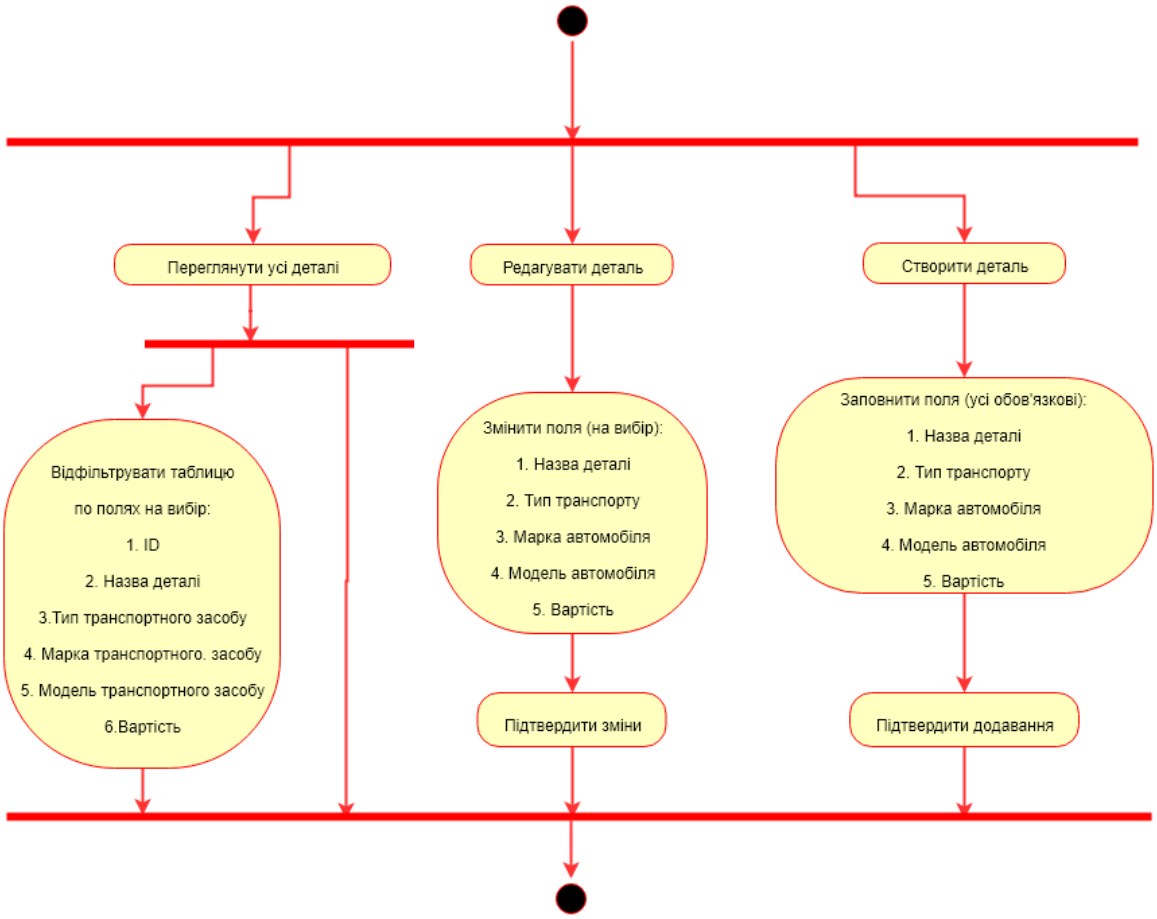


Рисунок 2.3 – Діаграма діяльності модуля «Менеджер»: управління деталями

2.3.4 Формування процесів управління задачами в модулі «Менеджер»

Задля забезпечення зручності та простоти створення та редагування інформації про задачу варто реалізувати цей процес у вигляді заповнення (зміни) усіх необхідних полів форми в модальному вікні із зазначенням обов’язковості на необов’язковості деяких з них. Для зберігання усієї необхідної інформації про задачу доцільно буде створити наступні поля для заповнення менеджером: «Назва задачі», «Виконавець», «Доручити задачу», «Час початку», «Час виконання» та «Вартість».

В логіці роботи форми з даними задачі необхідно також реалізувати перевірку на правильність введених даних та заповнення усіх обов’язкових полів та вивід відповідних повідомлень про помилки в діях менеджера.

Дані про усі задачі в рамках інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування зручно буде відображати у виді таблиці. Варто зазначити, що функція видалення задачі не буде доступна для менеджера. Дана частина менеджменту задачами буде доступна тільки адміністратору.

Менеджер може змінити статус задачі на «Анульовано» якщо дана задача більш не є актуальною, або завершити задачу якщо вважає що робота над нею закінчена в рамках визначеного часу. Також, користувач з даною роллю може оновити задачу якщо вважає, що задачу ще потрібно доопрацювати. Зміни статусів відбуваються шляхом натиснення відповідних кнопок: «Завершити задачу», «Оновити задачу», «Анулювати задачу».

Опишемо усі доступні процеси управління задачами для модуля «Менеджер за допомогою UML діаграми діяльності. Діаграма діяльності модуля «Менеджер»: управління задачами відображена на рисунку 2.4.

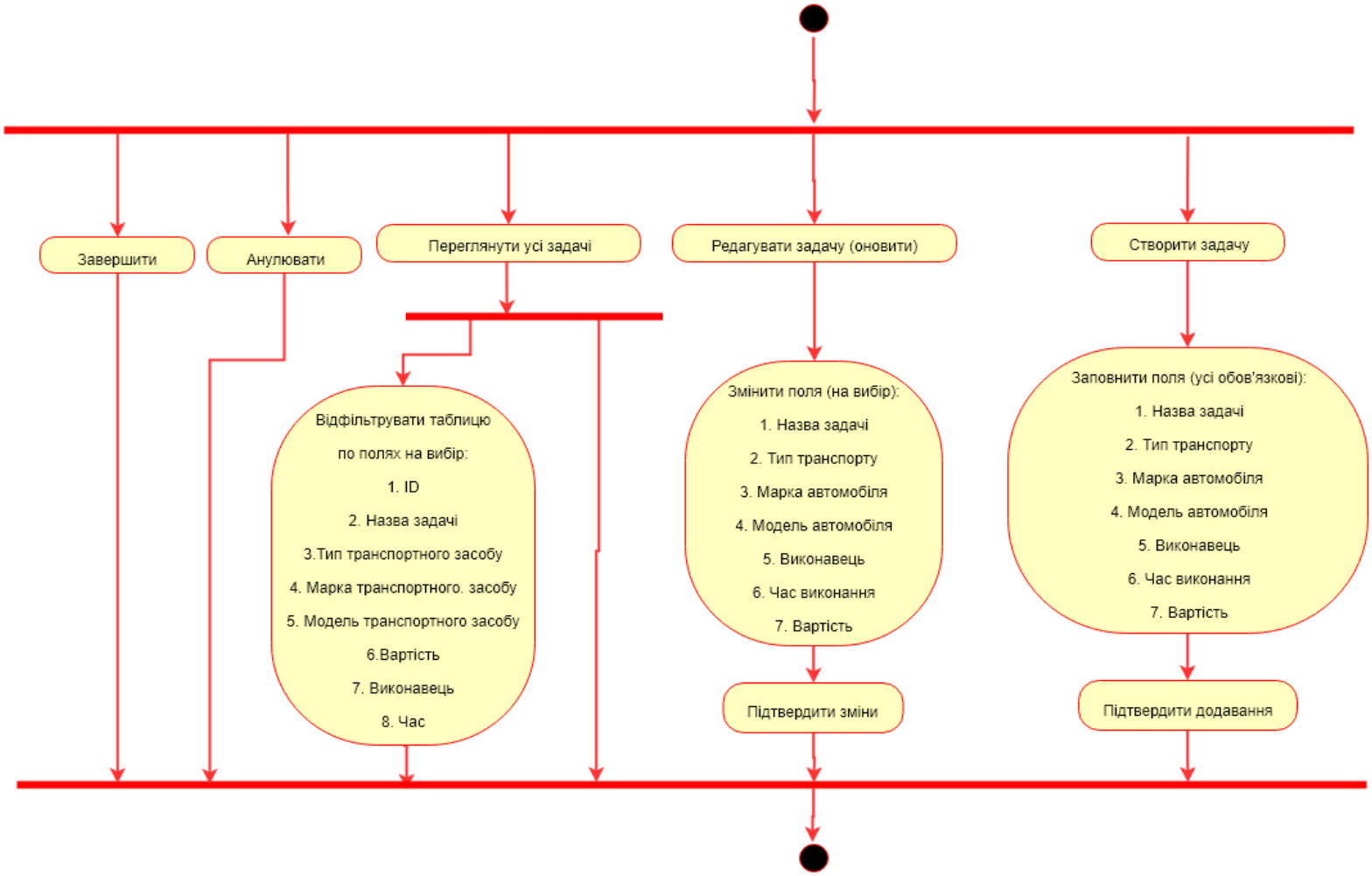


Рисунок 2.4 – Діаграма діяльності модуля «Менеджер»: управління задачами

2.3.5 Формування процесів управління замовленнями в модулі «Менеджер»

Задля забезпечення зручності та простоти створення та редагування інформації про замовлення варто реалізувати цей процес у вигляді заповнення (зміни) усіх необхідних полів форми в на окремій сторінці створення (редагування) замовлення із зазначенням обов’язковості на необов’язковості деяких з них. Для зберігання усієї необхідної інформації про замовлення доцільно буде створити наступні поля для заповнення менеджером: «Тип транспорту», «Марка транспорту», «Модель транспорту», «Клієнт», «Назва замовлення», «Час початку», «Орієнтовний час виконання», «Опис замовлення» та «Коментар». Останні два поля не будуть обов’язковими для заповнення, оскільки навіть без них вистачає інформації для формування та подальшого виконання замовлення.

В логіці роботи форми з даними замовелння необхідно також реалізувати перевірку на правильність введених даних та заповнення усіх обов’язкових полів та вивід відповідних повідомлень про помилки в діях менеджера.

Дані про усі задачі в рамках інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування зручно буде відображати у виді таблиці. Варто зазначити, що функція видалення замовлення не буде доступна для менеджера. Дана частина менеджменту задачами буде доступна тільки адміністратору.

Задля покращення взаємодії з клієнтом та прозорості процесу формування суми оплати замовлення менеджером, буде реалізована функція автоматичного формування та друку квитанції із зазначенням усієї необхідної для користувача інформації. В тому числі: списку виконаних задач, деталей та сум, що повинен заплатити клієнт за надані послуги.

Оскільки менеджер та адміністратор в рамках інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування будуть виконувати функцію контролю та управління за всім життєвим циклом замовлення, у їх обов’язки буде входити зміна статусів замовлення протягом усього процесу його виконання: «Замовлення в очікуванні», «Замовлення виконується», «Замовлення виконано», «Замовлення анульовано», «Замовлення видано».

Змінювати стан замовлення на будь-який з визначених системою станів, буде зручно шляхом натискання відповідних кнопок, розміщених в блоці відповідного замовлення: «Почати», «Доопрацювати», «Завершити», «Анулювати», «Видати».

Опишемо усі доступні процеси управління замовленнями для модуля «Менеджер за допомогою UML діаграми діяльності.

Діаграма діяльності модуля «Менеджер»: управління замовленнями відображена на рисунку 2.5.

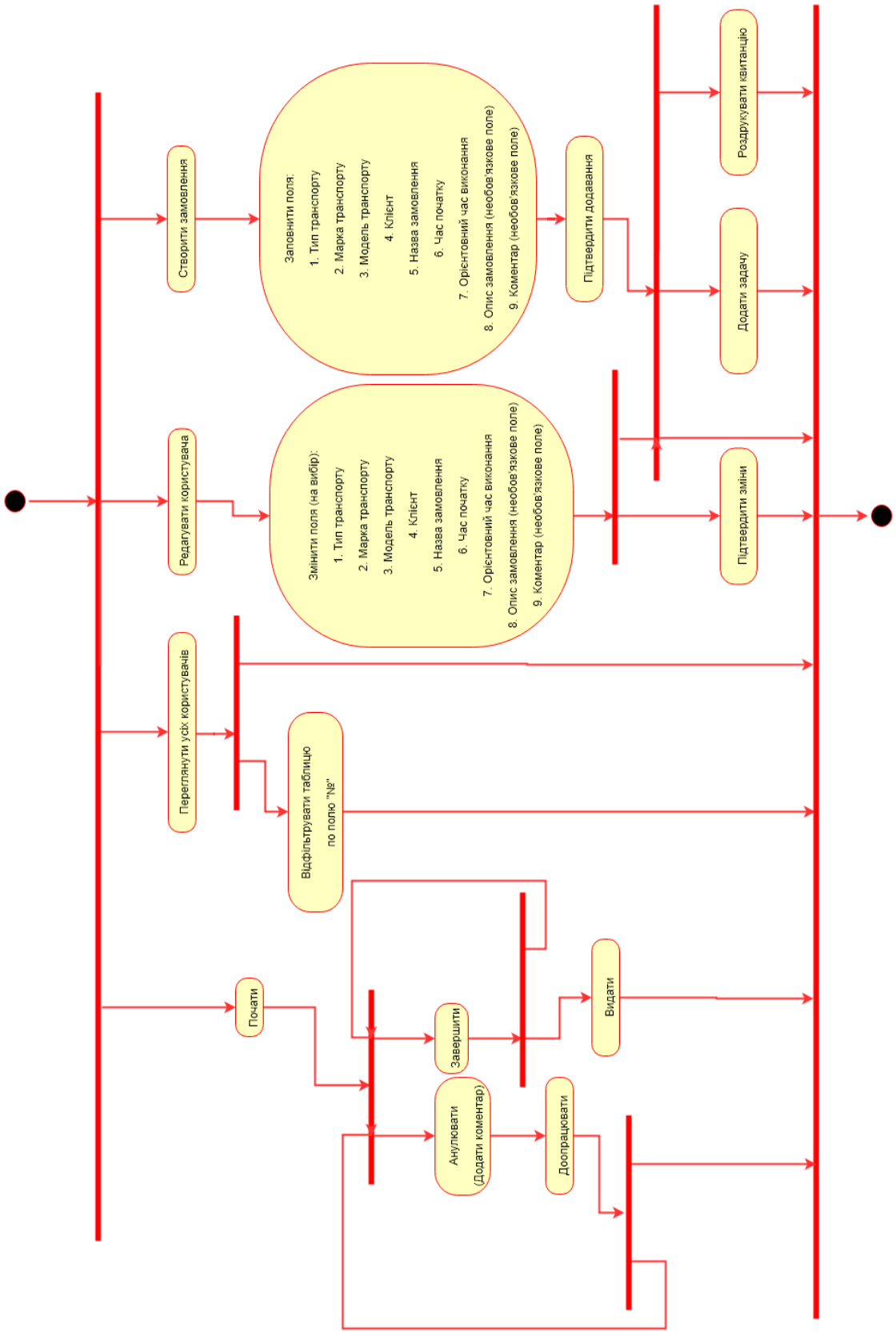


Рисунок 2.5 – Діаграма діяльності модуля «Менеджер»: управління замовленням

2.4 Формування процесів та методів роботи модуля «Інженер»

Безпосереднє виконання замовлень на станції технічного обслуговування – одна з найважливіших функцій, яку повинна виконувати відповідальна особа – інженер. Даній ролі буде надано можливість та засоби управління над своїми задачами, які в сукупності являють собою замовлення.

2.4.1 Формування методів роботи модуля «Інженер»

Користувачам з типом інженер доступні для перегляду та опрацювання лише певні задачі, що є частиною замовлення, виконавцем яких обраний він. Інженер має можливість змінити статус задачі на «Виконується», «Виконано» або «Зупинено». При виборі останнього стану, інженеру необхідно буде вказати причину зупинки виконання задачі (не має необхідних деталей і т.д.).

Функціонал, що повинна забезпечувати інформаційна автоматизована система робочих місць для співробітників станції технічного обслуговування для ролі інженер:

1) Редагувати задачі;
2) Змінити статус задачі на:

«Задача виконується» – кнопка «Почати задачу»;

«Задачу зупинено» – кнопка «Відсутні запчастини»;

«Задачу виконано» – кнопка «Завершити задачу».

3) Переглядати власні задачі з можливістю сортування по полю «ID».

Отже, увесь функціонал, що повинна забезпечувати інформаційна автоматизована система робочих місць для співробітників станції технічного обслуговування для ролі інженер описаний в use-case діаграмі. UML діаграма прецедентів модуля «Інженер» на рисунку 2.6.

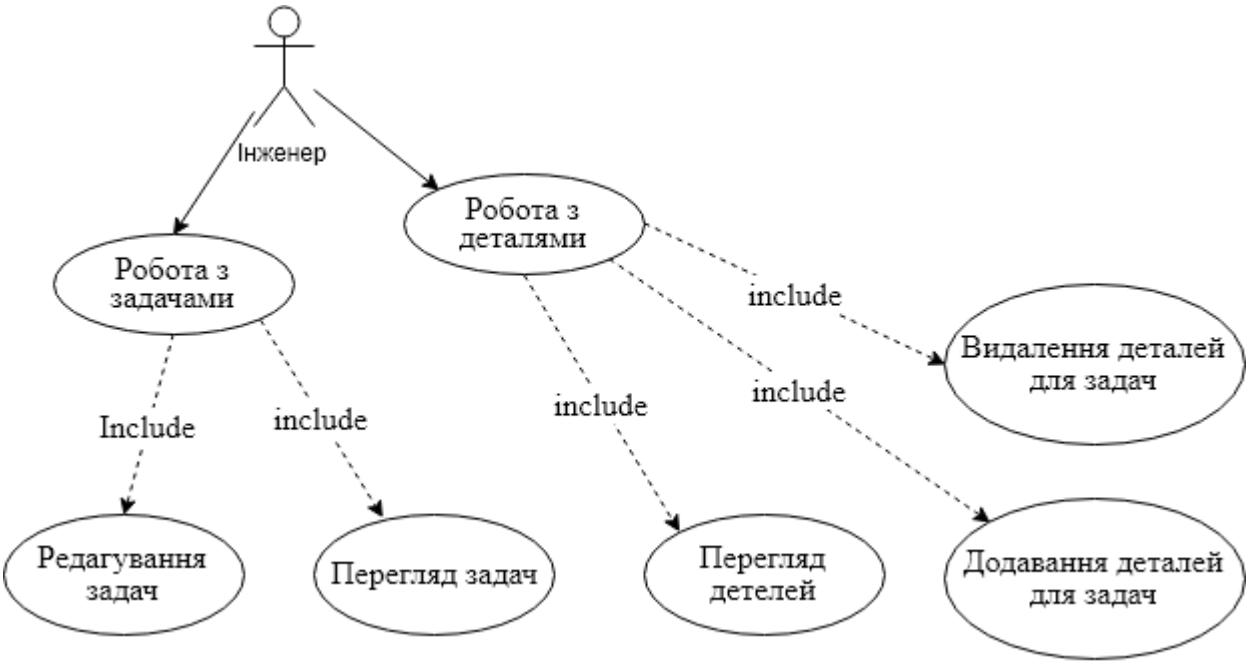


Рисунок 2.6 – UML діаграма прецедентів модуля «Інженер»

2.4.2 Формування процесу управління задачами в модулі «Інженер»

Задля забезпечення зручності та простоти редагування даних задачі варто реалізувати цей процес у вигляді зміни необхідних полів форми в модальному вікні із зазначенням обов’язковості на необов’язковості деяких з них. Варто зауважити що в обов’язки менеджера не входить зміна полів «Назва задачі», «Виконавець». Доступ до зміни цих полів наявний у менеджера та адміністратора. Для оновлення усієї необхідної інформації про задачу доцільно буде створити наступні поля для редагування інженером: «Доручити задачу», «Опис задачі», «Час початку», «Час виконання», «Вартість», «Деталі» та «Коментар». Варто зазначити що поле «Коментар» не є обов’язковим для заповнення чи редагування інженером, тобто є опціональним.

В логіці роботи форми з даними користувача необхідно також реалізувати перевірку на правильність введених даних та заповнення усіх обов’язкових полів та вивід відповідних повідомлень про помилки в діях інженера.

Дані про усі задачі інженера в інформаційній автоматизованій системі робочих місць для співробітників станції технічного обслуговування зручно буде відображати у виді таблиці.

Варто зазначити, що функція видалення задачі не буде доступна для інженера. Дана частина менеджменту задачами буде доступна тільки адміністратору.

Інженер може змінити статус задачі на «Задача виконується», натиснувши на кнопку «Почати задачу» якщо він пересвідчився в наявності усіх необхідних деталей та готовий приступити до виконання.

Також, користувач з даною роллю може зупинити задачу, натиснувши на кнопку «Відсутні запчастини» якщо вважає, що для виконання задачі не вистачає якихось деталей.

І коли інженер завершив роботу над своєю задачею, він змінює їй статус на «Задача виконана» шляхом натискання на кнопку «Завершити задачу».

Опишемо усі доступні процеси управління власними задачами для модуля «Інженер» за допомогою UML діаграми діяльності. Діаграма діяльності модуля «Інженер»: управління задачами відображена на рисунку 2.7.

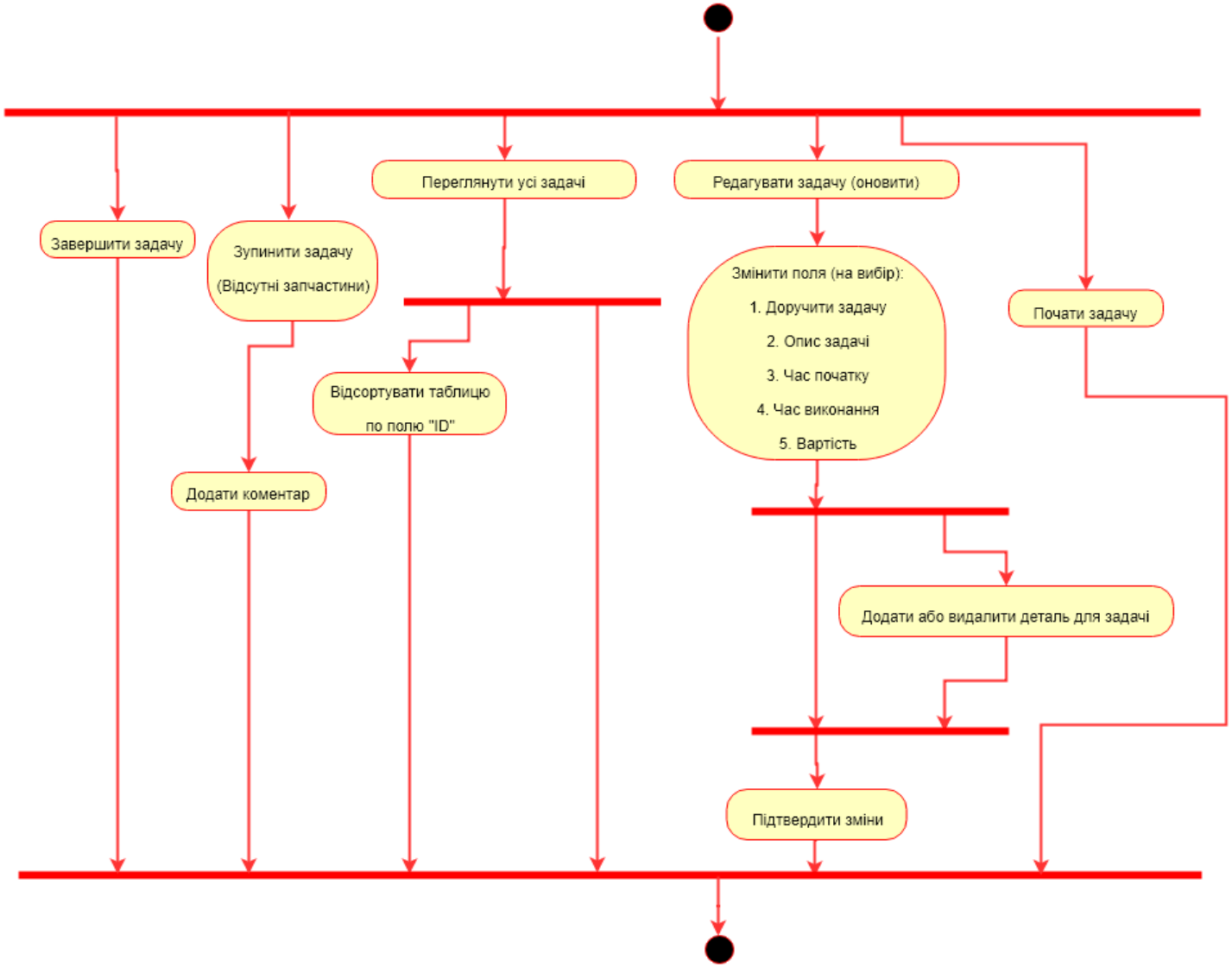


Рисунок 2.7 – Діаграма діяльності модуля «Інженер»: управління задачами

2.5 Формування процесів та методів роботи модуля «Клієнт»

Найголовнішою роллю в інформаційній автоматизованій системі робочих місць для співробітників станції технічного обслуговування є та, за для якої здійснювалась реалізація інших п’яти – це, безпосередньо, клієнт.

Даній ролі буде надано можливість та засоби для перегляду власних замовлень та задач, на які вони поділені.

2.5.1 Формування методів роботи модуля «Клієнт»

Клієнт має можливість лише спостерігати за замовленнями оформленими на нього.

Клієнт бачить усю інформацію про хід робіт: хто виконує задачі, в якому стані вони знаходяться, які деталі використовуються і їх вартість, скільки часу заплановано на роботу та які проблеми виникають у процесі виконання.

Функціонал, що повинна забезпечувати інформаційна автоматизована система робочих місць для співробітників станції технічного обслуговування для ролі клієнт – переглядати власні замовлення та задачі з можливістю сортування по полю «ID».

Отже, увесь функціонал для ролі інженер описаний в use-case діаграмі. UML діаграма прецедентів модуля «Клієнт» на рисунку 2.8.

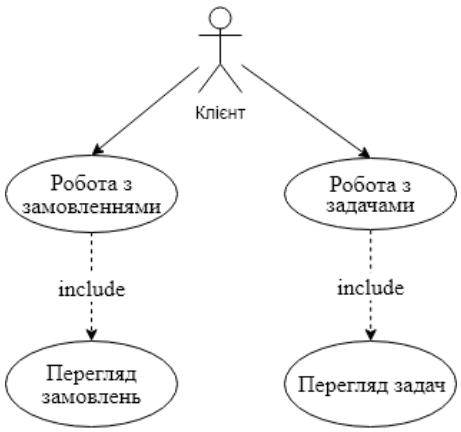


Рисунок 2.8 – UML діаграма прецедентів модуля «Клієнт»

2.5.2 Формування процесу перегляду замовлень та задач в модулі «Клієнт»

Забезпечення зручного, простого та комфортного перегляду інформації про замовлення, його статус виконання в даний момент часу є основним завданням, що підлягає реалізації під час розробки інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування.

Найоптимальнішим варіантом відображення списку доступних для даного типу користувача даних є таблиця з можливістю сортування та візуального розрізнення статусу замовлення за допомогою відповідності певного кольору до певного статусу.

Також, зручним рішенням для розрізнення замовлень буде реалізація двох категорій в таблиці: «Активні» – де будуть відображатись поточні замовлення зі статусами «Замовлення в очікуванні», «Замовлення анульовано», «Замовлення зупинено» та «Замовлення виконується», та «Історія» – тільки «Замовлення видано», тобто повністю закінчені замовлення.

Крім того, необхідно реалізувати сповіщення користувача про зміни статусів замовлення через відсилання листів на електронну пошту.

Опишемо усі доступні процеси управління власними задачами для модуля «Інженер» за допомогою UML діаграми діяльності. Діаграма діяльності модуля «Інженер» відображена на рисунку 2.9.

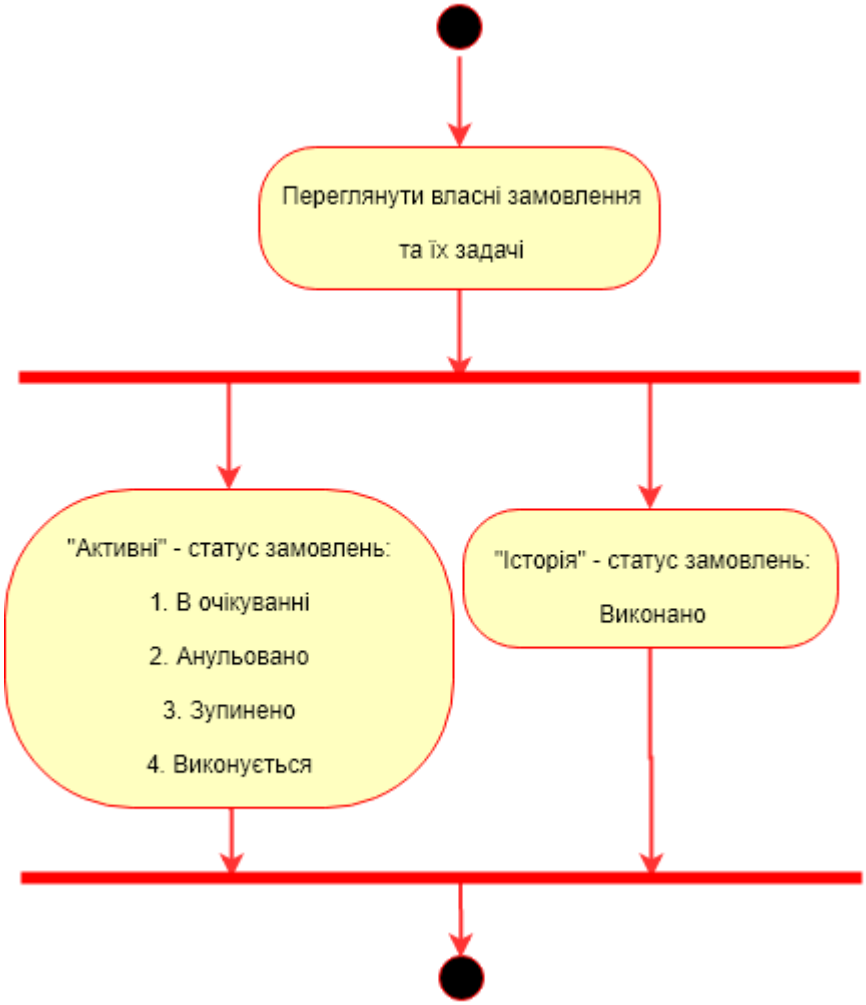


Рисунок 2.9 – Діаграма діяльності модуля «Клієнт»

2.6 Формування процесів та методів управління в модулі «Адміністратор»

Для успішної роботи будь-якої установи, крім виробничих потужностей, необхідно мати й високі управлінські показники. Завжди є щонайменше одна людина, яка розглядає підприємство як одну цілу одиницю. Головною задачею цієї особи є грамотна організація діяльності установи та підвищення ефективності роботи колективу.

При розробці інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування роль керуючого буде виконувати адміністратор.

Адміністратором мусить бути кваліфікований фахівець. В його обов’язки входить керування людьми і виробничими процесами установи, а також розпорядження власністю компанії в її ж інтересах.

2.6.1 Формування методів взаємодії модуля «Адміністратор»

В рамках розробки інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування адміністратор повинен мати доступ до усіх даних та процесів, що відбуваються в системі. Даний тип користувачів повинен мати повноваження на перегляд, створення, редагування та видалення будь-яких сутностей у системі.

Адміністратор повинен мати можливість:

- Переглядати, додавати, редагувати та видаляти дані користувачів системи;
- Додавати користувачів усіх типів, визначених системою;
- Переглядати, додавати, редагувати та видаляти дані про деталі в системі;
- Переглядати, додавати, редагувати та видаляти дані про типи транспортних засобів у системі;
- Переглядати, додавати, редагувати та видаляти дані про моделі транспортних засобів у системі;
- Переглядати, додавати, редагувати та видаляти дані про марки транспортних засобів у системі;
- Переглядати, додавати, редагувати та видаляти замовлення у системі;
- Переглядати, додавати, редагувати та видаляти задачі в системі;
- Переглядати статистику замовлень, задач та фінансів системи;
- Відправляти на друк документ, який містить у собі список виконаних робіт, використаних деталей з відповідними цінами за отримані послуги та придбані деталі;

- Підтверджувати факт оплати замовлення шляхом натискання кнопки «Розрахувати»;
- Анулювати факт оплати (наприклад при хибних діях бухгалтера);
- Змінювати стан замовлення на будь-який з визначених системою станів, шляхом натискання відповідних кнопок:
    1) Кнопки «Почати» та «Доопрацювати» змінюють статус замовлення на «Замовлення виконується»;
    2) Кнопка «Завершити» змінює статус замовлення на «Замовлення виконано»;
    3) Кнопка «Анулювати» змінює статус замовлення на «Замовлення анульовано»;
    4) Кнопка «Видати» змінює статус замовлення на «Замовлення видано».
    5) Змінювати стан задачі на певні з визначених системою станів, шляхом натискання відповідних кнопок:
    6) Кнопки «Оновити задачу» змінює статус задачі на «Задача в очікуванні»;
    7) Кнопка «Завершити задачу» змінює статус задачі на «Задачу виконано»;
    8) Кнопка «Анулювати задачу» змінює статус задачі на «Задачу анульовано»;
    9) Кнопка «Видати» змінює статус замовлення на «Замовлення видано».

Фільтрувати замовлення по статусах на категорії: «Усі», «Не початі», «Виконуються», «Очікують запчастини», «Анульовані», «Видані», «Виконані», «Розраховані» та «Видаленні».

Підсумуємо описані вище методи доступні адміністратору за допомогою UML діаграми прецедентів на рисунку 2.10.

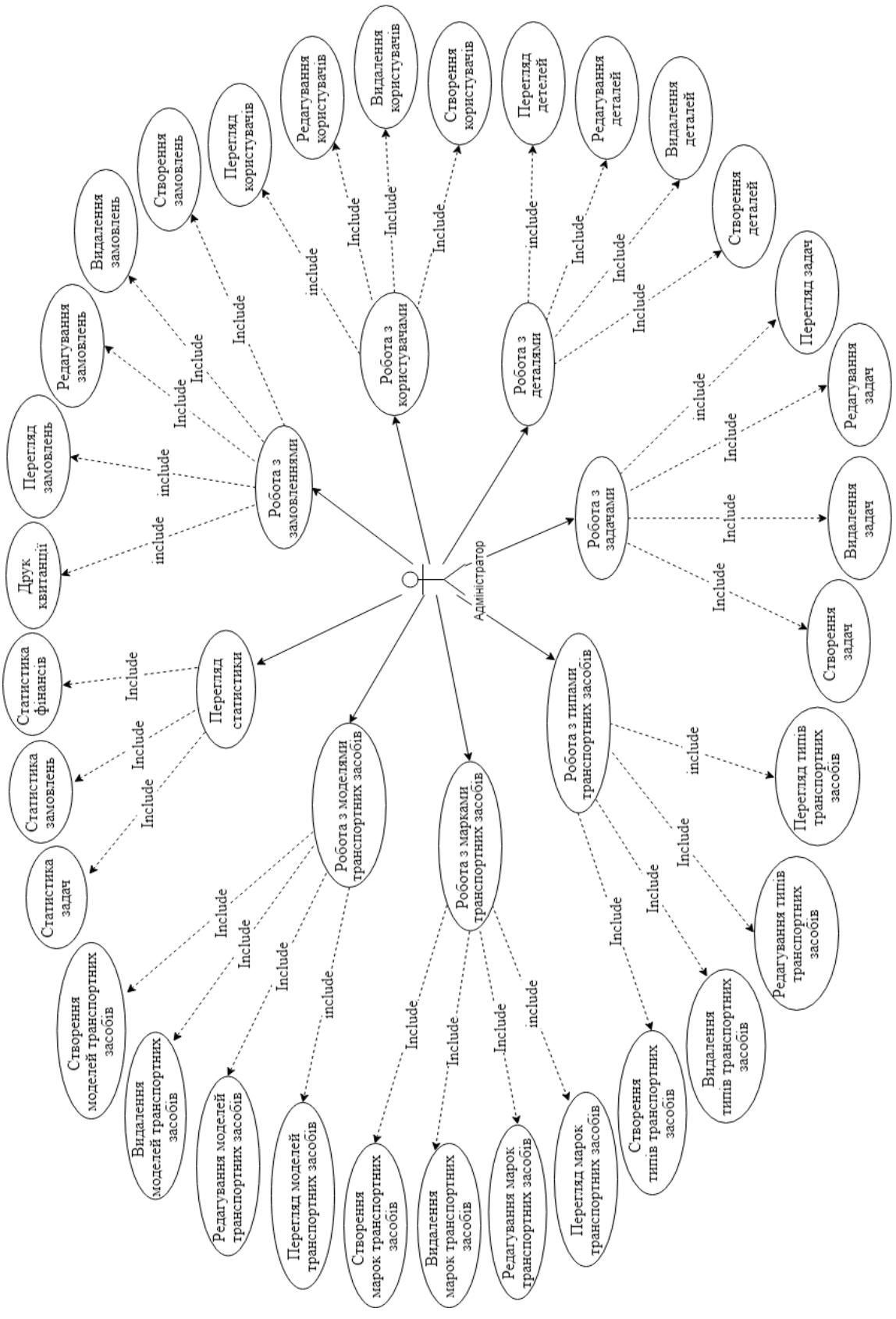


Рисунок 2.10 – Діаграма прецедентів модуля «Адміністратор»

2.6.2 Формування процесів управління користувачами в модулі «Адміністратор»

У модулі «Адміністратор» передбачаються наступні процеси управління користувачами: перегляд, створення, редагування та видалення.

Адміністратору, як головному користувачу, доступна для перегляду вся інформація про всіх користувачів системи. Також лише адміністратор, при створені користувачів, може визначати для них будь-які доступні в системі ролі. Процес створення чи редагування користувача повинен бути простим, швидким та зрозумілим. Доцільно реалізувати інтерфейс у вигляді форми, яка буде містити в собі рядки для заповнення чи редагування вже існуючої інформації. Щоб завершити процес створення чи редагування необхідно вказати наступні поля: «Ім’я», «Прізвище», «Компанія», «Адреса», «Контактний номер», «Логін», «Email», «Роль» та «Пароль». Форму потрібно налаштувати таким чином, щоб адміністратору було зрозуміло, які данні очікує система та куди їх вносити, які поля є обов’язковими і які можна проігнорувати. Також необхідно контролювати процес відправки форми, чи введено мінімум інформації, необхідний для завершення процесу створення чи редагування користувача, чи відповідають формати даних тим, що очікуються системою.

У модулі «Адміністратор» передбачена можливість видалення користувачів. Лише цей тип користувачів може видаляти інших учасників системи.

Також потрібно реалізувати процес обробки та виведення помилок. Користувач повинен одразу зрозуміти неправильність свої дій, прочитавши опис помилки, та досягти бажаного результату з мінімальними втратами часу на вирішення неприємної ситуації.

Підсумуємо описані вище процеси управління користувачами доступні адміністратору за допомогою UML діаграми діяльності на рисунку 2.11.

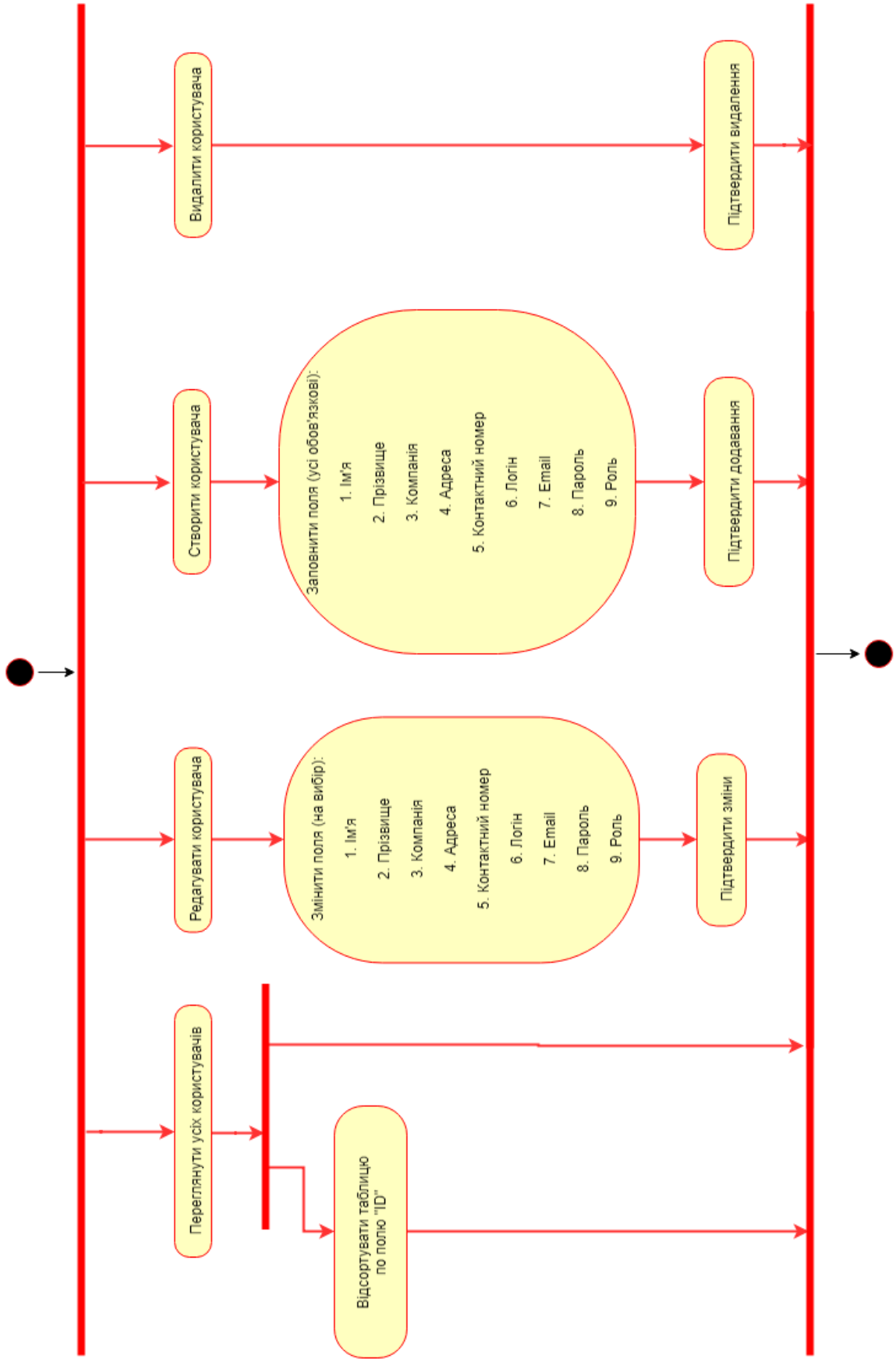


Рисунок 2.11 – Діаграма діяльності модуля «Адміністратор»: управління

2.6.3 Формування процесів управління деталями в модулі «Адміністратор»

У модулі «Адміністратор» передбачаються наступні процеси управління деталями: перегляд, створення, редагування та видалення. Адміністратору доступні для перегляду всі деталі зареєстровані в системі. Процес створення чи редагування деталі повинен бути простим, швидким та зрозумілим. Доцільно реалізувати інтерфейс у вигляді форми, яка буде містити в собі рядки для заповнення чи редагування вже існуючої інформації. Щоб завершити процес створення чи редагування необхідно вказати наступні поля: «Назва деталі», «Тип транспорту», «Марка автомобіля», «Модель автомобіля» та «Вартість». Форму потрібно налаштувати таким чином, щоб адміністратору було зрозуміло, які данні очікує система та куди їх вносити, які поля є обов’язковими і які можна проігнорувати. Також необхідно контролювати процес відправки форми, чи введено мінімум інформації, необхідний для завершення процесу створення чи редагування деталі, чи відповідають формати даних тим, що очікуються системою.

У модулі «Адміністратор» передбачена можливість видалення деталей. Також потрібно реалізувати процес обробки та виведення помилок. Користувач повинен одразу зрозуміти неправильність свої дій, прочитавши опис помилки, та досягти бажаного результату з мінімальними втратами часу на вирішення неприємної ситуації.

Слід зазначити, що лише адміністратор має доступ до даної частини системи і лише він може змінювати списки деталей з якими працює дана станція технічного обслуговування.

Підсумуємо описані вище процеси управління деталями доступні адміністратору за допомогою UML діаграми діяльності на рисунку 2.12.

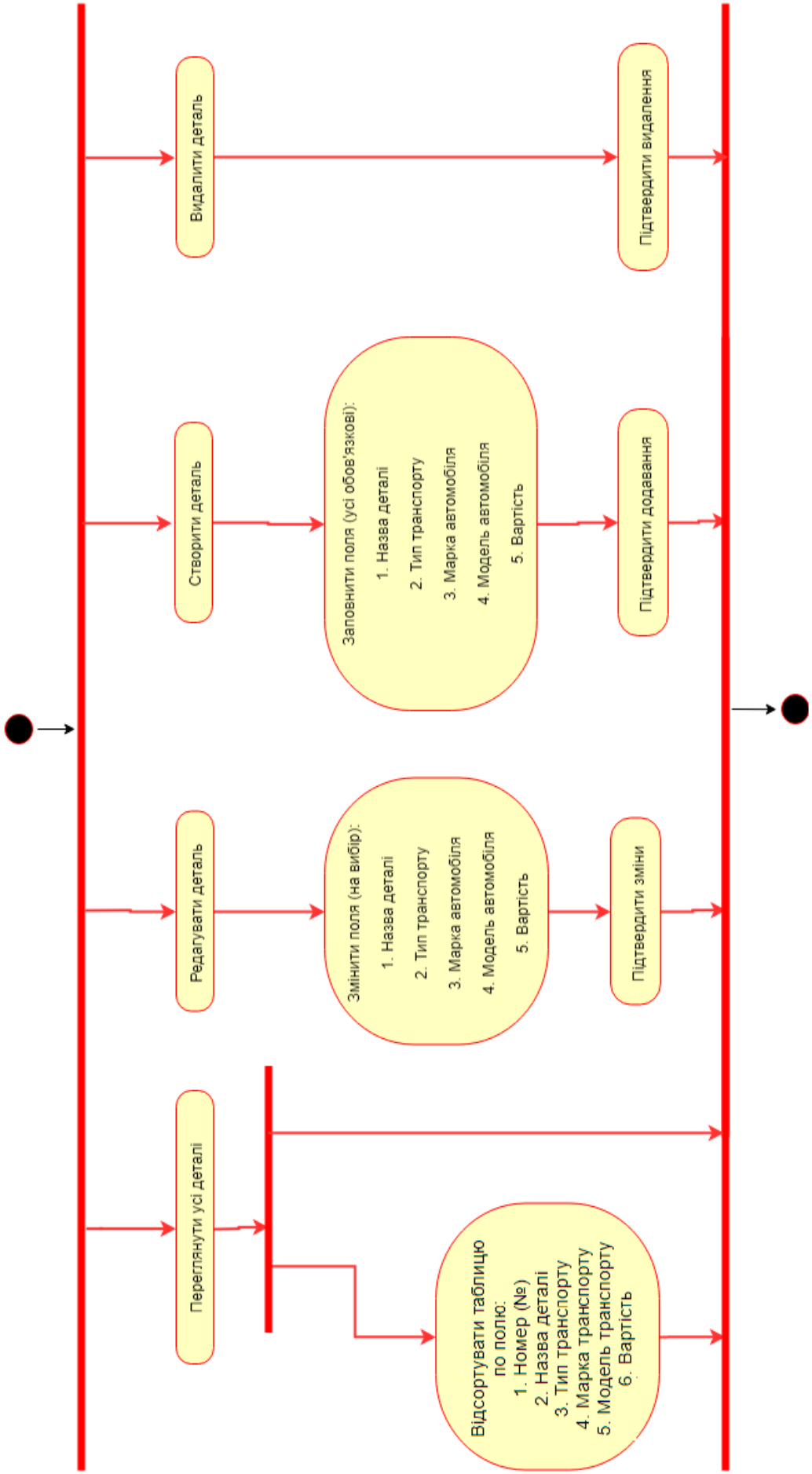


Рисунок 2.12 – Діаграма діяльності модуля «Адміністратор»: управління деталями

2.6.4 Формування процесів управління транспортними засобами в модулі «Адміністратор»

У модулі «Адміністратор» передбачаються наступні процеси управління транспортними засобами: перегляд, створення, редагування та видалення.

Адміністратору доступні для перегляду всі типи, марки та моделі транспортних засобів зареєстровані в системі.

Процес створення чи редагування типу, марки чи моделі транспортного засобу повинен бути простим, швидким та зрозумілим.

Доцільно реалізувати інтерфейс у вигляді форми, яка буде містити в собі рядки для заповнення чи редагування вже існуючої інформації.

Щоб завершити процес створення чи редагування типу транспортного засобу необхідно вказати «Назва типу транспорту».

Щоб завершити процес створення чи редагування марки транспортного засобу необхідно вказати наступні поля: «Тип транспорту» та «Назва марки транспорту».

Щоб завершити процес створення чи редагування моделі транспортного засобу необхідно вказати наступні поля: «Назва марки транспорту» та «Назва моделі транспорту».

Форму потрібно налаштувати таким чином, щоб адміністратору було зрозуміло, які данні очікує система та куди їх вносити, які поля є обов’язковими і які можна проігнорувати.

Також необхідно контролювати процес відправки форми, чи введено мінімум інформації, необхідний для завершення процесу створення чи редагування типу, марки або моделі транспортного засобу, чи відповідають формати даних тим, що очікуються системою.

В модулі «Адміністратор» передбачена можливість видалення типів, марок чи моделей транспортних засобів.

Також потрібно реалізувати процес обробки та виведення помилок. Користувач повинен одразу зрозуміти неправильність свої дій, прочитавши опис помилки, та досягти бажаного результату з мінімальними втратами часу на вирішення неприємної ситуації.

Слід зазначити, що лише адміністратор має доступ до даної частини системи і лише він може змінювати списки типів, марок чи моделей транспортних засобів з якими працює дана станція технічного обслуговування.

Підсумуємо описані вище процеси управління транспортними засобами доступні адміністратору за допомогою UML діаграми діяльності на рисунку 2.13.

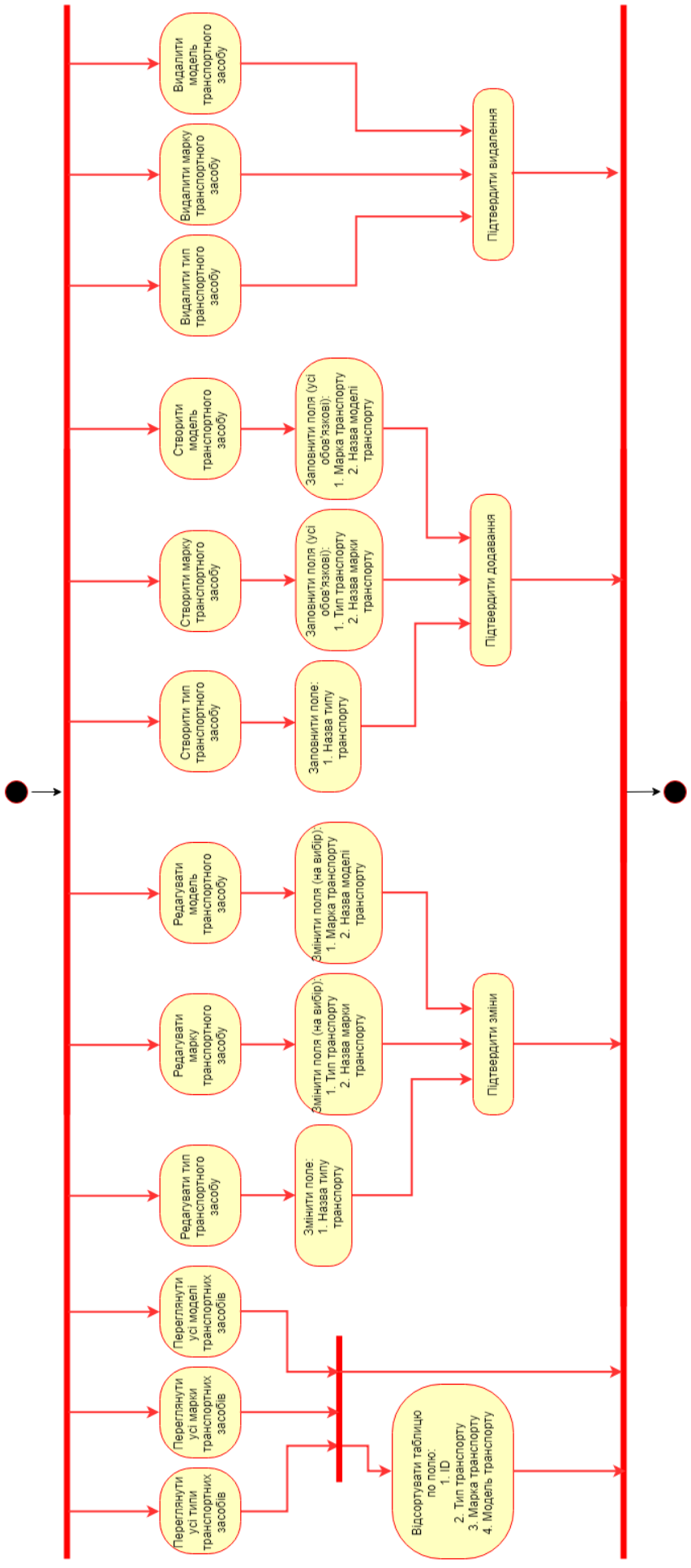


Рисунок 2.13 – Діаграма діяльності модуля «Адміністратор»: управління транспортними засобами

2.6.5 Формування процесів управління задачами в модулі «Адміністратор»

У модулі «Адміністратор» передбачаються наступні процеси управління задачами: перегляд, створення, редагування та видалення. Адміністратору доступні для перегляду всі задачі зареєстровані в системі. Процес створення чи редагування задачі повинен бути простим, швидким та зрозумілим. Доцільно реалізувати інтерфейс у вигляді форми, яка буде містити в собі рядки для заповнення чи редагування вже існуючої інформації. Щоб завершити процес створення чи редагування задачі необхідно вказати наступні поля: «Назва задачі», «Виконавець», «Доручити задачу», «Час початку», «Час виконання» та «Вартість». Також доцільно створити поля, в яких можна вказувати додаткову інформацію, такі як: «Опис задачі», «Коментар» та «Деталі». У адміністратора повинна бути можливість самостійно міняти статус задачі на «Задачу виконано», «Задачу анульовано» та «Задача в очікуванні» з використанням відповідних кнопок. Форму потрібно налаштувати таким чином, щоб адміністратору було зрозуміло, які данні очікує система та куди їх вносити, які поля є обов’язковими і які можна проігнорувати. Також необхідно контролювати процес відправки форми, чи введено мінімум інформації, необхідний для завершення процесу створення чи редагування задачі, чи відповідають формати даних тим, що очікуються системою. У модулі «Адміністратор» передбачена можливість видалення задач. Слід зазначити, що лише адміністратор має повноваження видаляти будь-які задачі зареєстровані в системі. Також потрібно реалізувати процес обробки та виведення помилок. Користувач повинен одразу зрозуміти неправильність свої дій, прочитавши опис помилки, та досягти бажаного результату з мінімальними втратами часу на вирішення неприємної ситуації.
Підсумуємо описані вище процеси управління задачами доступні адміністратору за допомогою UML діаграми діяльності на рисунку 2.14

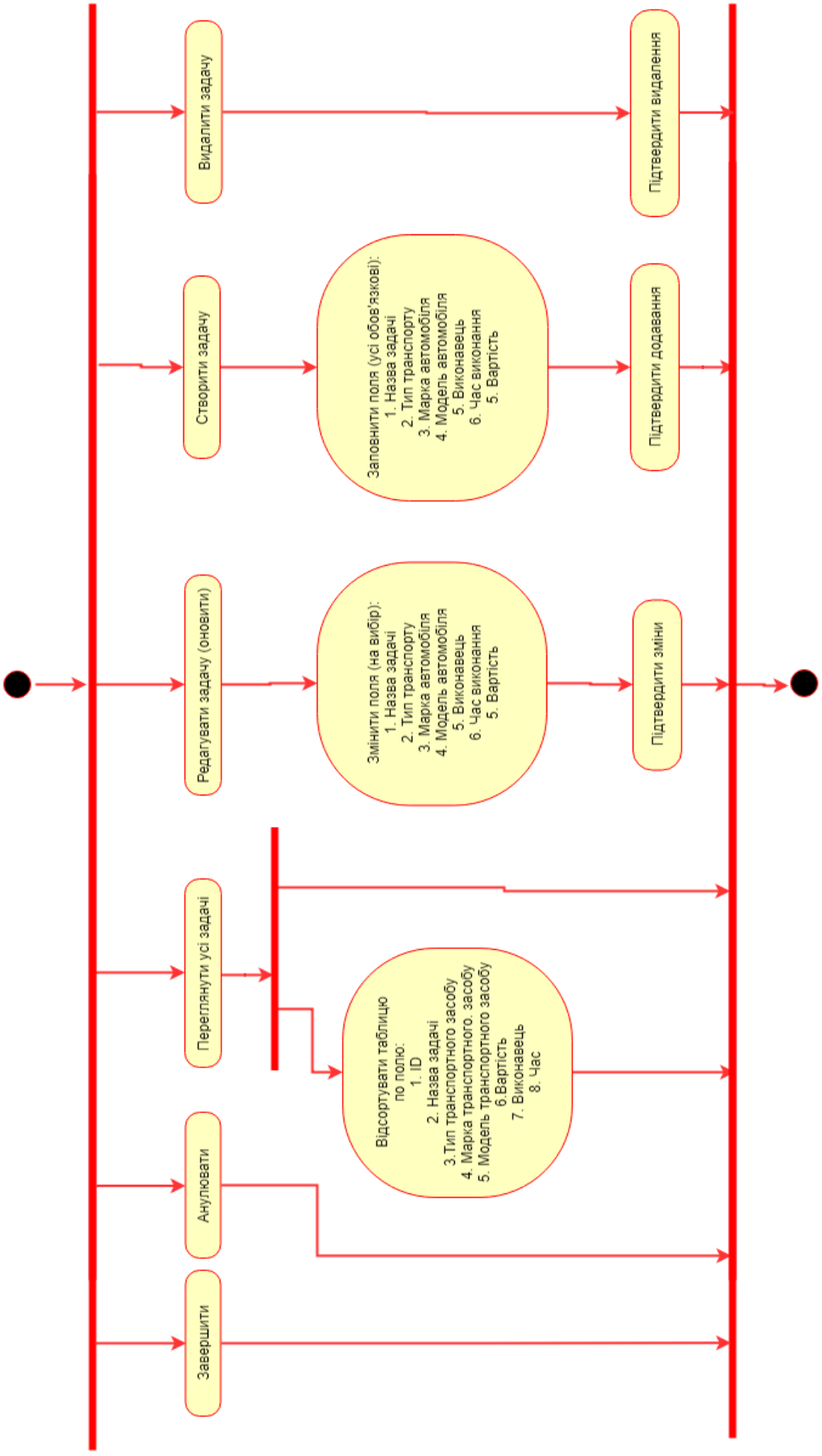


Рисунок 2.14 – Діаграма діяльності модуля «Адміністратор»: управління задачами

2.6.6 Формування процесів управління замовленнями в модулі «Адміністратор»

У модулі «Адміністратор» передбачаються наступні процеси управління замовленнями: перегляд, створення, редагування та видалення.

Адміністратору доступні для перегляду всі замовлення зареєстровані в системі. Процес створення чи редагування замовлення повинен бути простим, швидким та зрозумілим. Доцільно реалізувати інтерфейс у вигляді форми, яка буде містити в собі рядки для заповнення чи редагування вже існуючої інформації.

Щоб завершити процес створення чи редагування замовлення необхідно вказати наступні поля: «Тип транспорту», «Марка транспорту», «Модель транспорту», «Клієнт», «Назва замовлення», «Час початку» та «Орієнтовний час виконання». Також доцільно створити поля, в яких можна вказувати додаткову інформацію, такі як: «Опис замовлення» та «Коментар». У адміністратора повинна бути можливість самостійно міняти статус замовлення на «Замовлення виконано», «Замовлення анульовано», «Замовлення виконується», «Замовлення видано», «Розраховано» та «Не розраховано» з використанням відповідних кнопок.

Форму потрібно налаштувати таким чином, щоб адміністратору було зрозуміло, які данні очікує система та куди їх вносити, які поля є обов’язковими і які можна проігнорувати. Також необхідно контролювати процес відправки форми, чи введено мінімум інформації, необхідний для завершення процесу створення чи редагування замовлення, чи відповідають формати даних тим, що очікуються системою. У модулі «Адміністратор» передбачена можливість видалення замовлення. Слід зазначити, що лише адміністратор має повноваження видаляти будь-які замовлення зареєстровані в системі. Також потрібно реалізувати процес обробки та виведення помилок.

Підсумуємо описані вище процеси управління замовленнями доступні адміністратору за допомогою UML діаграми діяльності на рисунку 2.14.

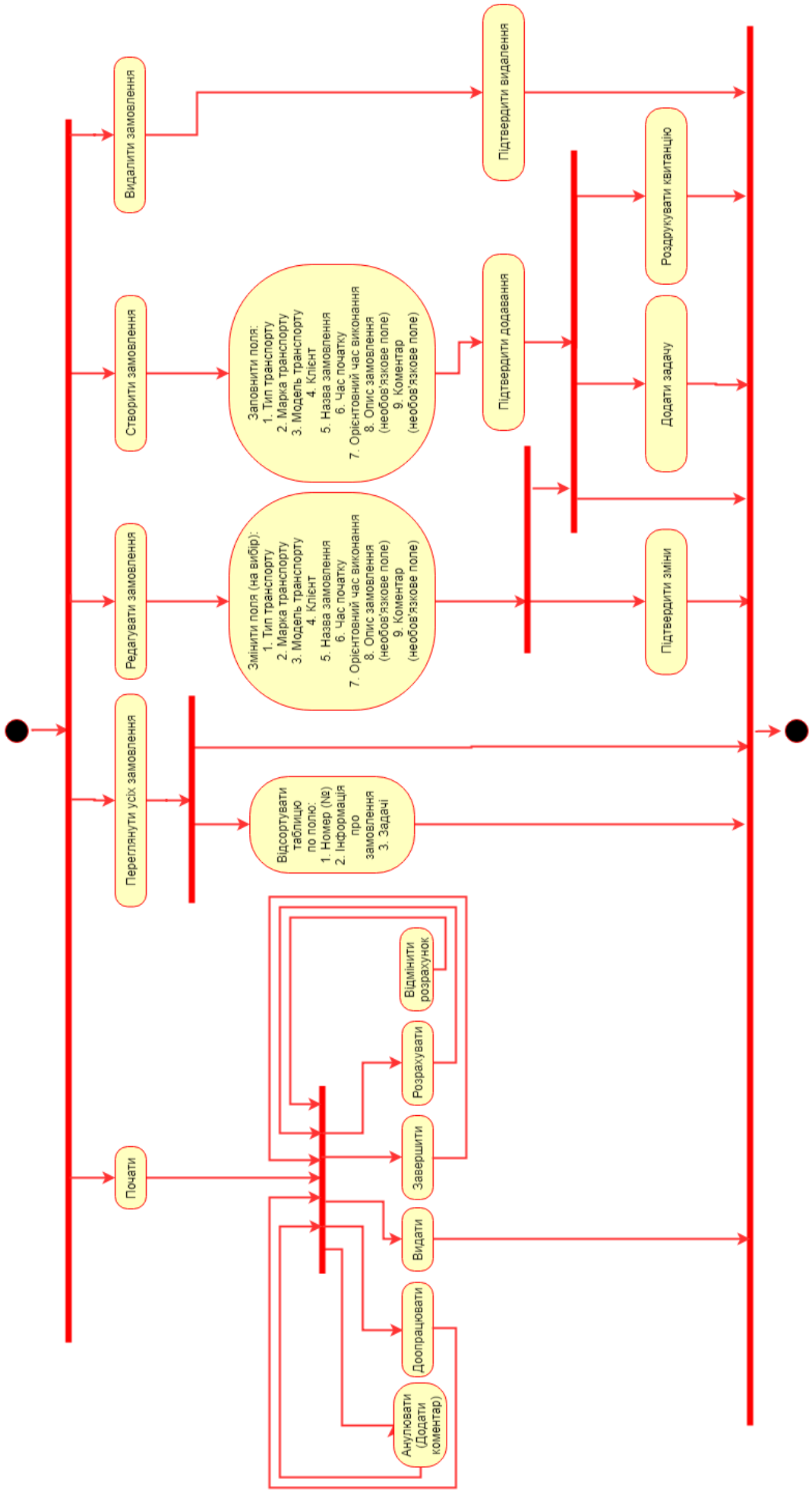


Рисунок 2.14– Діаграма діяльності модуля «Адміністратор»: управління замовленнями

2.6.7 Формування процесів управління статистикою в модулі «Адміністратор»

У модулі «Адміністратор» передбачається лише один процес управління статистикою – перегляд.

Адміністратор повинен мати можливість перегляду статистики по замовленнях, задачах та фінансах даного підприємства. Для зручності пошуку необхідно реалізувати два поля при натисненні на які відкриються календарі. Обравши дати початку та кінця, користувач запустить процес обрахунків статистики за вказаний період.

Статистика по замовленнях повинна бути представленою у вигляді графіків з кривою та складатися з трьох окремих складових: кількість нових замовлень, кількість виконаних замовлень та кількість анульованих замовлень.

Статистика по задачах повинна бути представленою у вигляді таблиці. Таблиця повинна містити список усіх інженерів, з обрахованою кількість задач що очікуються, виконуються, виконані, анульовані, зупинені по кожному з інженерів. Остання колонка таблиці повинна показувати на яку суму грошей певний інженер виконав задач.

Статистика по фінансах повинна бути представленою у вигляді графіка з кривою. Крива показує прибуток станції по кожному дню шляхом суми завершених замовлень.

Слід зазначити, що лише адміністратор має повноваження переглядати статистику в системі. Підсумуємо описані вище процеси управління статистикою доступні адміністратору за допомогою UML діаграми діяльності на рисунку 2.15.

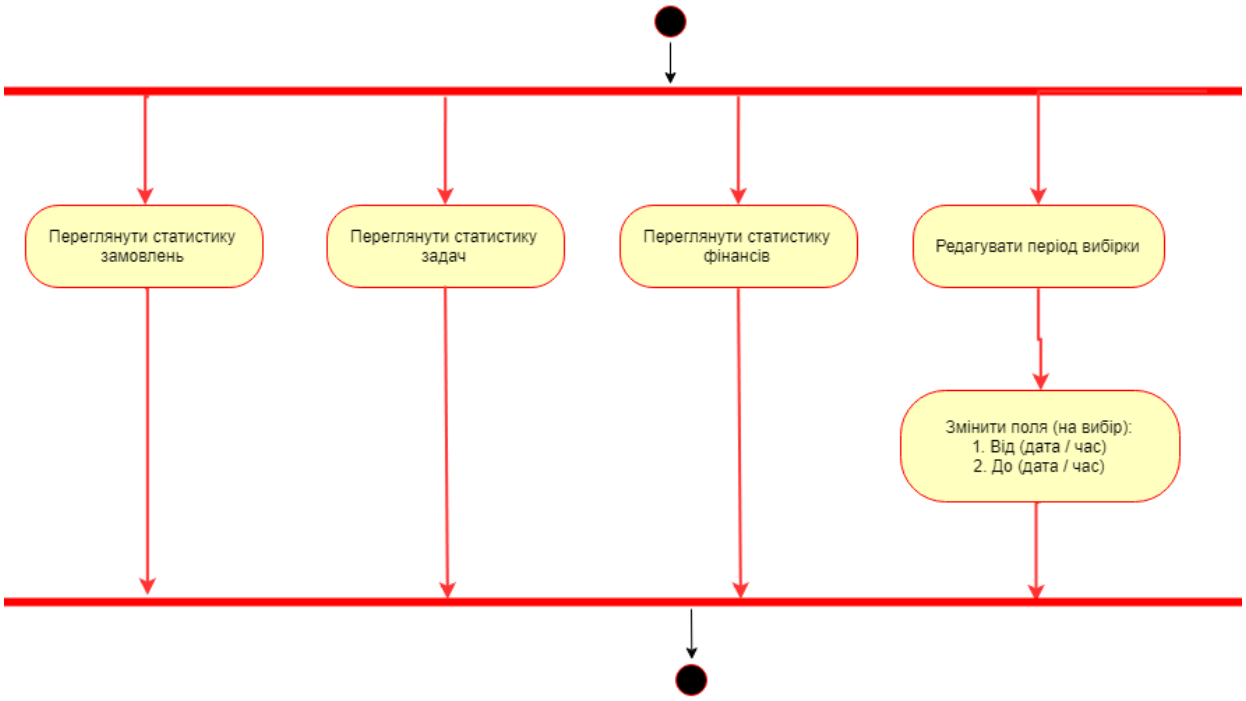


Рисунок 2.15 – Діаграма діяльності модуля «Адміністратор»: управління статистикою

2.7 Формування процесів та методів управління в модулі «Завідувач складом»

Завідувач складом – це особа, яка керує роботою складу з приймання, зберігання та відпускання товарно-матеріальних цінностей, їх розміщення з урахуванням найбільш раціонального використання складських площ, полегшення і прискорення пошуку необхідних матеріалів, інвентарю тощо.

Забезпечує зберігання складованих товарно-матеріальних цінностей, додержання режимів зберігання, правил оформлення та здавання прибутково-видаткових документів.

2.7.1 Формування методів роботи модуля «Завідувач складом»

У рамках розробки інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування, завідувач складом повинен мати доступ до усіх задач активним виконавцем який обраний він. Даний тип користувачів повинен мати повноваження на перегляд та редагування задач.

Завідувач складом повинен мати можливість:

1) Змінити статус задачі шляхом натиснення кнопок «Є на складі» або «Анулювати»;
2) При зміні статусу задачі на «Задачу анульовано», завідувач складом повинен обов’язково вказати причину такого рішення (наприклад необхідні деталі не вдалося знайти і т.д.). Якщо причину не буде вказано, то задачу анулювати буде неможливо;
3) Додавати та редагувати коментарі до задач, активним виконавцем яких обрано його.
4) Коментарі можуть бути використані для внесення інформації про ціну знайдених деталей, а також про терміни їх доставки на станцію технічного обслуговування;

Підсумуємо описані вище методи доступні адміністратору за допомогою UML діаграми прецедентів на рисунку 2.16.

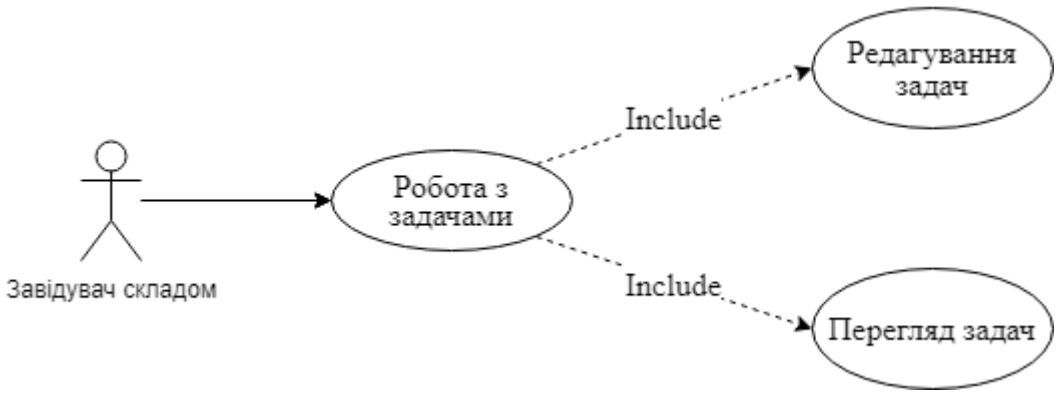


Рисунок 2.16 – Діаграма прецедентів модуля «Завідувач складом»

2.7.2 Формування процесів управління задачами в модулі «Завідувач складом»

У модулі «Завідувач» передбачаються наступні процеси управління задачами: перегляд та редагування.

Завідувачу складом доступні для перегляду лише задачі активним виконавцем яких, являється він сам. Процес редагування задачі повинен бути простим, швидким та зрозумілим. Доцільно реалізувати інтерфейс у вигляді форми, яка буде містити в собі рядки для заповнення чи редагування вже існуючої інформації. Завідувач складом повинен мати можливість змінювати лише поле «Коментар».

Також завідувач складом повинен самостійно міняти статус задачі на «Задачу зупинено» та «Задача в очікуванні» з використанням відповідних кнопок. Форму потрібно налаштувати таким чином, щоб завідувачу складом було зрозуміло, які данні очікує система та куди їх вносити, які поля є обов’язковими і які можна проігнорувати. Також необхідно контролювати процес відправки форми, чи введено мінімум інформації, необхідний для завершення процесу редагування задачі, чи відповідають формати даних тим, що очікуються системою.

Також потрібно реалізувати процес обробки та виведення помилок. Користувач повинен одразу зрозуміти неправильність свої дій, прочитавши опис помилки, та досягти бажаного результату з мінімальними втратами часу на вирішення неприємної ситуації.

Підсумуємо описані вище процеси управління задачами доступні завідувачу складом за допомогою UML діаграми діяльності на рисунку 2.17.

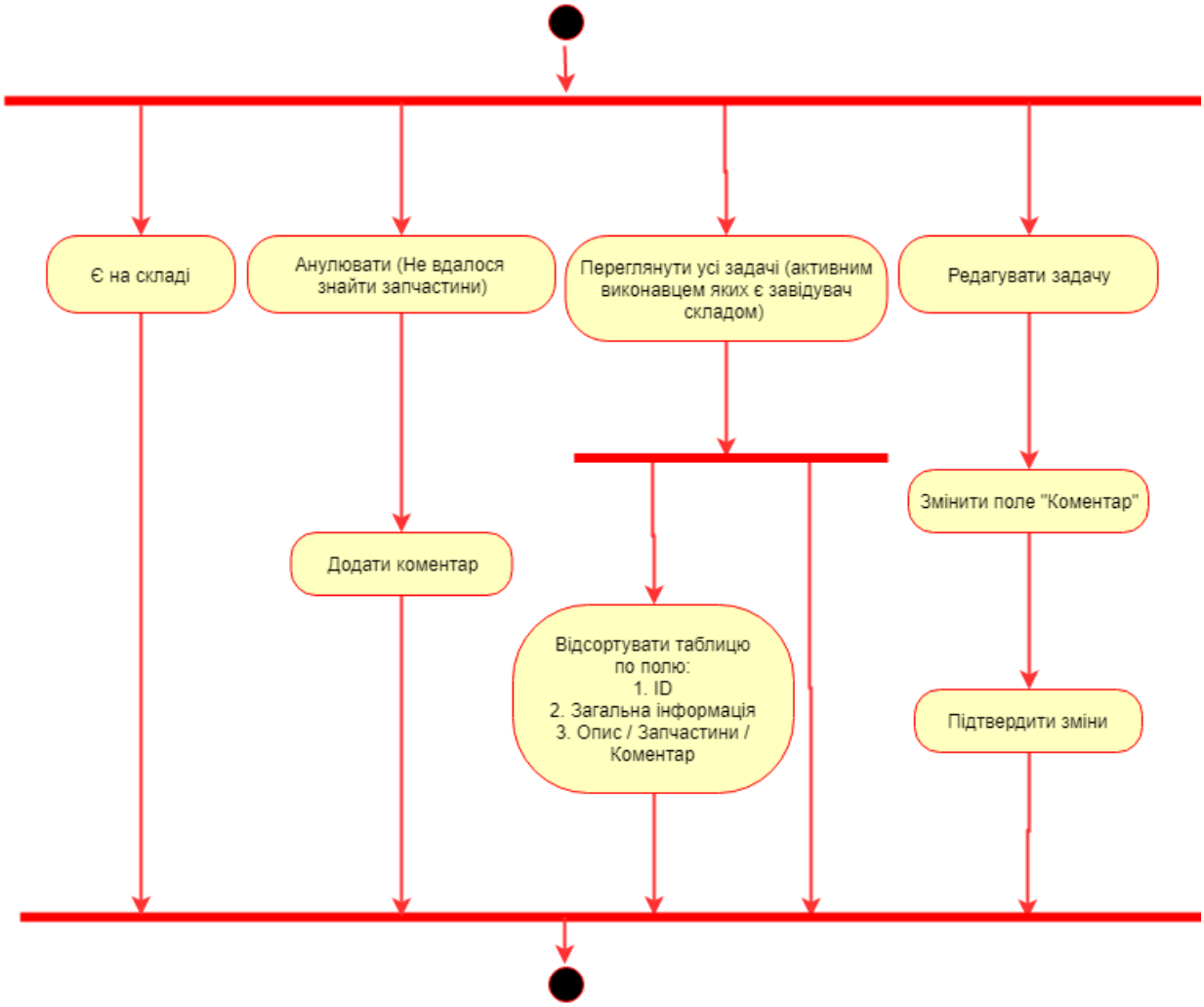


Рисунок 2.17 – Діаграма діяльності модуля «Завідувач складом»: управління задачами

2.8 Формування процесів та методів управління в модулі «Бухгалтер»

Бухгалтер – керівник операційних робітників, контролер законності і правильності здійснення операцій у банку, організатор технології цих операцій не тільки у своєму відділенні банку, але і на підприємствах та в організаціях, які обслуговує даний банк.

Технічною роботою він не займається, а виконує, як правило, лише контрольні функції, тобто перевіряє і підписує документи з деяких операцій, де помилки можуть спричинити втрату коштів і цінностей, або де потрібний контроль за збереженням фінансової дисципліни.

2.8.1 Формування методів роботи модуля «Бухгалтер»

У рамках розробки інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування, бухгалтер повинен мати доступ до усіх замовлень та задач зареєстрованих у системі. Даний тип користувачів повинен мати повноваження на перегляд лише на перегляд інформації. Бухгалтер повинен мати можливість:

1) Переглядати дані усіх замовлень без можливості додавати нові, редагувати чи видаляти вже існуючі;
2) Переглядати дані усіх задач без можливості додавати нові, редагувати чи видаляти вже існуючі;
3) Фільтрувати замовлення по статусах на категорії: «Усі», «Виконані», «Розраховані» та «Нерозраховані»;
4) Підтверджувати факт оплати замовлення шляхом натискання кнопки «Розрахувати»;
5) Відправляти на друк документ, який містить у собі список виконаних робіт, використаних деталей з відповідними цінами за отримані послуги та придбані деталі.

Підсумуємо описані вище методи доступні бухгалтеру за допомогою UML діаграми прецедентів на рисунку 2.18.

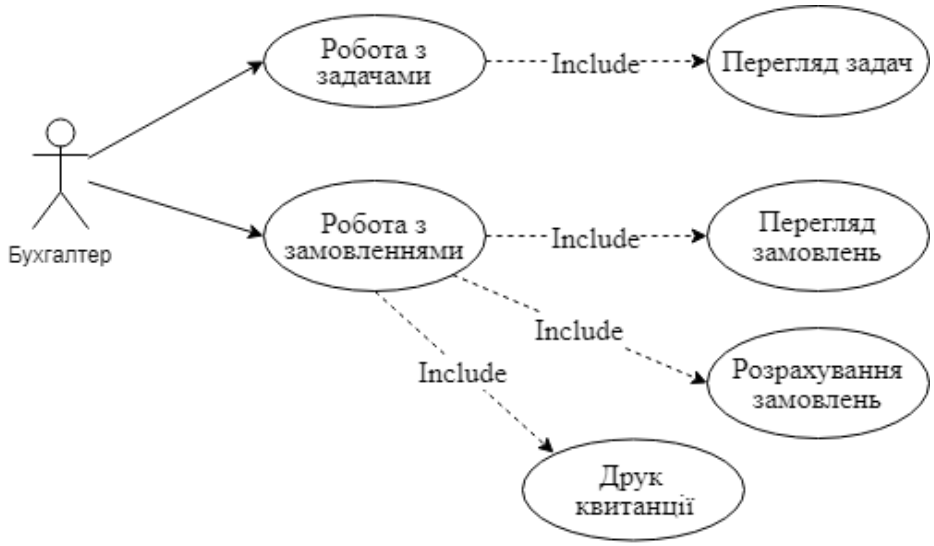


Рисунок 2.18 – Діаграма прецедентів модуля «Бухгалтер»

2.8.2 Формування процесів управління замовленнями та задачами в модулі «Бухгалтер»

У модулі «Бухгалтер» передбачається лише процес перегляду замовлень і задач. Для зручності пошуку необхідних замовлень потрібно реалізувати можливість відфільтрувати їх на групи: «Усі», «Виконанні», «Розраховані» та «Нерозраховані».

Також бухгалтер повинен мати можливість розрахувати клієнта шляхом натискання кнопки «Розрахувати». Оплативши рахунок, бухгалтеру необхідно мати можливість надрукувати квитанцію, яка буде містити список виконаних робіт та затрачених деталей на їх виконання, а також ціни за всі надані послуги.

Підсумуємо описані вище процеси управління задачами доступні завідувачу складом за допомогою UML діаграми діяльності на рисунку 2.19.

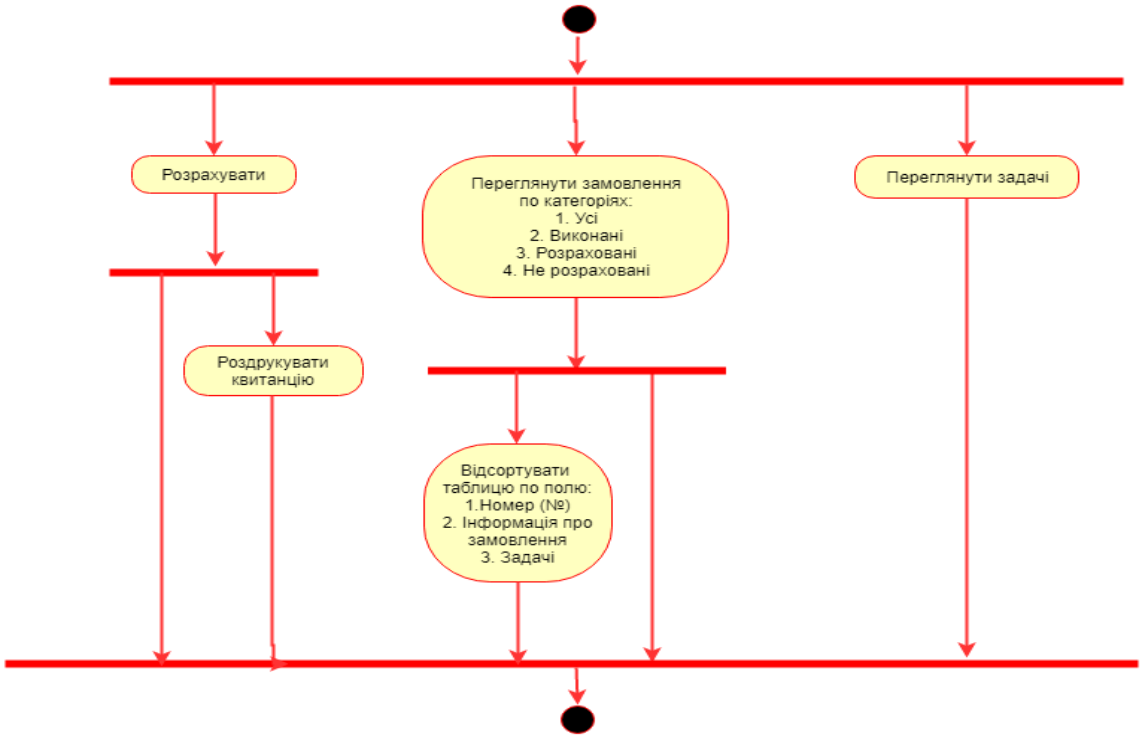


Рисунок 2.19 – Діаграма діяльності модуля «Бухгалтер»: управління замовленнями та задачами

2.9 Опис життєвого циклу замовлення

Основними сутностями інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування є замовлення, задачі які є підскладовими замовлення та деталі, які, в свою чергу, є підскладовими задач.

Переглянемо роботу сервісу на прикладі екземпляру робочого сценарію. У водія зламалась машина, тому він викликав евакуатор, доїхав до сервісу і менеджер починає реєструвати нового користувача в системі (у випадку існування цього клієнта в системі, даний крок пропускається), заповнюючи поля: «Ім'я», «Прізвище», «Компанія», «Адреса», «Контактний номер», «Логін», «Email» та «Пароль».

Після підтвердження додавання замовлення клієнту, на його пошту, що була вказана при його реєстрації, буде надісланий лист-повідомлення про успішну реєстрацію та його логін і пароль для входу в особистий кабінет.

Після успішної реєстрації клієнта в системі, слід почати реєструвати замовлення, заповнюючи усі необхідні поля: «Тип транспорту», «Марка транспорту», «Модель транспорту», «Клієнт», «Назва замовлення», «Час початку» та «Орієнтовний час виконання». При необхідності включення додаткової інформації для реєстрації замовлення можна заповнити необов'язкові поля «Опис замовлення» і «Коментар».

Після підтвердження додавання нового замовлення користувачеві на пошту буде надісланий лист-сповіщення про успішну реєстрацію нового замовлення. Тепер користувач може зайти в систему, ввівши свій логін та пароль і відслідковувати інформацію про процес виконання свого замовлення, статус в якому воно знаходиться, та статус його задач.

Наступним кроком робочого процесу в інформаційній автоматизованій системі робочих місць для співробітників станції технічного обслуговування буде створення задач для замовлення. В рамках роботи системи, замовлення є загальним поняттям, що описує факт необхідності працювати із автомобілем клієнта.

Проте, за для пришвидшення та спрощення виконання замовлення буде реалізоване розділення основного замовлення та підзадачі та розподілення їх між інженерами шляхом доручення кожному з них свої окремих задач.

Отож, створення задачі являє собою процес заповнення менеджером або адміністратором наступних полів: «Назва задачі», «Виконавець», «Доручити задачу», «Час початку», «Час виконання», «Вартість» та «Деталі». Якщо необхідно здійснити автоматичне призначення задач, то поле «Виконавець» треба залишити пустим.

Також при умові існування замовлення робочого процесу в інформаційній автоматизованій системі робочих місць для співробітників станції технічного обслуговування менеджер або адміністратор має змогу виконати автоматичне формування та друк квитанції на вимогу клієнта.

При необхідності вказати додаткову інформацію для реєстрації та виконання задачі, менеджер чи адміністратор може заповнити необов’язкові поля «Опис задачі» та «Коментар» на вимогу клієнта.

Якщо для виконання конкретної задачі потрібно використати додаткові деталі, менеджер або адміністратор повинен заповнити поля «Назва», «Тип» та «Кількість» для кожної деталі.

Після успішного сформування та підтвердження створення кожної задачі для замовлення менеджер має змогу на вимогу клієнта ініціювати автоматичне формування квитанції та її друк. Квитанція містить в собі загальну інформацію про замовлення, список задач які необхідно виконати, перелік деталей, їх ціну та загальну суму, яку необхідно заплатити клієнту за надані послуги.

Після погодження та видачі квитанції клієнту менеджер може ініціювати початок виконання замовлення, натиснувши кнопку «Почати» в результаті чого усі працівники ,яких раніше зареєстрував адміністратор, і за якими тепер закріплені задачі отримують завдання на своїх сторінках (автоматизованих робочих місцях), в які заходять ввівши свої логіни та паролі.

Наступним кроком є робота з задачами інженерів. Виявивши нову задачу, інженер може провести процес редагування замовлення, якщо якісь поля на його думку підлягають зміні: «Доручити задачу», «Опис задачі», «Час початку», «Час виконання», «Вартість» та «Деталі». Після редагування, інженер має змогу розпочати роботу над задачею, натиснувши на кнопку «Почати задачу». Якщо в процесі виконання задачі, інженер виявив нехватку деталей, він маж змогу натиснути кнопку «Відсутні запчастини», заповнивши обов’язкове поле «Коментар» із зазначенням чого саме бракує для виконання задачі. В цей час в менеджера в таблиці замовлень статус задачі змінюється на «Задачу зупинено».

В такому випадку, менеджер зв’язується з клієнтом для повідомлення йому інформації про відсутність необхідних запчастин.

Якщо клієнт погоджується на купівлю запчастин в сервісі, то менеджер передоручає дану задачу завідувачу складом.

Наступним етам життєвого циклу задачі є поява її на власній сторінці завідувача складом, який може або у випадку відсутності деталі на складі повідомити про це менеджера, або знайшовши необхідні запчастини передоручити задачу назад відповідальному за неї інженеру.

Після цього, у власному кабінеті інженера та менеджера статус задачі знову повернеться до «Задача в очікувані». В разі виконання задачі інженер натисне на кнопку «Завершити задачу». А після того, як всі задачі замовлення будуть завершені замовлення змінить автоматично свій статус на «Замовлення виконано» і клієнту на пошту прийде лист-сповіщення про успішне виконання замовлення, а на його сторінці, де він може відслідковувати увесь життєвий цикл свого замовлення та задач, замовлення змінить свій статус на «Замовлення виконано».

Після цього клієнт повинен розплатитись за виконання замовлення в бухгалтера, в разі чого бухгалтер зможе натиснути кнопку «Розрахувати». Після факту успішного розрахунку клієнта, менеджер або адміністратор має натиснути кнопку «Видати»

Замовлення змінить свій статус на «Замовлення видано» в особистих кабінетах адміністратора, менеджера та клієнта, а останній зможе забрати свій відремонтований транспортний засіб.

2.10 Висновок

Отже, виконавши розробку інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування було виконано опис типів користувачів, формування процесів та методів роботи усіх модулів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ АВТОМАТИЗОВАНОЇ СИСТЕМИ РОБОЧИХ МІСЦЬ ДЛЯ СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ

3.1 Реалізація моделі "User"

Для реалізації усіх модулів, а саме для зберігання інформації про користувачів треба створити модель "User" з такими полями: id, ім’я користувача; прізвище користувача; назва компанії; адреса користувача; телефон користувача, email користувача, логін користувача, пароль користувача, id типу користувача, дата створення користувача, дата зміни користувача.

Структура моделі «User» зображена на рисунку 3.1.

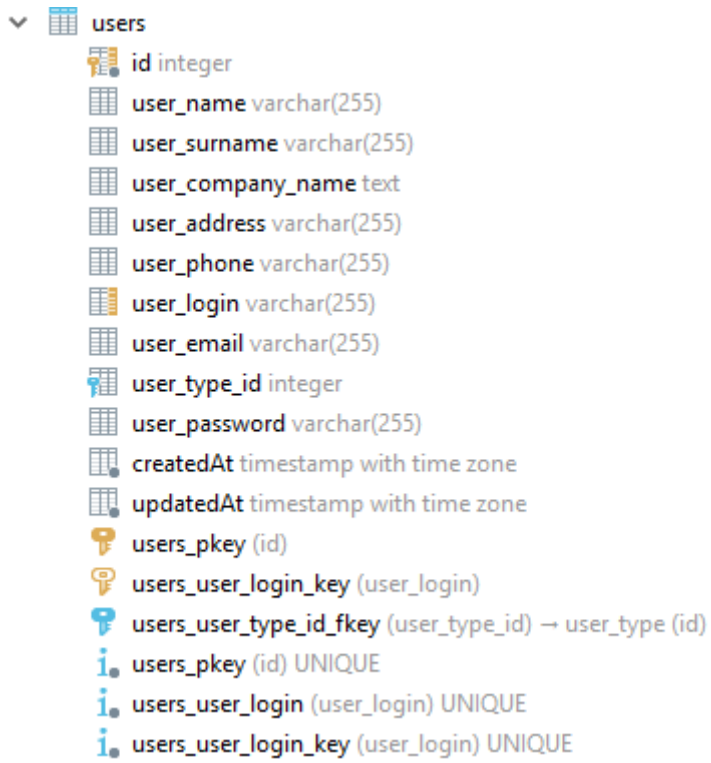


Рисунок 3.1 – Структура моделі «User»

3.2 Реалізація моделі "UserType"

Для реалізації усіх модулів, а саме для зберігання інформації про типи користувача треба створити молель "UserType:" з такими полями: id, назва типу користувача.

Структура моделі «UserType» зображена на рисунку 3.2.

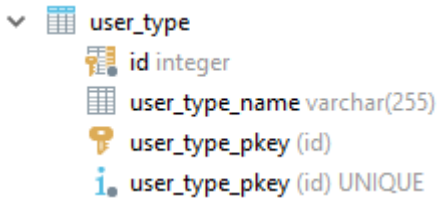


Рисунок 3.2 – Структура моделі «UserType»

3.2 Реалізація моделі "Request"

Для реалізації усіх модулів, а саме для зберігання інформації про замовлення треба створити молель "Request:" з такими полями: id, id клієнта, назва замовлення, опис замовлення, коментар, статус замовлення, почате/ні замовлення, час початку замовлення, розрахунковий час на виконання замовлення, ціна замовлення, тип транспорту, марка транспорту, модель транспорту, видалене/ні замовлення, оплачене/ні замовлення, видане/ні замовлення, дата створення, дата зміни.

Структура моделі «Request» зображена на рисунку 3.3.

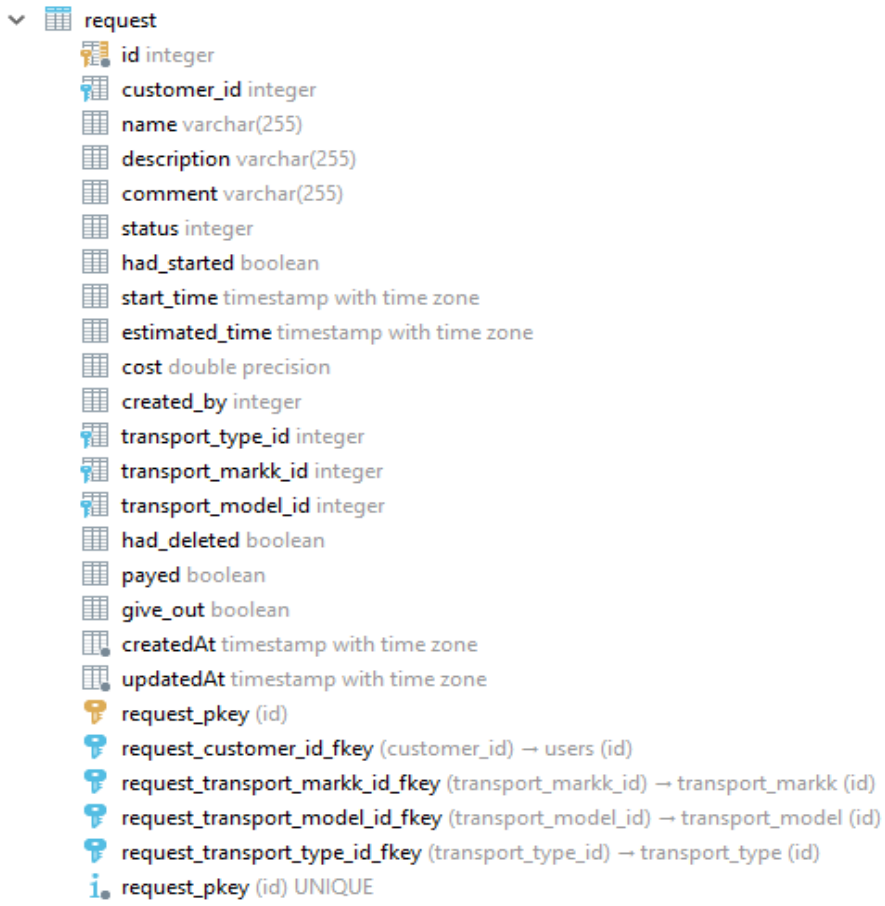


Рисунок 3.3 – Структура моделі «Request»

3.4 Реалізація моделі "RequestHistory".

Для реалізації усіх модулів, а саме для зберігання інформації про історію замовлень треба створити модель "RequestHistory" з такими полями: id, id замовлення, статус замовлення, дата створення, дата зміни. Структура моделі «RequestHistory» зображена на рисунку 3.4.

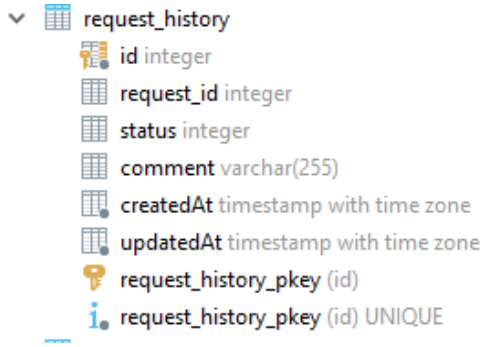


Рисунок 3.4 – Структура моделі «RequestHistory»

3.5 Реалізація моделі "Detail"

Для реалізації усіх модулів, а саме для зберігання інформації про деталі треба створити молель "Detail" з такими полями: id, назва деталі, ціна деталі, id транспортного засобу, id марки транспортного засобу, id моделі транспортного засобу.

Структура моделі «Detail» зображена на рисунку 3.5.

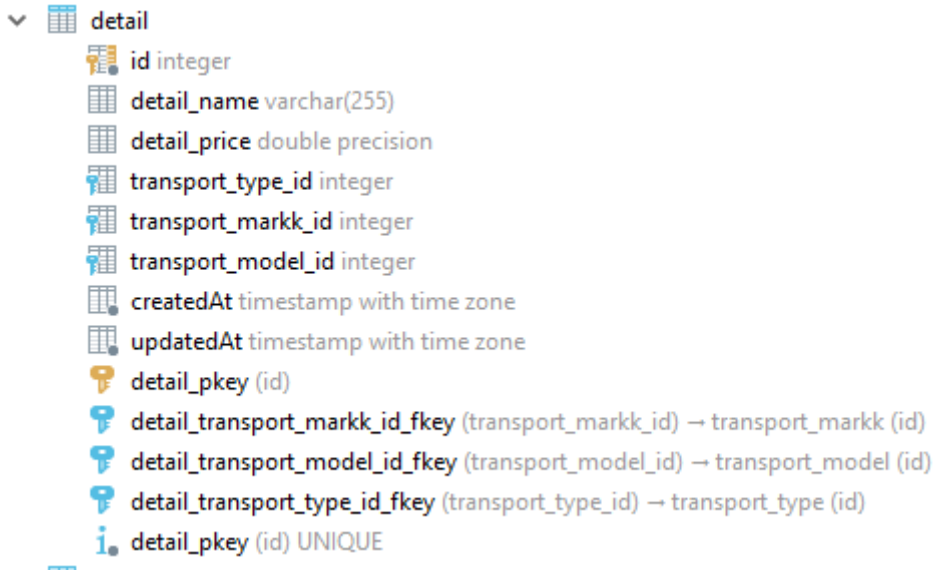


Рисунок 3.5 – Структура моделі «Detail»

3.6 Реалізація моделі «Task»

Модель «Task» містить наступні атрибути: ID; Назва задачі; ID інженера; ID дорученого користувача; Вартість задачі; Час виконання; Час початку; Час кінця; Статус; Коментар; ID замовлення; ID типу задачі; Дата створення; Дата зміни.

Структура моделі «Task» у базі даних зображено на рисунку 3.6.

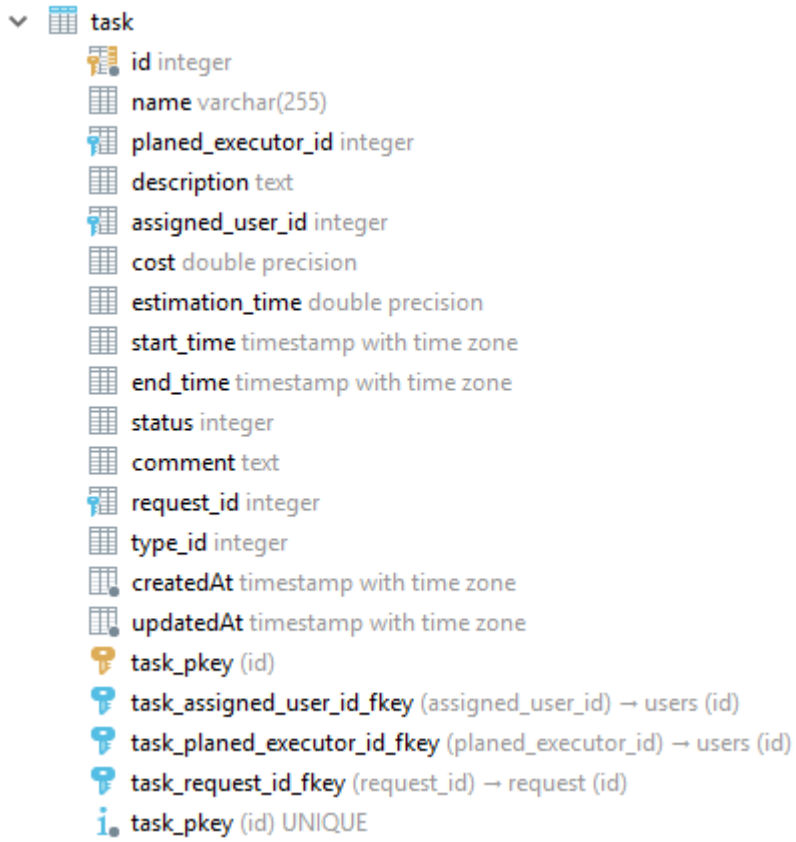


Рисунок 3.6 – Структура моделі «Task»

3.7 Реалізація моделі «TaskType»

Модель «TaskType» містить наступні атрибути: ID; Назва типу задачі; Тип транспорту; Марка транспорту; Модель транспорту; Вартість; Час виконання; ID запланованого виконавця; Дата створення; Дата зміни.

Структура моделі «TaskType» у базі даних зображено на рисунку 3.7.

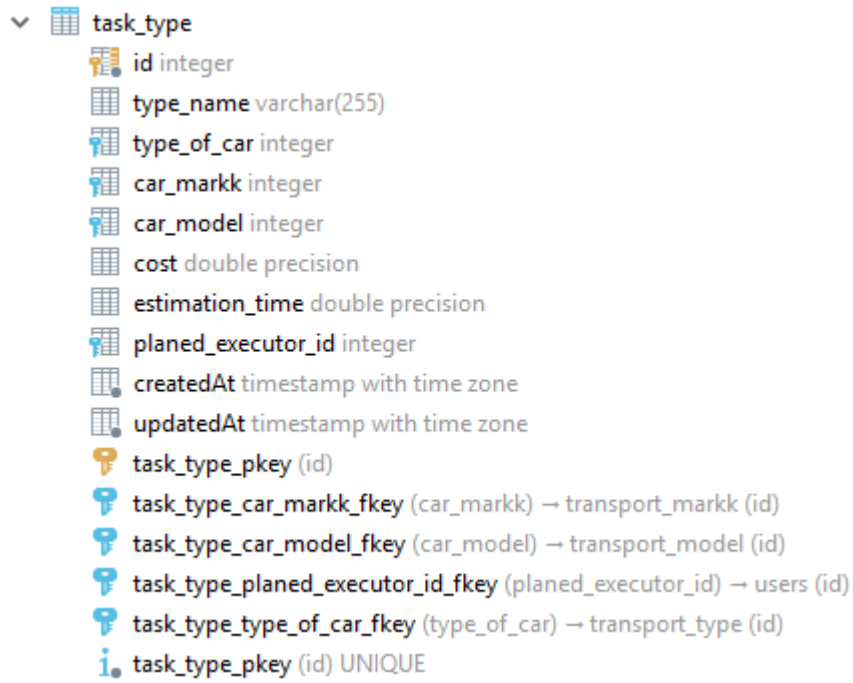


Рисунок 3.7 – Структура моделі «TaskType»

3.8 Реалізація моделі «TaskDetail»

Модель «TaskDetail» містить наступні атрибути: ID; ID задачі; ID деталі; Кількість деталей; Тип деталі; Дата створення; Дата зміни.

Структура моделі «TaskDetail» у базі даних зображено на рисунку 3.8.

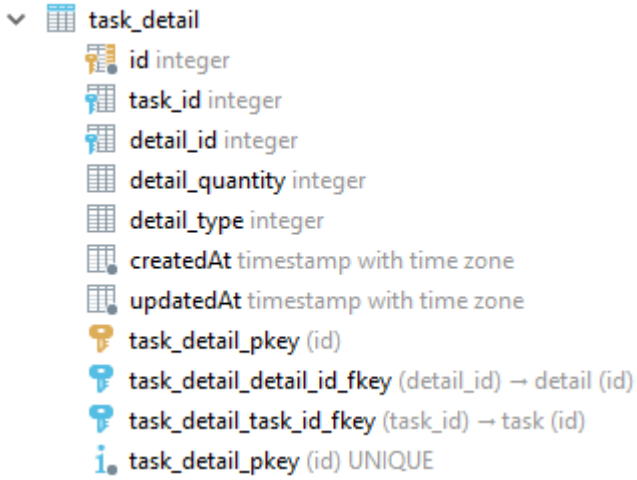


Рисунок 3.8 – Структура моделі «TaskDetail»

3.9 Реалізація моделі «TransportType»

Модель «TransportType» містить наступні атрибути: ID; Назва типу транспорту; Дата створення; Дата зміни.

Структура моделі «TransportType» у базі даних зображено на рисунку 3.9.

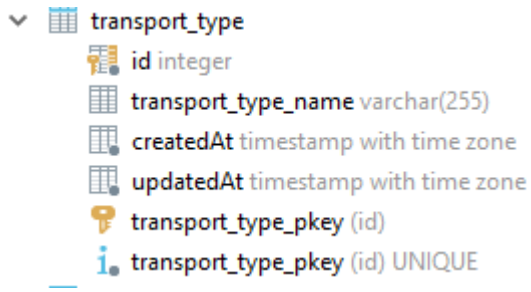


Рисунок 3.9 – Структура моделі «TransportType»

3.10 Реалізація моделі «TransportMarkk»

Модель «TransportMarkk» містить наступні атрибути: ID; ID типу транспорту; Назва марки транспорту; Дата створення; Дата зміни.

Структура моделі «TransportMarkk» у базі даних зображено на рисунку 3.10.

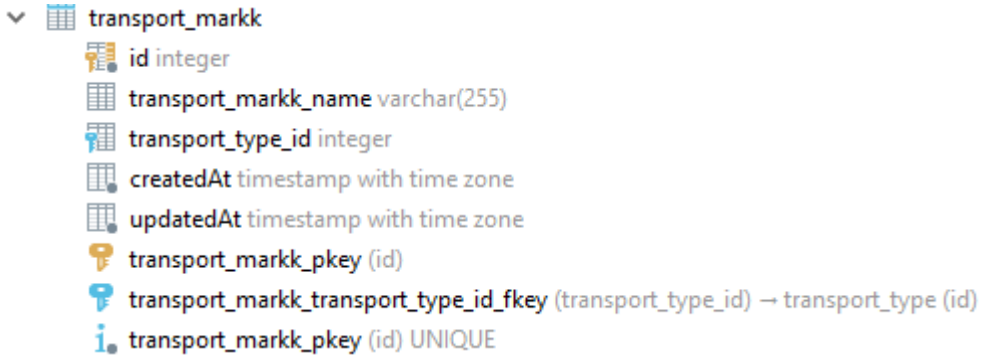


Рисунок 3.10 – Структура моделі «TransportMarkk»

3.11 Реалізація моделі «TransportModel»

Модель «TransportModel» містить наступні атрибути: ID; ID марки транспорту; Назва моделі транспорту; Дата створення; Дата зміни.

Структура моделі «TransportModel» у базі даних зображено на рисунку 3.11.

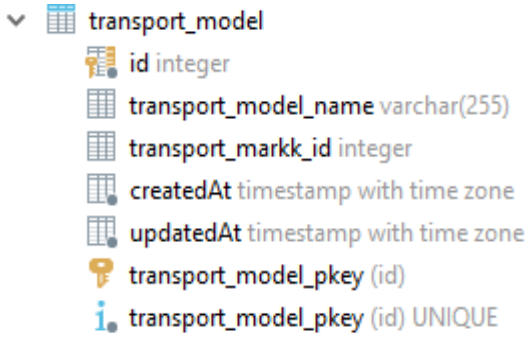


Рисунок 3.11 – Структура моделі «TransportModel»

3.12 Висновок

Отже, у даному розділі було описано структури моделей кожного об’єкта бази даних та продемонстровано їх схеми моделей.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності [20].

Магістерська кваліфікаційна робота за темою “Інформаційна автоматизована система робочих місць для співробітників станції технічного обслуговування” передбачає розробку сайту на сам перед для українських станцій технічного обслуговування та різного роду діагностичних сервісів усіх видів автотранспорту із підтримкою наступних посад на підприємстві:

- Адміністратор СТО – головна роль в системі, який має доступ до всієї інформації підприємства. В нього наявна можливість управління, зміни, створення та видалення усіх сутностей системи: користувачі, замовлення, задачі, транспортні засоби, деталі та ін.
- Менеджер – відповідальний за формування замовлення та розподіл задач між виконавцями.
- Інженер – власне, виконавець розподілених задач, який може управляти статусом своїх робіт: починати, завершувати, робити запити на додаткові запчастини.
- Завідувач складом – роль, відповідальна за ведення складського обліку і видачі запчастин інженеру.
- Бухгалтер – відповідальний за перевірку оплати замовлення клієнтом та видачі квитанції про оплату наданих на СТО послуг.

Веб сайт буде задовольняти наступні потреби:

- Заміну паперового документообігу на підприємстві єдиною системою управління та збереження усієї інформації на віддаленому сервері;
- Організацію управління усіма співробітниками підприємства через власний кабінет кожного робітника на сайті а також наявності окремої сторінки зі статистикою для прогнозування прибутку;
- Максимально відкриту взаємодію з клієнтом авто сервісу за рахунок наявності власної сторінки у останнього для отримання інформації про його замовлення, статус підзадач, прогрес виконання, виконавців, а також орієнтовний час закінчення роботи та ціну за виконання;
- Можливість автоматичного призначення задач для інженерів для оптимізації робочого процесу та пришвидшення виконання замовлень, беручи за основу кваліфікацію, час інженера, а також – його ставку та складність задач.

Для об’єктивного оцінювання комерційного потенціалу та необхідності в розробці такої системи проведемо порівняльний аналіз у вигляді таблиці між наявним на ринку конкурентом в галузі автоматизації управління підприємствами, такими як сервіси технічного обслуговування. Порівняльний аналіз між розробкою та системою «1С: Підприємство» відображений в Таблиці 4.1.

Таблиця 4.1 – Порівняльний аналіз між розробкою та системою «1С: Підприємство»

Критерій	Розробка	1С: Підприємство
Необхідні обчислювальні ресурси	1 сервер для хостингу сайту та планшети (або моноблоки чи смартфони) для співробітників з доступом в Інтернет та мінімальними обчислювальним характеристиками	1 сервер для хостингу ядра системи та персональні комп’ютери для кожного співробітника, з потужними характеристиками для оптимальної роботи системи
Середовище для обчислювальних	Немає необхідності в дотримані додаткових стандартів для	Кожен персональний комп’ютер повинен мати

ресурсів	утримання планшетів чи моноблоків	спеціально обладнану кімнату із дотриманням стандартів робочого місця, пожежної безпеки та ін.
Доступність рішення	Необхідна наявність інтернету або локальної мережі	Наявні адміністративні проблеми з доступом в Україні
Функціонал	Наявність інтегрованих алгоритмів автоматичного призначення задач виконавцям	Можливість лише ручного призначення задач виконавцям

В результаті порівняльного аналізу даної розробки із наявним на ринку конкурентом в галузі автоматизації управління підприємствами можна пересвідчитись в доцільності розробки та необхідності в реалізації такої системи.

Проведемо оцінювання комерційного потенціалу даної розробки. Для проведення технологічного аудиту було залучено 3-х незалежних експертів: Козачко Олексій Миколайович – керівник магістерської кваліфікаційної роботи, доцент, кандидат технічних наук; Крижановський Євгеній Миколайович; Ящолт Андрій Русланович.

В таблиці 4.2 наведено критерії оцінювання комерційного потенціалу розробки та їх оцінки в балах.

Таблиця 4.2 – Критерії оцінювання наукового та комерційного потенціалу розробки та їх бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження таблиці 4.2

Ринкові перспективи					
5	Експлуатацій ні витрати значно вищі, ніж в аналогів	Експлуатацій ні витрати дещо вищі, ніж в аналогів	Експлуатацій ні витрати на рівні експлуатаційн их витрат аналогів	Експлуатацій ні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Практична здійсненність					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовую ться у військово-промисловом у комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовують ся у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років.	Термін реалізації ідеї від 3-х до 5-ти років.	Термін реалізації ідеї менше 3-х років.	Термін реалізації ідеї менше 3-х років.
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки потрібно звести в таблицю за зразком таблиці 4.3.

Таблиця 4.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Козачко О. М	Крижановський Є.М.	Ящолт А.Р.
	Бали, виставлені експертами:		
1	4	3	4
2	3	3	2
3	4	4	3
4	4	2	3
5	3	3	4
6	2	4	2
7	4	2	3
8	3	4	3
9	3	2	3
10	4	4	3
11	3	3	2
12	4	3	4
Сума балів	$СБ_1$=41	$СБ_2$=37	$СБ_1$=36
Середньоарифметична сума балів СБ	$\overline{СБ}\frac{\sum_1^3 СБ_i}{3} = \frac{114}{3} = 38.$		

За даними таблиці 4.3 можна зробити висновок, щодо рівня комерційного потенціалу розробки. Зважимо на результат й порівняємо його з рівнями комерційного потенціалу розробки, що представлено в таблиці 4.4.

Таблиця 4.4 – Рівні комерційного потенціалу розробки.

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Рівень комерційного потенціалу розробки, становить 38 балів, що відповідає рівню «вище середнього».

4.2 Прогнозування витрат на виконання магістерської роботи та впровадження результатів

Проведемо прогнозування витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи для розробки програмного забезпечення, яке складається з таких етапів:

1-й етап: розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи;

2-й етап: розрахунок загальних витрат на виконання даної роботи;

3-й етап: прогнозування загальних витрат на виконання та впровадження результатів даної роботи.

1. Виконаємо розрахунок витрат приймаючи до уваги те, що для розробки інформаційної технології було залучено одного розробника програмного забезпечення. Основна заробітна плата кожного із розробників (дослідників) $З_о$, якщо вони працюють в наукових установах бюджетної сфери:

$$З_о = \frac{М}{Т_р} \cdot t \text{ [грн]}, \qquad (4.1)$$

де М – місячний посадовий оклад конкретного розробника (інженера, до-слідника, науковця тощо), грн;

$Т_р$ – число робочих днів в місяці; приблизно $Т_р$ = (21…23) дні;

t – число робочих днів роботи розробника (дослідника), розробка програмного забезпечення триває 80 днів.

Зроблені розрахунки внесені до таблиці 4.5.

Таблиця 4.5 – Основна заробітна плата розробників.

Найменування посади виконавця	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на оплату праці, грн.
Програміст	10 000	454.54	80	36 363,2
Всього				36 363,2

2. Додаткова заробітна плата $З_Д$ всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12%) від суми основної заробітної плати розробників та робітників, розраховується за формулою:

$$З_Д = 0.10 \cdot 36\ 363{,}2 = 3\ 636{,}32(грн).$$

3. Нарахування на заробітну плату $Н_{ЗП}$ розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується за формулою:

$$Н_{зп} = (З_0 + З_Д) \cdot \frac{\beta}{100}\ [грн], \qquad (4.2)$$

де $З_0$ – основна заробітна плата розробника, грн.;

$З_Д$ – додаткова заробітна плата розробника, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування – 37,3%.

$$Н_{ЗП} = (36\ 363{,}2 + 3\ 636{,}32) \cdot\ 0{,}22 = 8\ 799{,}89(грн).$$

4. Амортизація обладнання, комп'ютерів та приміщень А, які використовувались під час (чи для) виконання даного етапу роботи.

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

У спрощеному вигляді амортизаційні відрахування А в цілому бути розраховані за формулою:

$$A = \frac{Ц \cdot Т}{12 \cdot Т_В}[грн], \quad (4.3)$$

Де Ц – загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн;

Т – фактична тривалість використання, міс;

$Т_В$– термін, використання обладнання, приміщень тощо, місяці, роки.

Зроблені розрахунки наведено в таблиці 4.6

Таблиця 4.6 – Амортизаційні відрахування

Найменування	Балансова вартість, грн	Термін використання, роки	Фактична тривалість використання, міс.	Величина амортизаційних відрахувань, грн
Офісне приміщення	10000	1	4	2500
Ноутбук	21000	1	4	2100
Всього				4600

5. Інформацію про матеріали, що використовуються при виготовленні даного інноваційного продукту внесено до таблиці 4.7.

Таблиця 4.7 – Матеріали, що використовуються при виготовленні даного продукту

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено, шт.	Вартість витраченого матеріалу з урахуванням доставки, грн
Папір (пачка)	90,00	1	99,00
Канцтовари	40,00	1	44,00

Всього	143,00

Під час розробки програмного продукту використовували безкоштовні програмні продукти. Здійснювалась оплата домену та хостингу для розміщення сайту. Програмні послуги, що використовуються при виготовленні даного продукту внесено до таблиці 4.8.

Таблиця 4.8 – Програмні послуги, що використовуються при виготовленні даного продукту

Найменування послуги	Ціна за 1 міс., грн.	Кількість місяців	Вартість використаної послуги.
Домен	50	4	200
Хостинг	100	4	400
Всього	600		

6. Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п \ [грн], \quad (4.4)$$

де В – вартість 1 кВт-год. електроенергії, 1,8 грн/кВт;

П – установлена потужність обладнання, кВт;

Ф – фактична кількість годин роботи обладнання, годин;

$К_п$ – коефіцієнт використання потужності;

Потужність використовуваного комп'ютера становить П=0.6 кВт.

Фактична кількість годин роботи обладнання – 640 год (80 робочих днів по 8 годин на день).

$$B_e = 1{,}8 \cdot 0{,}6 \cdot 640 \cdot 0{,}6 = 414{,}72 \text{ (грн)}.$$

7. Сума всіх попередніх статей витрат дає витрати на виконання даної частини розділу роботи В.

$$B = 36\,363{,}2 + 3\,636{,}32 + 8\,799{,}89 + 4600 + 143 + 414{,}72 + 600$$
$$= 54\,557{,}13 \text{ (грн)}.$$

2-й етап: розрахунок загальних витрат на виконання даної роботи.

$$B_{заг} = \frac{B}{\alpha}[грн], \tag{4.5}$$

$$\text{В}_{\text{заг}} = \frac{54\,557{,}13}{1} = 54\,557{,}13 \text{ (грн)},$$

3-й етап. Прогнозування загальних витрат на виконання та впровадження результатів виконаної роботи. Прогнозування витрат ЗВ на виконання та впровадження виконаної роботи здійснюється за формулою:

$$\text{ЗВ} = \frac{\text{В}_{\text{заг}}}{\beta}[грн], \tag{4.7}$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Так, як розробка знаходиться:

- на стадії розробки дослідного зразка, то $\beta \approx 0{,}5$
- на стадії технічного проектування , то $\beta \approx 0{,}2$
- на стадії розробки конструкторської документації то $\beta \approx 0{,}3$
- на стадії розробки технології, то $\beta \approx 0{,}4$
- на стадії науково-дослідних робіт, то $\beta \approx 0{,}1$
- на стадії промислового зразка, $\beta \approx 0{,}7$
- на стадії впровадження, то $\beta \approx 0{,}9$

$$ЗВ = \frac{54\ 557,13}{0,7} = 77\ 938,76 \text{ (грн)}.$$

Отже, прогноз загальних витрат на виконання та впровадження результатів становить 77 938,76 грн.

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У даному підрозділі проведемо кількісне прогнозування, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. В умовах ринку узагальнюючим позитивним результатом, що його отримує підприємство від впровадження результатів тієї чи іншої розробки, є збільшення чистого прибутку підприємства. Зростання чистого прибутку можна оцінити у теперішній вартості грошей.

Зростання чистого прибутку забезпечить підприємству надходження додаткових коштів, які дозволять покращити фінансові результати діяльності.

Виконання даної наукової роботи та впровадження її результатів складає приблизно 1 рік.

Позитивні результати від впровадження розробки очікуються вже в перший рік впровадження.

Проведемо детальніше прогнозування позитивних результатів та кількісне їх оцінювання по роках.

Обчислимо збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_я = \sum_1^n (\Delta П_я \cdot N + П_я \cdot \Delta N)_n \text{ [грн]}, \quad (4.8)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

Припустимо, що внаслідок впровадження результатів наукової розробки покращується якість, що дозволяє збільшити прибуток підприємства на 100грн, а кількість одиниць реалізованої послуги збільшиться: протягом першого року – на 200 од., протягом другого року – ще на 300 од., протягом третього року – ще на 350 од. Орієнтовно: реалізація послуг до впровадження результатів наукової розробки складала 25 шт., а її прибуток підприємства – 500рн.

Спрогнозуємо збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства $\Delta П_1$ протягом першого року складе:

$$\Delta П_1 = 25 \cdot 500 + (500+100) \cdot 200 = 132\,500 \text{ (грн)}.$$

Обчислимо збільшення чистого прибутку підприємства $\Delta П_2$ протягом другого року:

$$\Delta П_2 = 25 \cdot 500 + (500+100) \cdot (200+300) = 312\,500 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства $\Delta П_3$ протягом третього року становитиме:

$$\Delta П_3 = 25 \cdot 500 + (500+100) \cdot (200+300+350) = 522\ 500 \text{ (грн)}.$$

Отже, розрахунки показують, що відповідно прогнозуванню комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку підприємства.

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Основними показниками, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахунок ефективності вкладених інвестицій передбачає:

1-й крок. Розрахунок теперішньої вартості інвестицій PV, що вкладаються в наукову розробку. Такою вартістю ми можемо вважати прогнозовану величину загальних витрат ЗВ на виконання та впровадження результатів НДДКР, тобто ЗВ = PV = 77 938,76 (грн).

2-й крок. Розрахунок очікуваного збільшення прибутку $\Delta П_i$, що його отримає підприємство (організація) від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження проведено вище.

3-й крок. Будуємо вісь часу, на якій відображаємо всі платежі (інвестиції та прибутки), що мають місце під час виконання науково-дослідної роботи та впровадження її результатів. Платежі показуємо у ті терміни, коли вони здійснюються.

Припустимо, що загальні витрати ЗВ на виконання та впровадження результатів НДДКР (або теперішня вартість інвестицій PV) дорівнює 77 938,76 грн. Результати вкладених у наукову розробку інвестицій почнуть з'являтися протягом трьох років.

Ці результати виявляться у тому, що у першому році підприємство отримає збільшення чистого прибутку на 132 500 грн відносно базового року, у другому році – збільшення чистого прибутку на 312 500 грн (відносно базового року), у третьому році – збільшення чистого прибутку на 522 500 грн (відносно базового року).

Тоді рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, наведений на рис. 4.1.

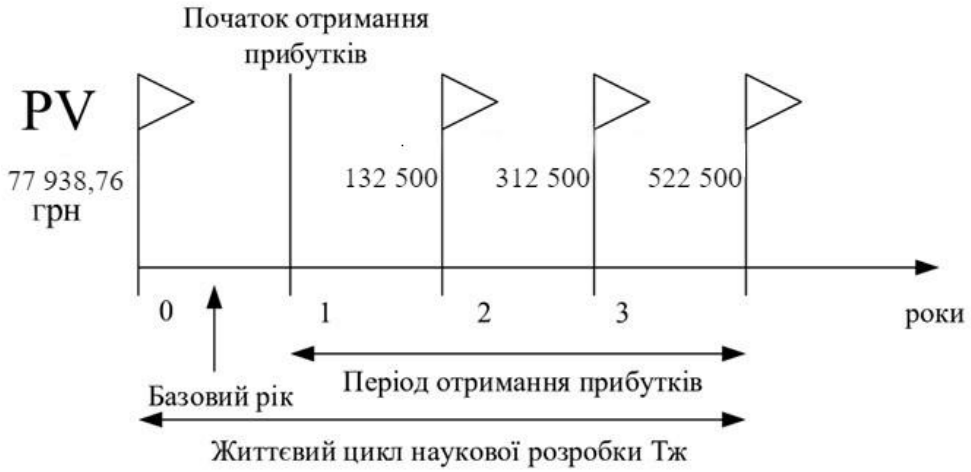


Рисунок 4.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн.

Приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^t}, \quad (4.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні - 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки „0”.

$$ПП = \frac{132\,500}{(1+0{,}1)^1} + \frac{312\,500}{(1+0{,}1)^2} + \frac{522\,500}{(1+0{,}1)^3} = 771\,280{,}99 \text{ (грн)}.$$

$$Е_{абс} = 771\,280{,}99 – 77\,938{,}76 = 693\,342{,}23 \text{ (грн)}.$$

Оскільки $Е_{абс} > 0$, результат від проведення наукових досліджень щодо розробки програмного продукту та їх впровадження принесе прибуток, тобто є доцільним, але це ще не свідчить про те, що інвестор буде зацікавлений у фінансуванні даної програми.

5-й крок. Розраховують відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (4.11)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{1 + \frac{693\,342{,}23}{77\,938{,}76}} - 1 = \sqrt[3]{7{,}95} - 1 = 1{,}08 \text{ або } 108{,}00\%.$$

Порівняємо $E_в$ з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть.

Спрогнозуємо величину $\tau_{мін.}$ У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f, \qquad (4.12)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; d = 0,14;

f – показник, що характеризує ризикованість вкладень; величина f = 0,3.

$$\tau = 0{,}14 + 0{,}3 = 0{,}44.$$

Припустимо, що за даних умов прибуток буде збільшуватись, то у інвестора є потенційна зацікавленість у фінансуванні даної наукової розробки.

6-й крок. Розраховують термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ за формулою:

$$T_{ок} = \frac{1}{E_в} \text{ [роки]}. \qquad (4.13)$$

$$T_{ok} = \frac{1}{1{,}08} = 0{,}93 \text{ (роки)}.$$

Оскільки термін окупності вкладених у реалізацію наукового проекту інвестицій менше трьох років ($T_{ок}$ < 3 років), то фінансування нової розробки є доцільним.

4.5 Висновок

В даному розділі було здійснено оцінювання комерційного потенціалу розробки автоматизованої інформаційної системи робочих місць для співробітників станції технічного обслуговування.

Проведено технологічний аудит з залученням трьох експертів. Аналіз експертних даних показав, що рівень комерційного потенціалу розробки вище середнього. Дослідження комерційного потенціалу розробки показав, що програмний продукт за своїми характеристиками випереджає аналогічні програмні продукти і є перспективною розробкою. Він має кращі функціональні показники, а тому є конкурентоспроможним товаром на ринку.

На основі зроблених підрахунків в економічній частині магістерської кваліфікаційної роботи досягнуті наступні результати:

- витрати на розробку та її впровадження складають 77 938,76 грн;
- абсолютний ефект розробки складає 693 342,23 грн;
- за здійсненими підрахунками було виявлено, що внутрішня норма дохідності була досягнута і складає 108%;
- термін окупності системи, що розробляється складає 0,93 років.

Отже, дана розробка є економічно вигідною та доцільною для вкладання в неї коштів інвесторів.

ВИСНОВКИ

В ході виконання магістерської кваліфікаційної роботи було розроблено усі модулі для інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування.

Основні результати, які були отримані в процесі вирішення поставлених завдань та становлять наукову новизну дослідження, полягають в наступному:

1. Подальшого розвитку набули моделі автоматизованого процесу призначення задач для співробітників СТО на базі генетичного та угорського алгоритмів, які на відміну від існуючих, враховують одночасно, складність задачі, час виконання задач та кваліфікацію робітників.

2. Розроблено інформаційну модель автоматизованої системи співробітників станції технічного обслуговування, яка на відмінну від існуючих, здійснює миттєву взаємодію між програмними модулями системи за рахунок розробленого API, яка забезпечує миттєве зчитування/запис інформації до бази даних.

Практична цінність магістерської роботи полягає у наступному:

1. Розроблено алгоритм автоматизованого призначення роботи для співробітників станції технічного обслуговування на базі генетичного та угорського алгоритмів;

2. Розроблено структуру базу даних для оптимального представлення усієї інформації предметної області;

3. Розроблено API з можливістю швидкого зчитування/запису інформації до бази даних та зручним обміном цими даними з клієнтською частиною;

4. Розроблено крос-платформну клієнтську частину зі зручним та зрозумілим інтерфейсом, для швидкої взаємодії користувача із системою та її базою даних в режимі реального часу.

Отже всі завдання, які поставлені в магістерській кваліфікаційній роботі виконано у повному обсязі. Мету роботи досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вирішення задачі про призначення в модульній інформаційній системі станції технічного обслуговування [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2019/paper/viewFile/8087/6759.
2. Система "1С: Підприємство" [Електронний ресурс]. – Режим доступу: http://pidruchniki.com/1538061063985/informatika/sistema_1s_pidpriyemstvo.
3. Сервіс пошуку та онлайн запису на автосервіс [Електронний ресурс]. – Режим доступу до ресурсу: https://autobooking.com/ua-ua.
4. Задача про призначення [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Задача_про_призначення.
5. Симплекс-метод [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Симплекс_метод.
6. Алгоритм аукціону [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Алгоритм_аукціону.
7. Угорський алгоритм [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Угорський_алгоритм.
8. Генетичний алгоритм [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Генетичний_алгоритм.
9. Встановлення та налаштування Node.js [Електронний ресурс]. – Режим доступу до ресурсу: https://internetdevels.ua/blog/node-js-installation-setting;
10. Express.js [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Express.js.
11. Бази даних та інформаційні системи. [Електронний ресурс]. – Режим доступу до ресурсу: http://oracle.com.edgesuite.net/timeline/java/.

12. Карпова В.П. Організація баз даних. Підручник. / Карпова В.П. – Санкт-Петербург: ПИТЕР, 2001. – 260с.
13. Лузанов П. PostgreSQL для початківців. / Лузанов П., Рогов Е., Левшин І. – Москва: Postgres Professional, 2017. – 118 с.
14. Уорслі Дж. PostgreSQL. Для професіоналів. / Уорслі Дж. Дрейк Дж. – Київ: ПИТЕР, 2003. – 498с.
15. Мартін Груббер. Розуміння SQL. / Мартін Груббер. – Санкт-Петербург: ПИТЕР, 1993. – 289с.
16. Гевін Кінг. Гібернація у дії. / Гевін Кінг, Крістіан Бауер. – США: Manning Publications Co., 2005. – 430с.
17. ORM-система Sequelize для Node.js [Електронний ресурс]. – Режим доступу до ресурсу: https://hackerx.ru/orm-sequelize-node-js/.https://www.zina.design/uk/dictionary/bootstrap/http://webstudio2u.net/ua/programming/120-jquery.htmlc[].
18. Шаблонізатор Dust [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Dust.
19. Bootstrap [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Bootstrap.
20. Козловський В. О. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт. / Козловський В. О. – Вінниця: ВНТУ, 2012. – 22 с.

Додаток А

Технічне завдання

Вінницький національний технічний університет
Факультет комп'ютерних систем і автоматики

ПОГОДЖЕНО
Директор ТОВ «СКАЙСОФТТЕК»
_____ Гуральник Б. О.
(підпис)
"__" _____ 2019 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри САКМІГ
___ д. т. н., проф. В.Б. Мокін
(підпис)
"__" _____ 2019

ТЕХНІЧНЕ ЗАВДАННЯ

на магістерську кваліфікаційну роботу

ІНФОРМАЦІЙНА АВТОМАТИЗОВАНА СИСТЕМА РОБОЧИХ МІСЦЬ ДЛЯ СПІВРОБІТНИКІВ СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ

08-53.МКР.004.02.000 ТЗ

Керівник магістерської
кваліфікаційної роботи
к.т.н., доц.
_____О.М. Козачко
(підпис)
"__" _____ 2019 р.
Розробив студент гр. ІСТ-18м
_____А.Ю. Дячук
(підпис)
"__" _____ 2019 р.

Вінниця 2019

1. Підстава для проведення робіт

Підставою для виконання роботи є наказ № __ по ВНТУ від «__» _____ 201_ р., та індивідуальне завдання на МКР, затверджене протоколом № __ засідання кафедри САКМІГ від «__» _____ 201_ р.

2. Джерела розробки

Серед джерел, які будуть використані в ході розробки магістерської кваліфікаційної роботи є: Державні стандарти України (ДСТУ) 3008-2015 «Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлювання», 2481-94 “Системи оброблення інформації. Інтелектуальні інформаційні технології. Терміни та визначення”, завдання на МКР, довідкова та технічна література.

3. Мета і призначення роботи.

Метою дослідження є пришвидшення та оптимізація робочого процесу на станціях технічного обслуговування за рахунок математичних моделей процесу призначення задач, які враховують складність задач та кваліфікацію робітників.

4. Вихідні дані для проведення робіт

1) Масив робітників СТО та кваліфікація, швидкість роботи та оплата.

2) Масив задач СТО та їх складність.

5. Методи дослідження

1) Угорський алгоритм вирішення задачі про призначення.

2) Генетичний алгоритм вирішення задачі про призначення.

6. Етапи роботи і терміни їх виконання

1) Огляд технічних та програмних засобів..……..__. – __

2) Розробка інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування……..__. – __

3) Реалізація інформаційної автоматизованої системи робочих місць для співробітників станції технічного обслуговування ……....__. – __

7. Очікувані результати та порядок реалізації

Розробка інформаційної моделі автоматизованої системи співробітників станції технічного обслуговування, яка на відмінну від існуючих, здійснює миттєву взаємодію між програмними модулями системи за рахунок розробленого API, яка забезпечує миттєве зчитування/запис інформації до бази даних.

8. Вимоги до розробленої документації

Пояснювальна записка оформлена у відповідності до вимог «Методичних вказівок до виконання та оформлення магістерських кваліфікаційних робіт для студентів спеціальності 126 – «Інформаційні системи та технології» денної форми навчання».

9. Порядок приймання роботи

Публічний захист…….__.__.201_ р.

Початок розробки «___» _____ 201_ р.

Граничні терміни виконання МКР «___» _____201_ р.

Розробив студент групи ІСТ-18м _____ Дячук А. Ю.

Додаток Б

Інструкція користувача

Переглянемо роботу сервісу на прикладі екземпляру робочого сценарію. У водія зламалась машина, тому він викликав евакуатор, доїхав до сервісу і менеджер починає реєструвати нового клієнта в системі (у випадку існування цього клієнта в системі, даний крок пропускається), заповнюючи поля: «Ім'я», «Прізвище», «Компанія», «Адреса», «Контактний номер», «Логін», «Email» та «Пароль». Вікно додавання нового клієнта відображене на рисунку Б.1

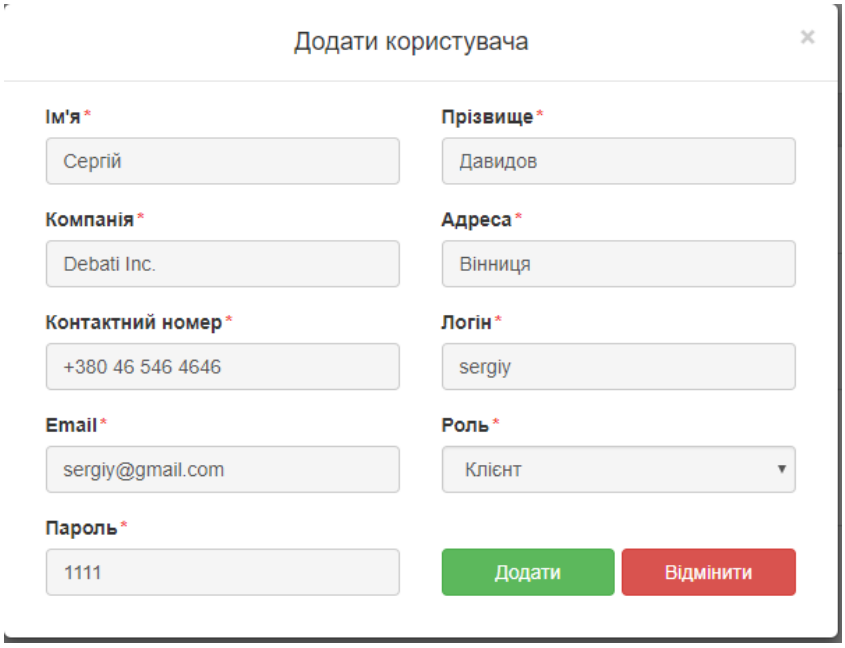


Рисунок Б.1 – Вікно додавання нового клієнта

Після підтвердження додавання замовлення клієнту, на його пошту, що була вказана при його реєстрації, буде надісланий лист-повідомлення про успішну реєстрацію та його логін і пароль для входу в особистий кабінет.

Після успішної реєстрації клієнта в системі, слід почати реєструвати замовлення, заповнюючи усі необхідні поля: «Тип транспорту», «Марка транспорту», «Модель транспорту», «Клієнт», «Назва замовлення», «Час початку» та «Орієнтовний час виконання». При необхідності включення додаткової інформації для реєстрації замовлення можна заповнити необов'язкові поля «Опис замовлення» і «Коментар». Вікно реєстрації нового замовлення відображене на рисунку Б.2.

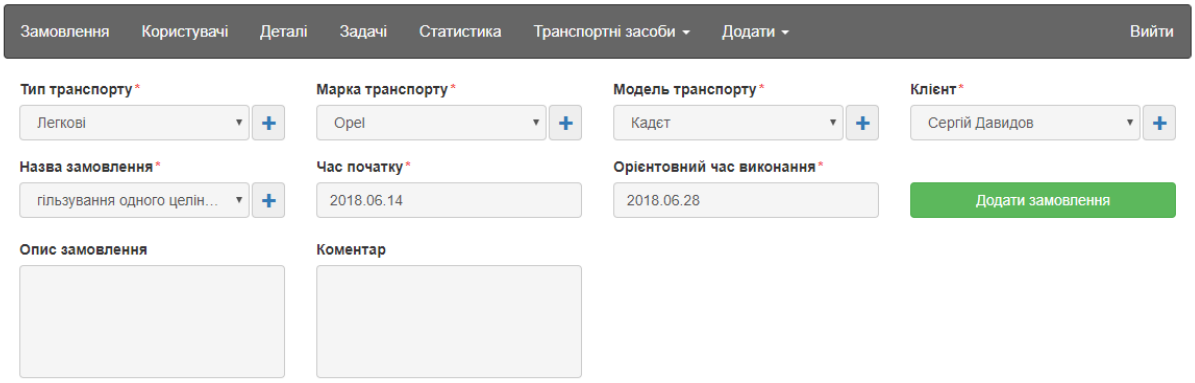


Рисунок Б.2 – Вікно реєстрації нового замовлення

Після підтвердження додавання нового замовлення користувачеві на пошту буде надісланий лист-сповіщення про успішну реєстрацію нового замовлення. Тепер користувач може зайти в систему, ввівши свій логін та пароль і відслідковувати інформацію про процес виконання свого замовлення, статус в якому воно знаходиться, та статус його задач. Результат додавання нового замовлення відображений на рисунку Б.3.

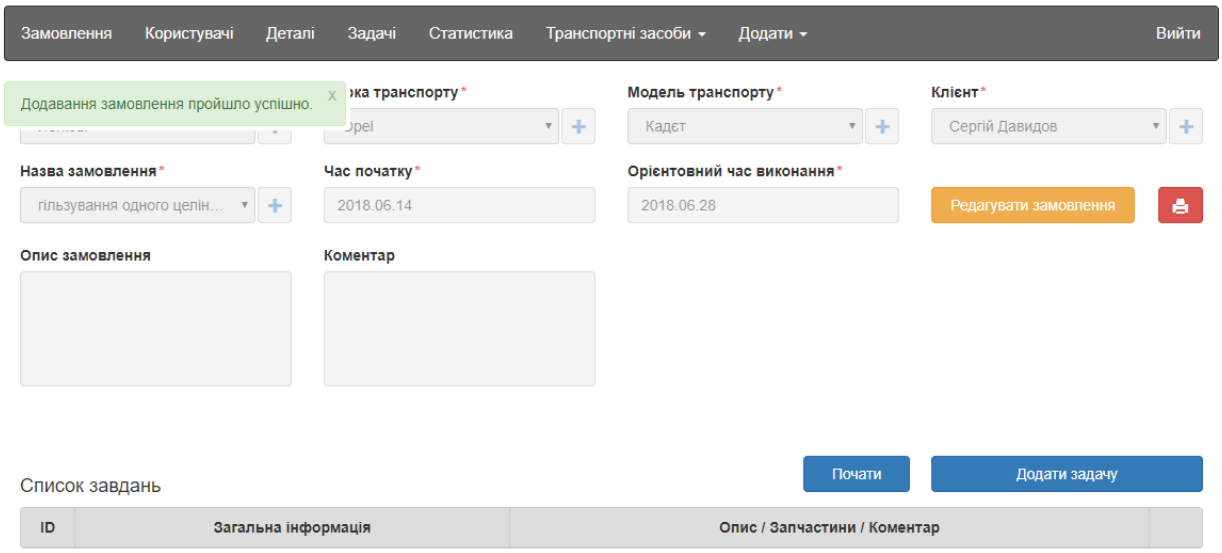


Рисунок Б.3 – Результат додавання нового замовлення

Вхід в систему клієнта під своїм логіном та паролем відображений на рисунку Б.4.

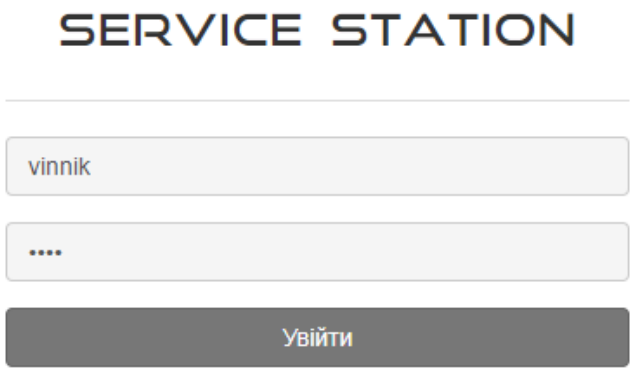


Рисунок Б.4 – Вхід в систему клієнта під своїм логіном та паролем

Головна сторінка клієнта відображена на рисунку Б.5.

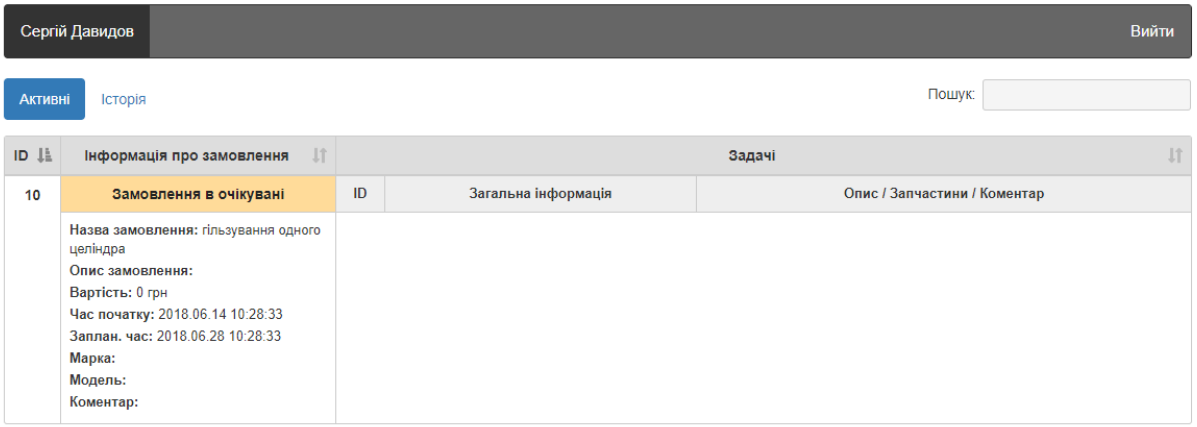


Рисунок Б.5 – Головна сторінка клієнта

Наступним кроком робочого процесу в інформаційній автоматизованій системі робочих місць для співробітників станції технічного обслуговування буде створення задач для замовлення. В рамках роботи системи, замовлення є загальним поняттям, що описує факт необхідності працювати із автомобілем клієнта. Проте, за для пришвидшення та спрощення виконання замовлення буде реалізоване розділення основного замовлення та підзадачі та розподілення їх між інженерами шляхом доручення кожному з них свої окремих задач.

Отож, створення задачі являє собою процес заповнення менеджером або адміністратором наступних полів: «Назва задачі», «Виконавець», «Доручити задачу», «Час початку», «Час виконання», «Вартість» та «Деталі». У разі необхідності автоматичного доручення задачі на основі алгоритму, поле «Виконавець» можна залишити пустим. Вікно додавання нової задачі відображене на рисунку Б.6.

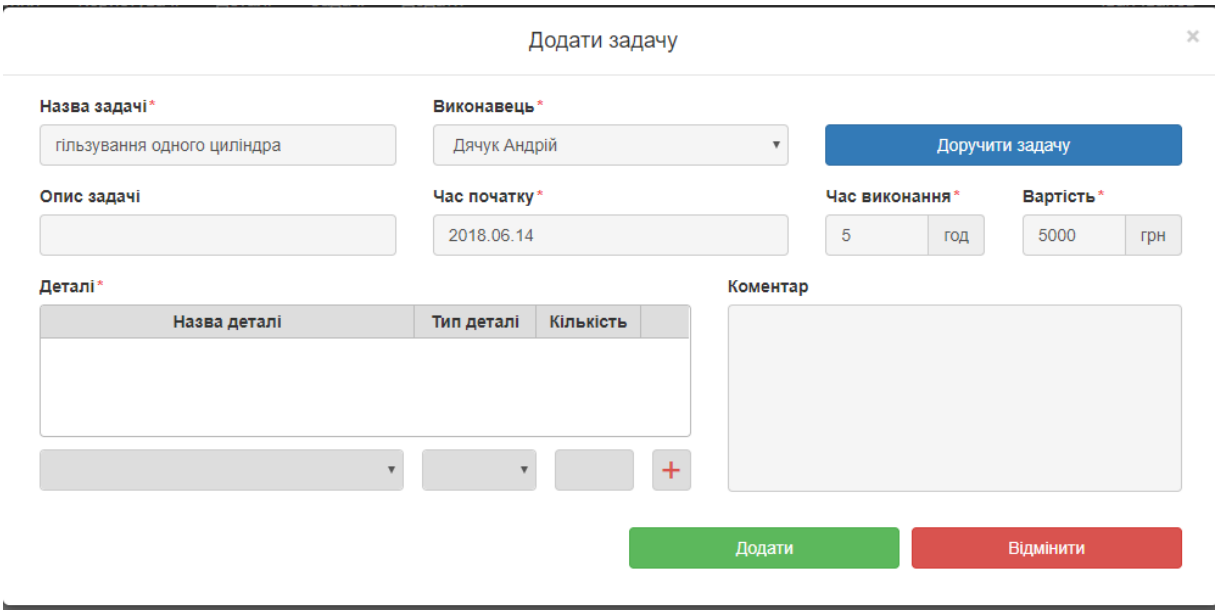


Рисунок Б.6 – Вікно додавання нової задачі

Також при умові існування замовлення в інформаційній автоматизованій системі робочих місць для співробітників станції технічного обслуговування менеджер або адміністратор має змогу виконати автоматичне формування та друк квитанції на вимогу клієнта.

При необхідності вказати додаткову інформацію для реєстрації та виконання задачі, менеджер чи адміністратор може заповнити необов'язкові поля «Опис задачі» та «Коментар» на вимогу клієнта. Якщо для виконання конкретної задачі потрібно використати додаткові деталі, менеджер або адміністратор повинен заповнити поля «Назва», «Тип» та «Кількість» для кожної деталі.

Після успішного сформування та підтвердження створення кожної задачі для замовлення менеджер має змогу на вимогу клієнта ініціювати автоматичне формування квитанції та її друк. Квитанція містить в собі загальну інформацію про замовлення, список задач які необхідно виконати, перелік деталей, їх ціну та загальну суму, яку необхідно заплатити клієнту за надані послуги. Квитанція замовлення відображена на рисунку Б.7.

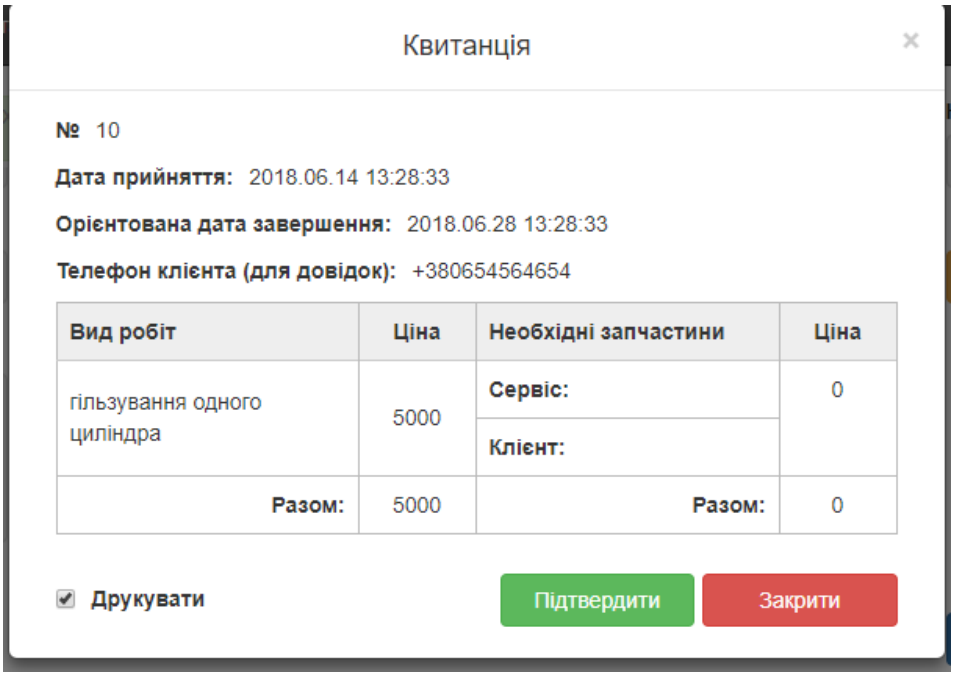


Рисунок Б.7 – Квитанція замовлення

Після погодження та видачі квитанції клієнту менеджер може ініціювати початок виконання замовлення, натиснувши кнопку «Почати» в результаті чого усі працівники, яких раніше зареєстрував адміністратор, і за якими тепер закріплені задачі отримують завдання на своїх сторінках (автоматизованих робочих місцях), в які заходять ввівши свої логіни та паролі. Замовлення на головній сторінці менеджера відображене на рисунку Б.8.

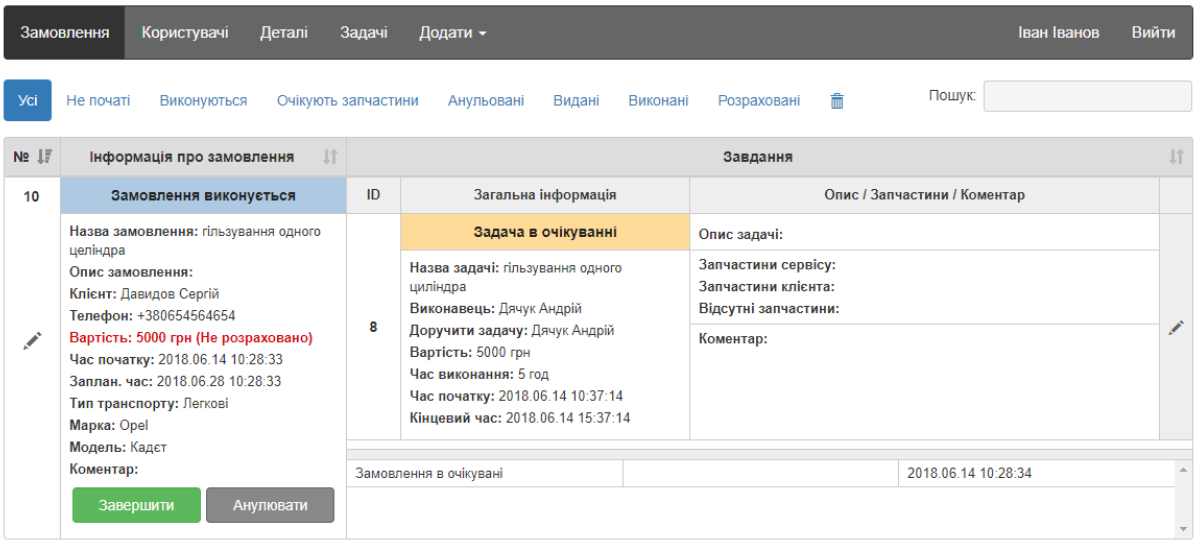


Рисунок Б.8 – Замовлення на головній сторінці менеджера

Наступним кроком є робота з задачами інженерів. Головне вікно інженера відображене на рисунку Б.9.

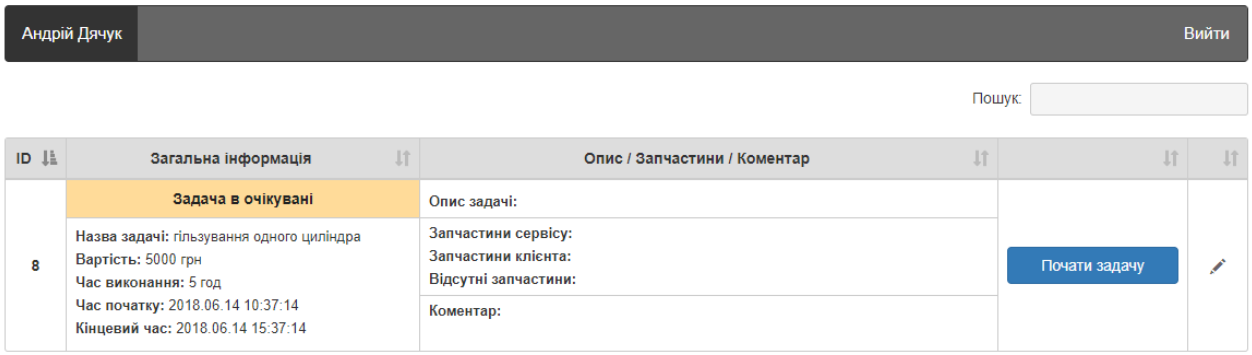


Рисунок Б.9 – Головне вікно інженера

Виявивши нову задачу, інженер може провести процес редагування замовлення, якщо якісь поля на його думку підлягають зміні: «Доручити задачу», «Опис задачі», «Час початку», «Час виконання», «Вартість» та «Деталі». Після редагування, інженер має змогу розпочати роботу над задачею, натиснувши на кнопку «Почати задачу». Початі задачі інженера відображені на рисунку Б.10.

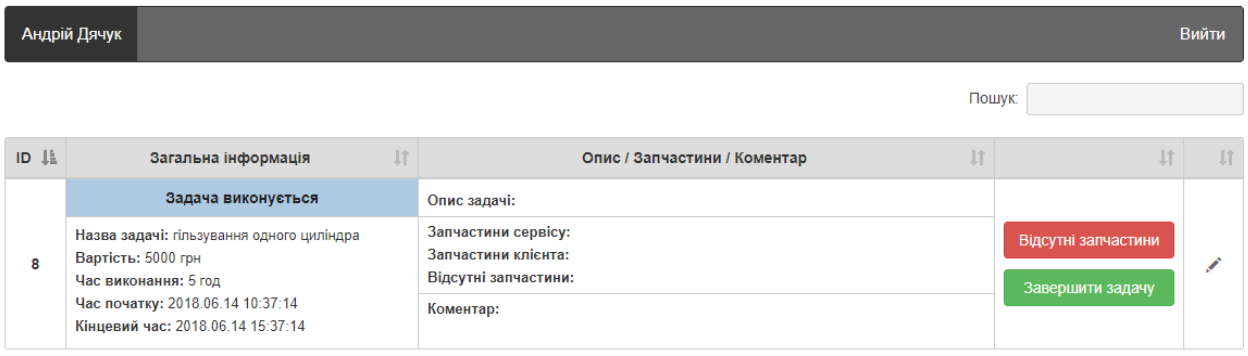


Рисунок Б.10 – Початі задачі інженера

Якщо в процесі виконання задачі, інженер виявив нехватку деталей, він має змогу натиснути кнопку «Відсутні запчастини», заповнивши обов’язкове поле «Коментар» із зазначенням чого саме бракує для виконання задачі. В цей час в менеджера в таблиці замовлень статус задачі змінюється на «Задачу зупинено».

В такому випадку, менеджер зв’язується з клієнтом для повідомлення йому інформації про відсутність необхідних запчастин.

Якщо клієнт погоджується на купівлю запчастин в сервісі, то менеджер передоручає дану задачу завідувачу складом.

Наступним етам життєвого циклу задачі є поява її на власній сторінці завідувача складом, який може або у випадку відсутності деталі на складі повідомити про це менеджера, або знайшовши необхідні запчастини передоручити задачу назад відповідальному за неї інженеру.

Після цього, у власному кабінеті інженера та менеджера статус задачі знову повернеться до «Задача в очікувані». В разі виконання задачі інженер натисне на кнопку «Завершити задачу».

А після того, як всі задачі замовлення будуть завершені замовлення змінить автоматично свій статус на «Замовлення виконано» і клієнту на пошту прийде лист-сповіщення про успішне виконання замовлення, а на його сторінці, де він може відслідковувати увесь життєвий цикл свого замовлення та задач, замовлення змінить свій статус на «Замовлення виконано». Завершені задачі та замовлення на головній сторінці інженера, менеджера та клієнта відображені на рисунках Б.11 – Б.13.

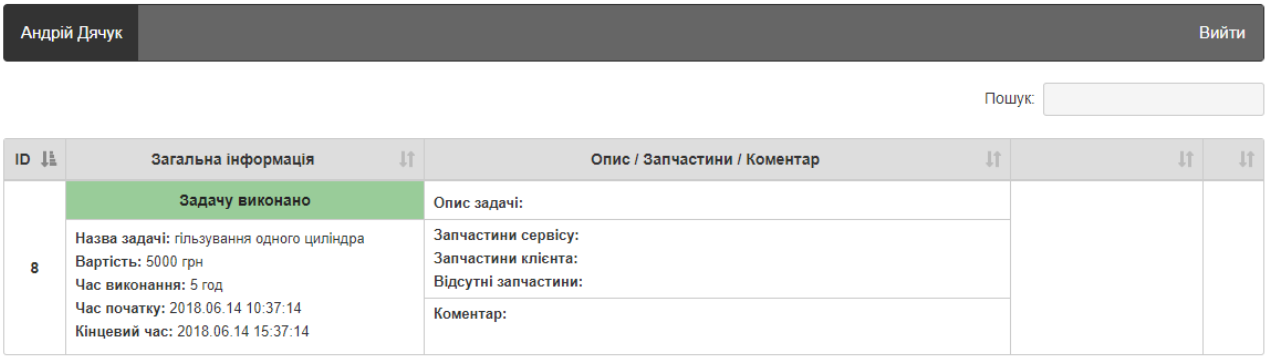


Рисунок Б.11 – Завершені задачі на головній сторінці інженера

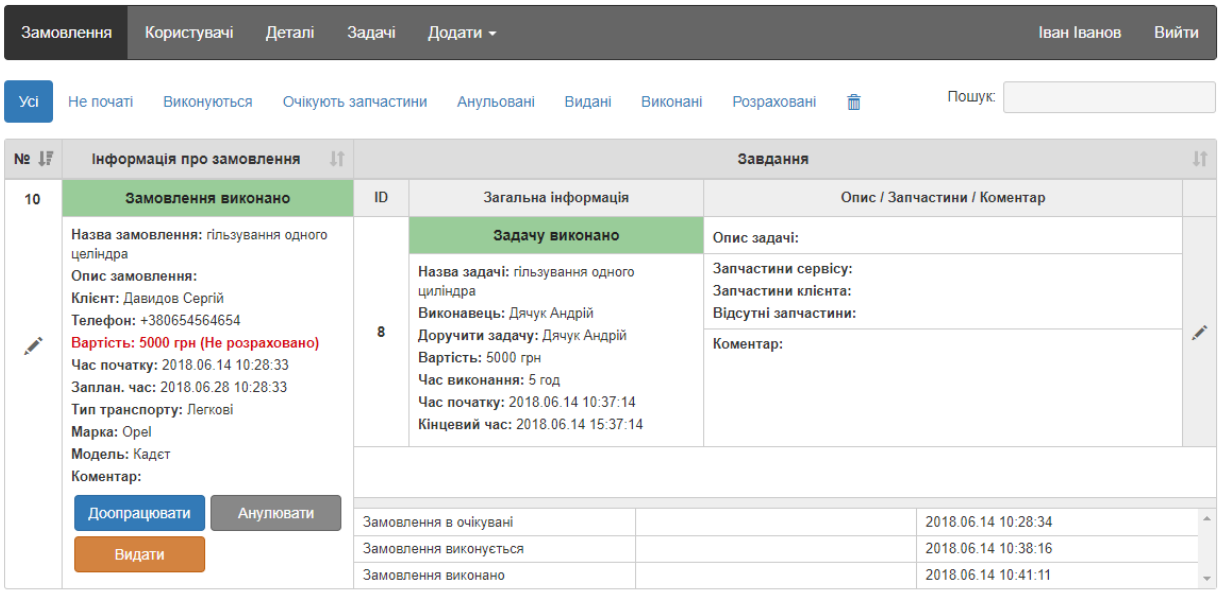


Рисунок Б.12 – Завершені задачі та замовлення на головній сторінці менеджера

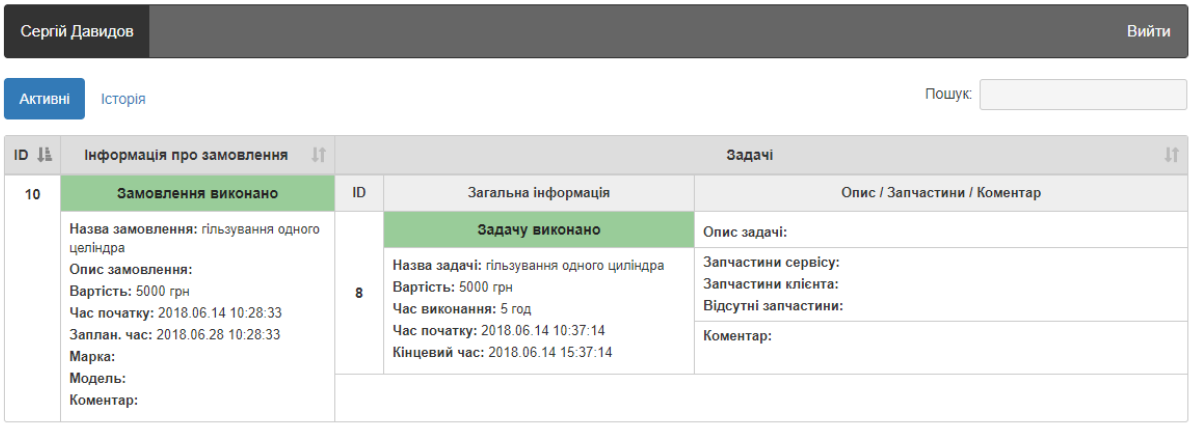


Рисунок Б.13 – Завершені задачі та замовлення на головній сторінці клієнта

Після цього клієнт повинен розплатитись за виконання замовлення в бухгалтера, в разі чого бухгалтер зможе натиснути кнопку «Розрахувати». Після факту успішного розрахунку клієнта, менеджер або адміністратор має натиснути кнопку «Видати». Видане замовлення на головній сторінці менеджера відображене на рисунку Б.14

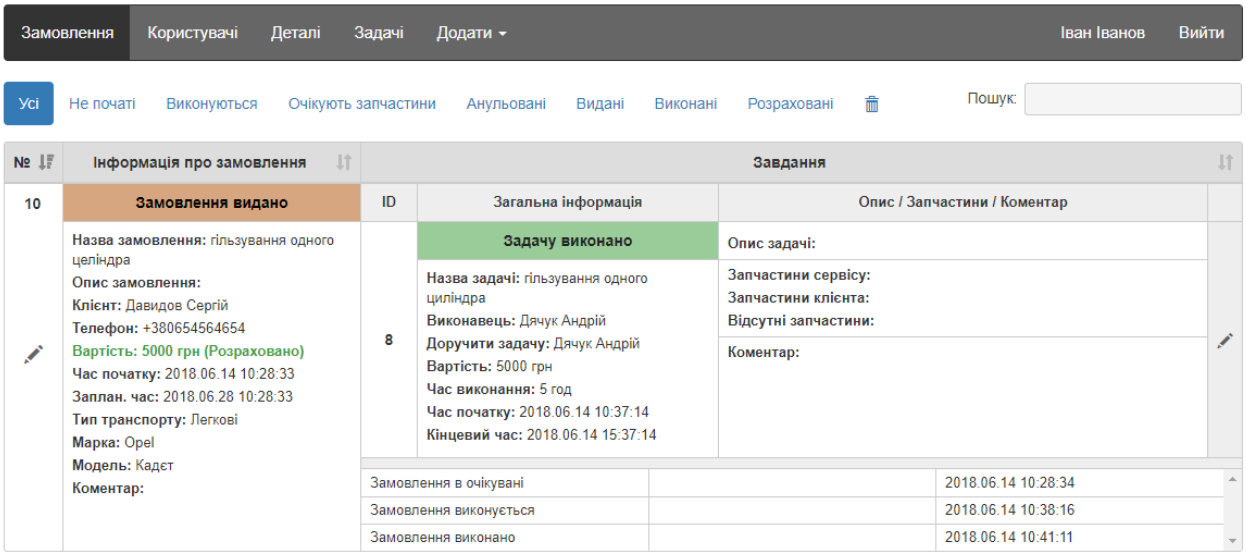


Рисунок Б.14 – Видане замовлення на головній сторінці менеджера

Замовлення змінить свій статус на «Замовлення видано» в особистих кабінетах адміністратора, менеджера та клієнта, а останній зможе забрати свій відремонтований транспортний засіб.

Додаток В

Лістинг програми

1. Код контролера моделі «Менеджер»

```
"use strict";

const User = require('../../models/User');
const Task = require('../../models/Task');
const Request = require('../../models/Request');
const TaskType = require('../../models/TaskType');
const RequestHistory = require('../../models/RequestHistory');
const TransportType = require('../../models/TransportType');
const TransportMarkk = require('../../models/TransportMarkk');
const TransportModel = require('../../models/TransportModel');
const Models = require('../../models/TaskDetail/relations');
const express = require('express');
const router = express.Router();

const roles = require('../../constants/roles');
const validation = require('../../middleware/validation');
const requestsFactory = require('../../helpers/requestsFactory');
const countRequestHistory = require('../../helpers/countRequestHistory');
const countMoney = require('../../helpers/countMoney');
const countTasks = require('../../helpers/countTasks');
const nodemailer = require('../../helpers/nodemailer');
const countEndTime = require('../../helpers/countEndTime');
const status = require('../../constants/status');

router.get('/users', (req, res) => {
    User
        .getAllUsers()
        .then(users => {
            res.render('roles/admin_moderator/users', {
                users: users,
                typeUser: req.session.passport.user.userTypeID
            });
        })
        .catch(error => {
            console.warn(error);
            res.render('roles/admin_moderator/users');
        });
});

router.post('/create-user', validation.createAndUpdateUser('create'), (req, res) => {
    if (req.baseUrl === '/moderator') {
        req.body.userTypeID = roles.CUSTOMER;
    }

    User
        .createUser(req.body)
        .then(() => {
            nodemailer('create-user', req.body);
            req.flash('success_alert', true);
            req.flash('success_msg', 'Додавання користувача пройшло успішно.');
            res.redirect('back');
        })
        .catch(error => {
            console.warn(error);
            req.flash('error_alert', true);
            req.flash('error_msg', {msg: 'Виникла помилка при додаванні
користувача.'});
            res.redirect('back');
```

```
        });
});

router.put('/update-user/:id', validation.createAndUpdateUser(), (req, res) => {
    User
        .updateUser(req.params.id, req.body)
        .then(() => {
            res.status(200).send();
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({
                errors: errors
            });
        });
});

router.delete('/delete-user/:id', (req, res) => {
    User
        .deleteUser(req.params.id)
        .then(() => {
            res.status(200).send({result: 'ok'});
        })
        .catch(error => {
            console.warn(error);
            res.status(400).send({errors: errors});
        });
});

router.get('/requests/:status', (req, res) => {
    let findBy, hold;

    if (req.params.status === 'all' || req.params.status === 'hold') {
        findBy = {
            status: {
                $between: [1, 5]
            },
            hadDeleted: false
        }
    }
    else if (req.params.status === 'pending') {
        findBy = {
            status: status.PENDING
        }
    }
    else if (req.params.status === 'processing') {
        findBy = {
            status: status.PROCESSING
        }
    }
    else if (req.params.status === 'done') {
        findBy = {
            status: status.DONE
        }
    }
    else if (req.params.status === 'canceled') {
        findBy = {
            status: status.CANCELED
        }
    }
    else if (req.params.status === 'give-out') {
        findBy = {
            giveOut: true
        }
    }
    else if (req.params.status === 'payed') {
        findBy = {
            payed: true,
            hadDeleted: false
        }
```

```
    }
    else if (req.params.status === 'not-payed') {
        findBy = {
            payed: false,
            hadDeleted: false
        }
    }
    else if (req.params.status === 'trash') {
        findBy = {
            hadDeleted: true
        }
    }

    Request
        .getAllRequests(findBy)
        .then(requests => {
            RequestHistory
                .getAllRequestHistory()
                .then(requestsHistory => {
                    method(req.params.status)
                        .then(tasks => {
                            User
                                .getAllUsers()
                                .then(users => {
                                    if (req.params.status === 'hold') {
                                        hold = true;
                                    }
                                    res.render('roles/admin_moderator/requests', {
                                        assignedExecutorUsers: users,
                                        requests: requestsFactory(requests, tasks,
hold, requestsHistory),
                                        typeUser:
req.session.passport.user.userTypeID,
                                        isSetPayed: true //for add class set_payed to
print modal
                                    });
                                })
                                .catch(error => {
                                    console.warn(error);
                                    res.render('roles/admin_moderator/requests');
                                });
                        });
                })
                .catch(error => {
                    console.warn(error);
                    res.render('roles/admin_moderator/requests');
                });
        })
        .catch(error => {
            console.warn(error);
            res.render('roles/admin_moderator/requests');
        });
});

router.get('/create-request', (req, res) => {
    User
        .getCustomerUsers()
        .then(usersCustomers => {
            User
                .getAllUsers()
                .then(users => {
                    TransportType
                        .getAllTransportType()
                        .then(types => {

                            res.render('roles/admin_moderator/create_request', {
                                assignedExecutorUsers: users,
                                customers: usersCustomers,
                                typeUser: req.session.passport.user.userTypeID,
                                types: types
```

```
                            })
                        })
                        .catch(err => {
                            console.warn(err);
                            res.render('roles/admin_moderator/create_request');
                        })
                })
                .catch(err => {
                    console.warn(err);
                    res.render('roles/admin_moderator/create_request');
                })
        })
});

router.post('/create-request', validation.createAndUpdateRequest(), (req, res) => {
    req.body.createdBy = req.session.passport.user.id;

    Request
        .createRequest(req.body)
        .then((result) => {
            User
                .getUserById(req.body.customerID)
                .then(customer => {
                    nodemailer('create-request', customer);
                    res.status(200).send({
                        result: result,
                        customer: customer
                    });
                })
                .catch(errors => {
                    res.status(400).send({errors: errors});
                });
        })
        .catch(errors => {
            res.status(400).send({errors: errors});
        });
});

router.put('/update-request/:id', validation.createAndUpdateRequest(), (req, res) => {
    Request
        .updateRequest(req.params.id, req.body)
        .then((result) => {
            Request
                .getRequestById(req.params.id)
                .then(request => {
                    User
                        .getUserById(request[0].dataValues.customerID)
                        .then(customer => {
                            res.status(200).send({request: request[0].dataValues,
result: result, customer: customer});
                        })
                        .catch(errors => {
                            res.status(400).send({errors: errors});
                        });
                })
                .catch(errors => {
                    res.status(400).send({errors: errors});
                });
        })
        .catch(errors => {
            res.status(400).send({errors: errors});
        });
});

router.delete('/delete-request/:id', (req, res) => {
    if (req.body.hadDeleted === '') {
        Task
            .changeStatusByRequestID(Number(req.params.id), status.CANCELED)
            .then(() => {
                Request
```

```
                    .changeStatus(Number(req.params.id), {status: status.CANCELED})
                    .then(() => {
                        Request
                            .updateRequest(Number(req.params.id), {hadDeleted: true})
                            .then(() => {
                                res.status(200).send();
                            })
                            .catch(error => {
                                console.warn(error);
                                res.status(400).send({errors: errors});
                            });
                    })
                    .catch(error => {
                        console.warn(error);
                        res.status(400).send({errors: errors});
                    });
            })
            .catch(error => {
                console.warn(error);
                res.status(400).send({errors: errors});
            });
    } else {
        Task
            .destroy({
                where: {
                    request_id: Number(req.params.id)
                }
            })
            .then(() => {
                Request
                    .deleteRequest(req.params.id)
                    .then(() => {
                        res.status(200).send();
                    })
                    .catch(error => {
                        console.warn(error);
                        res.status(400).send({errors: errors});
                    });
            })
            .catch(error => {
                console.warn(error);
                res.status(400).send({errors: errors});
            });

    }
});

router.put('/change-request-status/:id', (req, res) => {

    Request
        .changeStatus(req.params.id, req.body)
        .then(() => {
        Task
            .getAllTasksOfRequest(req.params.id)
            .then(tasks => {

                Task
                    .updateAllTasksStatusOfRequest(tasks, req.body.status)
                    .then(() => {

                        Request
                            .getRequestById(req.params.id)
                            .then(request => {

                                if (+req.body.status === status.DONE) {

                                    Request
                                        .getRequestById(req.params.id)
                                        .then((result) => {
```

```
                                        nodemailer('done-request',
result[0].user.dataValues);
                                    })
                                    .catch(error => {
                                        console.warn(error);
                                    });
                            }

                            if (req.body.hadDeleted){
                                var params = {
                                    hadDeleted: false
                                };

                                Request
                                    .updateRequest(req.params.id, params)
                                    .then()
                                    .catch(errors => {
                                        console.warn(errors);
                                        res.status(400).send({errors: errors});
                                    });
                            }
                            res.status(200).send({
                                status: req.body.status,
                                requestID: req.params.id,
                                request: request
                            });
                        })
                        .catch(errors => {
                            console.warn(errors);
                            res.status(400).send({errors: errors});
                        });
                })
                .catch(errors => {
                    console.warn(errors);
                    res.status(400).send({errors: errors});
                });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.put('/set-payed/:id', (req, res) => {
    Request
        .updateRequest(req.params.id, req.body)
        .then(() => {
            Request
                .getRequestById(req.params.id)
                .then(request => {
                    User
                        .getAllAdmins()
                        .then(admins => {
                            if (req.body.payed) {
                                nodemailer('set-payed', admins,
request[0].dataValues);
                            } else {
                                if (req.body.giveOut) {
                                    User
                                        .getAllBookkeepers()
                                        .then(bookkeepers => {
                                            bookkeepers.forEach(item => {
                                                admins.push(item);
                                            });
```

```
                                        nodemailer('give-out', admins,
request[0].dataValues);
                                    })
                                    .catch(errors => {
                                        console.warn(errors);
                                        res.status(400).send({errors: errors});
                                    });
                            }
                        }
                        res.status(200).send({
                            request: request
                        });
                    })
                    .catch(errors => {
                        console.warn(errors);
                        res.status(400).send({errors: errors});
                    });

            })
            .catch(errors => {
                console.warn(errors);
                res.status(400).send({errors: errors});
            });
    })
    .catch(errors => {
        console.warn(errors);
        res.status(400).send({errors: errors});
    });
});

router.get('/get-request-check/:id', (req, res) => {
    Request
        .getRequestById(req.params.id)
        .then(request => {
            User
                .getUserById(request[0].dataValues.customerID)
                .then(customer => {
                    Task
                        .getAllTasksOfRequest(request[0].dataValues.id)
                        .then(tasks => {
                            res.status(200).send({
                                request: request,
                                customerPhone: customer.dataValues.userPhone,
                                tasks: tasks
                            });
                        })

                });
    })
    .catch(errors => {
        console.warn(errors);
        res.status(400).send({errors: errors});
    });
});

router.post('/get-task-types', (req, res) => {

    TaskType
        .getTaskTypesByCarAttributes(req.body.typeOfCar, req.body.carMarkk,
req.body.carModel)
        .then(taskTypes => {
            Models.Detail
                .getDetailsByCarAttributes(req.body.typeOfCar, req.body.carMarkk,
req.body.carModel)
                .then(detailTypes => {
                    res.status(200).send({
                        detailTypes: detailTypes,
                        taskTypes: taskTypes
                    });
                })
```

```
                .catch(errors => {
                    console.warn(errors);
                    res.status(400).send({errors: errors});
                });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.post('/create-task', validation.createAndUpdateTask(), (req, res) => {

    req.body.endTime = countEndTime(req.body.startTime, +req.body.estimationTime);

    Request
        .getRequestById(req.body.requestID)
        .then(request => {
            var newCost = +request[0].dataValues.cost + +req.body.cost;
            Request
                .updateRequest(req.body.requestID, {cost: newCost})
                .then(() => {
                    var search = {
                        typeName: req.body.name,
                        typeOfCar: request[0].dataValues.transportTypeID,
                        carMarkk: request[0].dataValues.transportMarkkID,
                        carModel: request[0].dataValues.transportModelID,
                        cost: req.body.cost,
                        estimationTime: req.body.estimationTime,
                        planedExecutorID: req.body.planedExecutorID
                    };

                    if (!req.body.typeID) {
                        search.typeID = true;
                    }
                    TaskType
                        .createTaskType(search)
                        .then(result => {
                            if (!req.body.typeID) {
                                req.body.typeID = result.taskTypes[0].dataValues.id;
                            }
                            Task
                                .createTask(req.body)
                                .then(task => {
                                    Models.TaskDetail
                                        .createTaskDetail(task.id,
JSON.parse(req.body.detail))
                                        .then(() => {
                                            Models.TaskDetail
                                                .getTaskDetail(task.id)
                                                .then(taskDetail => {
                                                    res.status(200).send({
                                                        result: task,
                                                        taskDetail: taskDetail
                                                    });
                                                })
                                                .catch(errors => {
                                                    console.warn(errors);
                                                    res.status(400).send({errors:
errors});
                                                });
                                        })
                                        .catch(errors => {
                                            console.warn(errors);
                                            res.status(400).send({errors: errors});
                                        });
                                })
                                .catch(errors => {
                                    console.warn(errors);
                                    res.status(400).send({errors: errors});
```

```
                                });
                            })
                            .catch(errors => {
                                console.warn(errors);
                                res.status(400).send({errors: errors});
                            });
                    })
                    .catch(errors => {
                        console.warn(errors);
                        res.status(400).send({errors: errors});
                    });
            })
            .catch(errors => {
                console.warn(errors);
                res.status(400).send({errors: errors});
            });
});


router.delete('/delete-task/:id', (req, res) => {
    Task
        .destroy({
            where: {
                id: Number(req.params.id)
            }
        })
        .then(() => {
            Request
                .getRequestById(req.body.requestID)
                .then(request => {
                    var newCost = +request[0].dataValues.cost - +req.body.taskOldCost;

                    Request
                        .updateRequest(req.body.requestID, {cost: newCost})
                        .then(() => {
                            res.status(200).send({
                                request: request,
                                id: req.params.id,
                                requestID: req.body.requestID,
                                newCost: newCost
                            });
                        })
                        .catch(errors => {
                            console.warn(errors);
                            res.status(400).send({errors: errors});
                        });
                })
                .catch(errors => {
                    console.warn(errors);
                    res.status(400).send({errors: errors});
                });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.put('/cancel-task/:id', (req, res) => {
    Task
        .updateTask(req.params.id, req.body)
        .then(() => {
            Task
                .getTaskById(req.params.id)
                .then(task => {
                    res.status(200).send({
                        task: task
                    })
                })
                .catch(errors => {
```

```
                    console.warn(errors);
                    res.status(400).send({errors: errors});
                });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.get('/chart', (req, res) => {
    res.render('roles/admin_moderator/chart', {
        typeUser: req.session.passport.user.userTypeID
    });
});

router.post('/chart/requests', (req, res) => {
    RequestHistory
        .getChartRequestHistory(req.body)
        .then(requestHistory => {
            res.status(200).send({
                data: countRequestHistory(req.body, requestHistory)
            });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.post('/chart/finances', (req, res) => {
    Request
        .getAllRequestsForChart(req.body)
        .then(requests => {
            res.status(200).send({
                data: countMoney(req.body, requests)
            });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.post('/chart/tasks', (req, res) => {
    User
        .getAllExecutors()
        .then(users => {
            Task
                .getAllTasksForChart(req.body)
                .then(tasks => {
                    res.status(200).send({
                        data: countTasks(users, tasks)
                    });
                })
                .catch(errors => {
                    console.warn(errors);
                    res.status(400).send({errors: errors});
                });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.post('/get-task-prise/:id', (req, res) => {
    TaskType
        .getTaskTypeByID(req.params.id)
        .then(taskType => {
```

```
            res.status(200).send({
                taskType: taskType
            });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

function method(isHold) {
    return new Promise(function (resolve, reject) {

        if (isHold === 'hold') {
            Task
                .getAllHoldTasks()
                .then(result => {
                    resolve(result);
                })
        } else {
            Task
                .getAllTasks()
                .then(result => {
                    resolve(result);
                })
        }
    })
}

router.get('/task-type', (req, res) => {
    TaskType
        .getAllTaskType()
        .then(taskType => {
            User
                .getExecutorUsers()
                .then(users => {
                    TransportMarkk
                        .getAllTransportMarkks()
                        .then(transportMarkk => {
                            TransportModel
                                .getAllTransportModel()
                                .then(transportModel => {
                                    res.render('roles/admin_moderator/task_type', {
                                        users: users,
                                        taskType: taskType,
                                        typeUser:
req.session.passport.user.userTypeID,
                                        transportMarkk: transportMarkk,
                                        transportModel: transportModel
                                    });
                                })
                                .catch(error => {
                                    console.warn(error);
                                    res.render('roles/admin_moderator/requests/all');
                                });
                        })
                        .catch(error => {
                            console.warn(error);
                            res.render('roles/admin_moderator/requests/all');
                        });
                })
                .catch(error => {
                    console.warn(error);
                    res.render('roles/admin_moderator/task_type');
                });
        })
        .catch(error => {
            console.warn(error);
            res.render('roles/admin_moderator/task_type');
        });
```

```
});

router.post('/request-type', (req, res) => {
    Request
        .getRequestsWithoutCondition()
        .then(requestTypes  => {
            res.status(200).send({
                requestTypes: requestTypes
            });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });

});

router.post('/create-task-type', validation.createAndUpdateTaskType('create'), (req,
res) => {
    req.body.typeID = true;

    TaskType
        .createTaskType(req.body)
        .then((result) => {
            if (result.hasResult) {
                req.flash('success_alert', true);
                req.flash('success_msg', 'Додавання задачі пройшло успішно.');
                res.redirect(req.baseUrl + '/task-type');
            }
            else {
                var msg = 'Задача "' + req.body.typeName + '" вже існує. Скористайтеся
пошуком.';

                req.flash('error_alert', true);
                req.flash('error_msg', {msg: msg});
                res.redirect(req.baseUrl + '/task-type');
            }
        })
        .catch(error => {
            console.warn(error);
            req.flash('error_alert', true);
            req.flash('error_msg', {msg: 'Виникла помилка при додаванні задачі.'});
            res.redirect(req.baseUrl + '/task-type');
        });
});

router.put('/update-task-type/:id', validation.createAndUpdateTaskType(), (req, res)
=> {
    TaskType
        .updateTaskType(req.params.id, req.body)
        .then((taskType) => {
            User
                .getUserById(req.body.planedExecutorID)
                .then(user => {
                    res.status(200).send({
                        user: user,
                        taskType: taskType[0]
                    });
                })
                .catch(errors => {
                    console.warn(errors);
                    res.status(400).send({errors: errors});
                });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});
```

```
router.delete('/delete-task-type/:id', (req, res) => {
    TaskType
        .deleteTaskType(req.params.id)
        .then(() => {
            res.status(200).send();
        })
        .catch(error => {
            console.warn(error);
            res.status(400).send({errors: errors});
        });
});

router.put('/start-request/:id', (req,res) => {

    Request
        .changeStatus(req.params.id, {status: status.PROCESSING})
        .then(() => {
            res.status(200).send({
                isBeginButton: true
            })
        })
        .catch(error => {
            console.warn(error);
            res.status(400).send({errors: errors});
        });
});

/* ============== TRANSPORT TYPE ============== */
router.get('/transport-type', (req, res) => {
    TransportType
        .getAllTransportType()
        .then(transportTypes => {
            res.render('roles/admin_moderator/transport_type', {
                transportTypes: transportTypes,
                typeUser: req.session.passport.user.userTypeID
            });
        })
        .catch(error => {
            console.warn(error);
            res.render('roles/admin_moderator/requests/all');
        });
});

router.post('/create-transport-type',
validation.createAndUpdateTransportType('create'), (req, res) => {
    TransportType
        .createTransportType(req.body)
        .then((result) => {
            if (result.hasResult) {
                res.status(200).send({
                    transportType: result.transportType
                });
                // req.flash('success_alert', true);
                // req.flash('success_msg', 'Додавання типу транспорту пройшло
успішно.');
                // res.redirect('back');
            }
            else {
                var msg = 'Тип транспорту "' + req.body.transportTypeName + '" вже
існує. Скористайтеся пошуком.';

                req.flash('error_alert', true);
                req.flash('error_msg', {msg: msg});
                res.redirect('back');
            }
        })
        .catch(error => {
            console.warn(error);
            req.flash('error_alert', true);
            req.flash('error_msg', {msg: 'Виникла помилка при додаванні типу
```

```
транспорту.'});
            res.redirect('back');
        });
});

router.put('/update-transport-type/:id', validation.createAndUpdateTransportType(),
(req, res) => {
    TransportType
        .updateTransportType(req.params.id, req.body)
        .then(() => {
            res.status(200).send();
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.delete('/delete-transport-type/:id', (req, res) => {
    TransportType
        .deleteTransportType(req.params.id)
        .then(() => {
            res.status(200).send();
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

/* ============== TRANSPORT MARKK ============== */
router.get('/transport-markk', (req, res) => {
    TransportMarkk
        .getAllTransportMarkks()
        .then(markks => {
            TransportType
                .getAllTransportType()
                .then(types => {
                    res.render('roles/admin_moderator/transport_markk', {
                        markks: markks,
                        types: types,
                        typeUser: req.session.passport.user.userTypeID
                    });
                })
                .catch(error => {
                    console.warn(error);
                    res.render('roles/admin_moderator/users');
                });
        })
        .catch(error => {
            console.warn(error);
            res.render('roles/admin_moderator/users');
        });
});

router.post('/create-transport-markk',
validation.createAndUpdateTransportMarkk('create'), (req, res) => {
    TransportMarkk
        .createTransportMarkk(req.body)
        .then(result => {

            if (result.hasResult) {
                res.status(200).send({
                    transportMarkk: result.transportMarkk
                });
            }
            else {
                var msg = 'Марка транспорту "' + req.body.transportMarkkName + '" вже
існує. Скористайтеся пошуком.';
                req.flash('error_alert', true);
```

```
                req.flash('error_msg', {msg: msg});
                res.redirect('back');
            }
        })
        .catch(error => {
            console.warn(error);

            res.redirect('back');
        });
});

router.put('/update-transport-markk/:id', (req, res) => {
    TransportMarkk
        .updateTransportMarkk(req.params.id, req.body)
        .then(() => {
            res.status(200).send();
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.delete('/delete-transport-markk/:id', (req, res) => {
    TransportMarkk
        .deleteTransportMarkk(req.params.id)
        .then(() => {
            res.status(200).send();
        })
        .catch(error => {
            console.warn(error);
            res.status(400).send({errors: errors});
        });
});

/* ============== TRANSPORT MODEL ============== */
router.get('/transport-model', (req, res) => {
    TransportModel
        .getAllTransportModel()
        .then(transportModels => {
            TransportMarkk
                .getAllTransportMarkks()
                .then(transportMarkks => {
                    res.render('roles/admin_moderator/transport_model', {
                        transportModels: transportModels,
                        transportMarkks: transportMarkks,
                        typeUser: req.session.passport.user.userTypeID
                    });
                })
                .catch(error => {
                    console.warn(error);
                    res.render('roles/admin_moderator/requests/all');
                });
        })
        .catch(error => {
            console.warn(error);
            res.render('roles/admin_moderator/requests/all');
        });
});

router.post('/create-transport-model',
validation.createAndUpdateTransportModel('create'), (req, res) => {
    TransportModel
        .createTransportModel(req.body)
        .then((result) => {
            if (result.hasResult) {

                res.status(200).send({
                    transportModel: result.transportModel
                });
```

```
                // req.flash('success_alert', true);
                // req.flash('success_msg', 'Додавання моделі транспорту пройшло
успішно.');
                // res.redirect('back');
            }
            else {
                var msg = 'Модель транспорту "' + req.body.transportTypeName + '" вже
існує. Скористайтеся пошуком.';

                req.flash('error_alert', true);
                req.flash('error_msg', {msg: msg});
                res.redirect('back');
            }
        })
        .catch(error => {
            console.warn(error);
            req.flash('error_alert', true);
            req.flash('error_msg', {msg: 'Виникла помилка при додаванні моделі
транспорту.'});
            res.redirect('back');
        });
});

router.put('/update-transport-model/:id', validation.createAndUpdateTransportModel(),
(req, res) => {
    TransportModel
        .updateTransportModel(req.params.id, req.body)
        .then(() => {
            res.status(200).send();
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.delete('/delete-transport-model/:id', (req, res) => {
    TransportModel
        .deleteTransportModel(req.params.id)
        .then(() => {
            res.status(200).send();
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

//for create request page

router.get('/get-transport-type', (req, res) => {
    TransportType
        .getAllTransportType()
        .then(transportTypes => {
            res.status(200).send(transportTypes);
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.get('/get-transport-markk/:id', (req, res) => {
    TransportMarkk
        .getTransportMarkksOfTypeID(req.params.id)
        .then(markks => {
            res.status(200).send(markks);
        })
        .catch(errors => {
            console.warn(errors);
```

```
            res.status(400).send({errors: errors});
        });
});

router.get('/get-transport-model/:id', (req, res) => {
    TransportModel
        .getTransportModelOfMarkkID(req.params.id)
        .then(transportModels => {
            res.status(200).send(transportModels)
        })
        .catch(errors => {
            console.warn(errors);
            res.send(400).send({errors: errors});
        });
});

/* ============== DETAIL ============== */

router.get('/details', (req, res) => {
    Models.Detail
        .getAll()
        .then(details => {
            res.render('roles/admin_moderator/details', {
                typeUser: req.session.passport.user.userTypeID,
                details: details
            });
        })
        .catch(errors => {
            console.warn(errors);
            res.render('roles/admin_moderator/requests/all');
        });
});

router.post('/create-detail', validation.createAndUpdateDetail('create'), (req, res)
=> {
    req.body.typeID = true;

    Models.Detail
        .createDetail(req.body)
        .then((result) => {
            if (result.hasResult) {
                req.flash('success_alert', true);
                req.flash('success_msg', 'Додавання деталі пройшло успішно.');
                res.redirect(req.baseUrl + '/details');
            }
            else {
                let msg = 'Деталь "' + req.body.detailName + '" вже існує.
Скористайтеся пошуком.';

                req.flash('error_alert', true);
                req.flash('error_msg', {msg: msg});
                res.redirect(req.baseUrl + '/details');
            }
        })
        .catch(error => {
            console.warn(error);
            req.flash('error_alert', true);
            req.flash('error_msg', {msg: 'Виникла помилка при додаванні деталі.'});
            res.redirect(req.baseUrl + '/details');
        });
});

router.put('/update-detail/:id', validation.createAndUpdateDetail(), (req, res) => {
    Models.Detail
        .updateDetail(req.params.id, req.body)
        .then((detail) => {
            res.status(200).send({
                detail: detail[0]
            })
        })
```

```
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.delete('/delete-detail/:id', (req, res) => {
    Models.Detail
        .deleteDetail(req.params.id)
        .then(() => {
            res.status(200).send();
        })
        .catch(error => {
            console.warn(error);
            res.status(400).send({errors: errors});
        });
});

/* ============== DETAIL TYPE ============== */

router.get('/details-of-task/:id', (req, res) => {
    Models.TaskDetail
        .getTaskDetail(req.params.id)
        .then(details => {
            res.status(200).send({
                details: details
            })
        })
        .catch(error => {
            console.warn(error);
            res.status(400).send({errors: errors});
        });
});
module.exports = router;
```

2. Код контролера моделі «Інженер»

```
"use strict";

const express = require('express');
const router = express.Router();
const Task = require('../../models/Task');
const Request = require('../../models/Request');
const User = require('../../models/User');
const formatDate = require('../../helpers/formatDate');
const countEndTime = require('../../helpers/countEndTime');
const validation = require('../../middleware/validation');
const status = require('../../constants/status');
const Models = require('../../models/TaskDetail/relations');

router.get('/', (req, res) => {
    Task
        .getTaskByExecutorId(req.session.passport.user.id)
        .then(result => {
            for (var i = 0; i < result.length; i++) {
                result[i].dataValues.startTime =
formatDate(result[i].dataValues.startTime);
                result[i].dataValues.endTime =
formatDate(result[i].dataValues.endTime);
            }

            User
                .getAllUsers()
```

```
                .then(users => {
                    res.render('roles/executor', {
                        typeUser: req.session.passport.user.userTypeID,
                        tasks: result,
                        assignedExecutorUsers: users
                    });
                })
                .catch(errors => {
                    console.warn(errors);
                    res.status(400).send({errors: errors});
                });

        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.put('/update-task/:id', validation.createAndUpdateTask(), (req, res) => {
    req.body.endTime = countEndTime(req.body.startTime, +req.body.estimationTime);

    Task
        .updateTask(req.body.id, req.body)
        .then(() => {

            Request
                .getRequestById(req.body.requestID)
                .then(request => {

                    Task
                        .getTaskById(req.body.id)
                        .then(task => {

                            Models.TaskDetail
                                .createTaskDetail(req.body.id,
JSON.parse(req.body.detail))
                                .then(() => {

                                    Models.TaskDetail

.updateDetailType(JSON.parse(req.body.changeDetail))
                                        .then(() => {

                                            Models.TaskDetail

.deleteTaskDetail(JSON.parse(req.body.deleteDetail))
                                                .then(() => {

                                                    Models.TaskDetail
                                                        .getTaskDetail(req.body.id)
                                                        .then(details => {

                                                            res.status(200).send({
                                                                request: request,
                                                                task: task,
                                                                details: details
                                                            });
                                                        })
                                                        .catch(errors => {
                                                            console.warn(errors);

res.status(400).send({errors: errors});

                                                        });
                                                })
                                                .catch(errors => {
                                                    console.warn(errors);
                                                    res.status(400).send({errors:
errors});

                                                });
```

```
                                        })
                                        .catch(errors => {
                                            console.warn(errors);
                                            res.status(400).send({errors: errors});
                                        });
                                })
                                .catch(errors => {
                                    console.warn(errors);
                                    res.status(400).send({errors: errors});
                                });
                        })
                        .catch(errors => {
                            console.warn(errors);
                            res.status(400).send({errors: errors});
                        });
                })
                .catch(errors => {
                    console.warn(errors);
                    res.status(400).send({errors: errors});
                });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

router.put('/set-task-status/:id', (req, res) => {
    Task
        .updateTask(req.params.id, req.body)
        .then(() => {
            Task
                .getTaskById(req.params.id)
                .then(task => {
                    Task
                        .getAllTasksStatusOfRequest(task.requestID)
                        .then(tasks => {
                            var counter = 1;
                            for (var key in tasks) {
                               if (tasks[key].dataValues.status !== 3) {
                                   break;
                               }
                               if(counter === tasks.length) {
                                   Request
                                        .changeStatus(task.requestID,
{status:status.DONE})
                                        .then(() => {})
                                        .catch(errors => {
                                            console.warn(errors);
                                            res.status(400).send({errors: errors});
                                        });
                               }
                               counter++;
                            }

                            res.status(200).send();
                        })
                        .catch(errors => {
                            console.warn(errors);
                            res.status(400).send({errors: errors});
                        });
                })
                .catch(errors => {
                    console.warn(errors);
                    res.status(400).send({errors: errors});
                });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
```

```
        });
});

/* ============== DETAIL TYPE ============== */

router.get('/details-of-task/:id', (req, res) => {
    Models.TaskDetail
        .getTaskDetail(req.params.id)
        .then(details => {
            res.status(200).send({
                details: details
            })
        })
        .catch(error => {
            console.warn(error);
            res.status(400).send({errors: errors});
        });
});


router.post('/get-task-types', (req, res) => {
    Request
        .getRequestById(req.body.requestID)
        .then((request) => {
            Models.Detail
                .getDetailsByCarAttributes(request[0].transportTypeID,
request[0].transportMarkkID, request[0].transportModelID)
                .then(detailTypes => {
                    res.status(200).send({
                        detailTypes: detailTypes
                    });
                })
                .catch(errors => {
                    console.warn(errors);
                    res.status(400).send({errors: errors});
                });
        })
        .catch(errors => {
            console.warn(errors);
            res.status(400).send({errors: errors});
        });
});

module.exports = router;
```

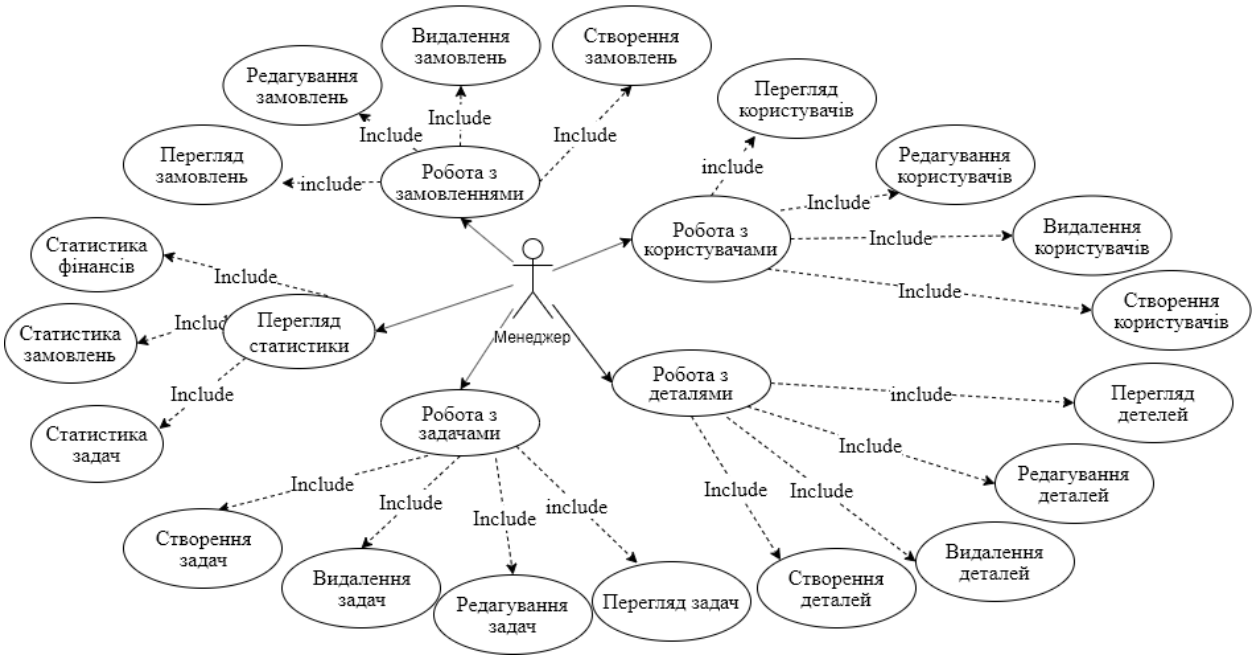
3. Код контролера моделі «Клієнт»

```
"use strict";

const express = require('express');
const router = express.Router();
const Task = require('../../models/Task');
const Request = require('../../models/Request');
const requestsFactory = require('../../helpers/requestsFactory');

router.get('/:status', (req, res) => {
    var findBy = {
        customerID: req.session.passport.user.id
    };

    if (req.params.status === 'active') {
        findBy.giveOut = false;
    }
    else {
        findBy.giveOut = true;
    }

    Request
        .getAllRequestsByCustomerId(findBy)
        .then(requests => {
            Task
                .getAllTasks()
                .then(tasks => {
                    res.render('roles/customer', {
                        user: req.session.passport.user,
                        requests: requestsFactory(requests, tasks),
                        typeUser: req.session.passport.user.userTypeID
                    });
                })
                .catch(error => {
                    console.warn(error);
                    res.render('roles/customer');
                });
        })
        .catch(error => {
            console.warn(error);
            res.render('roles/customer');
        });
});

module.exports = router;
```

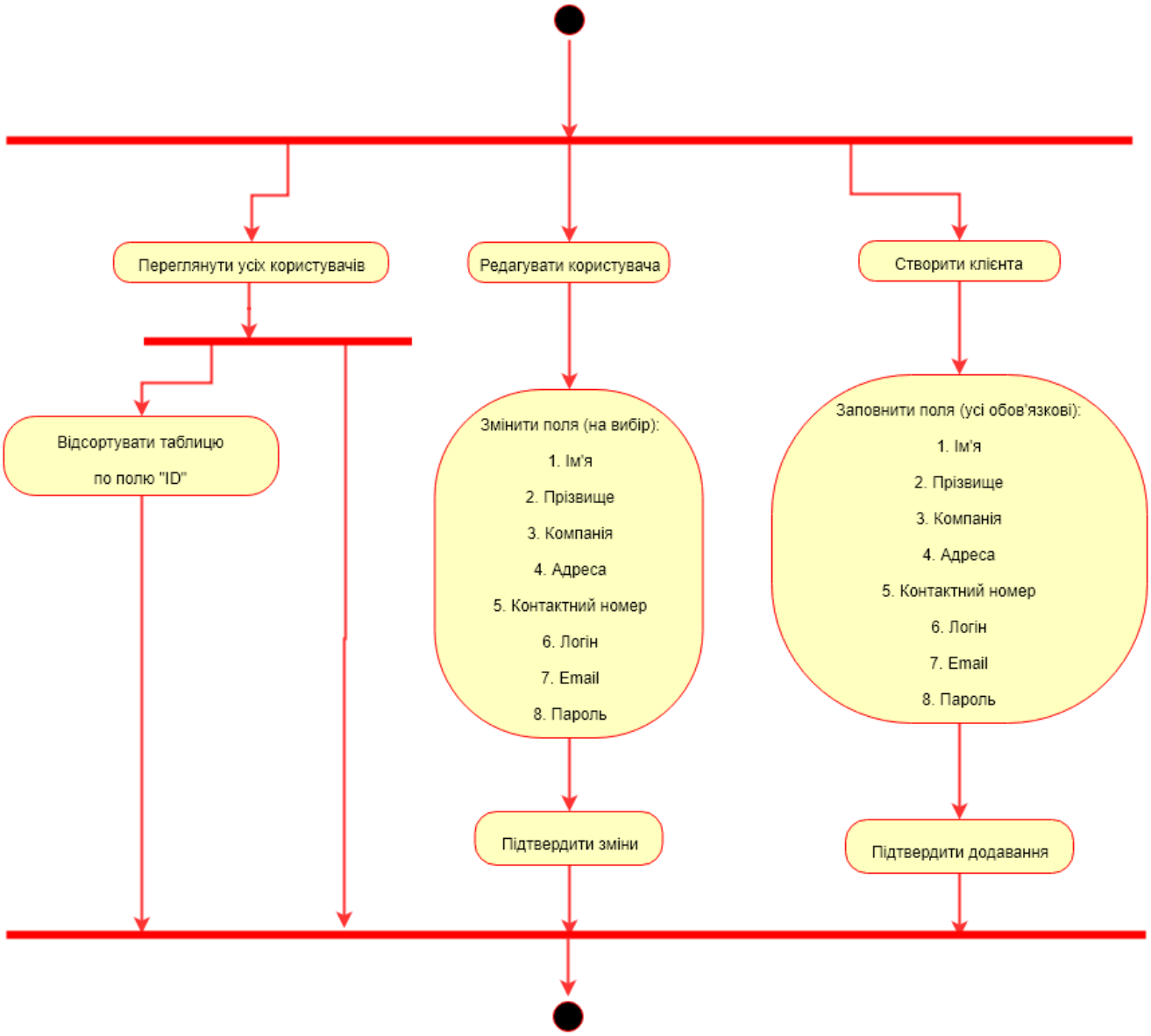
Додаток Г

Графічна частина

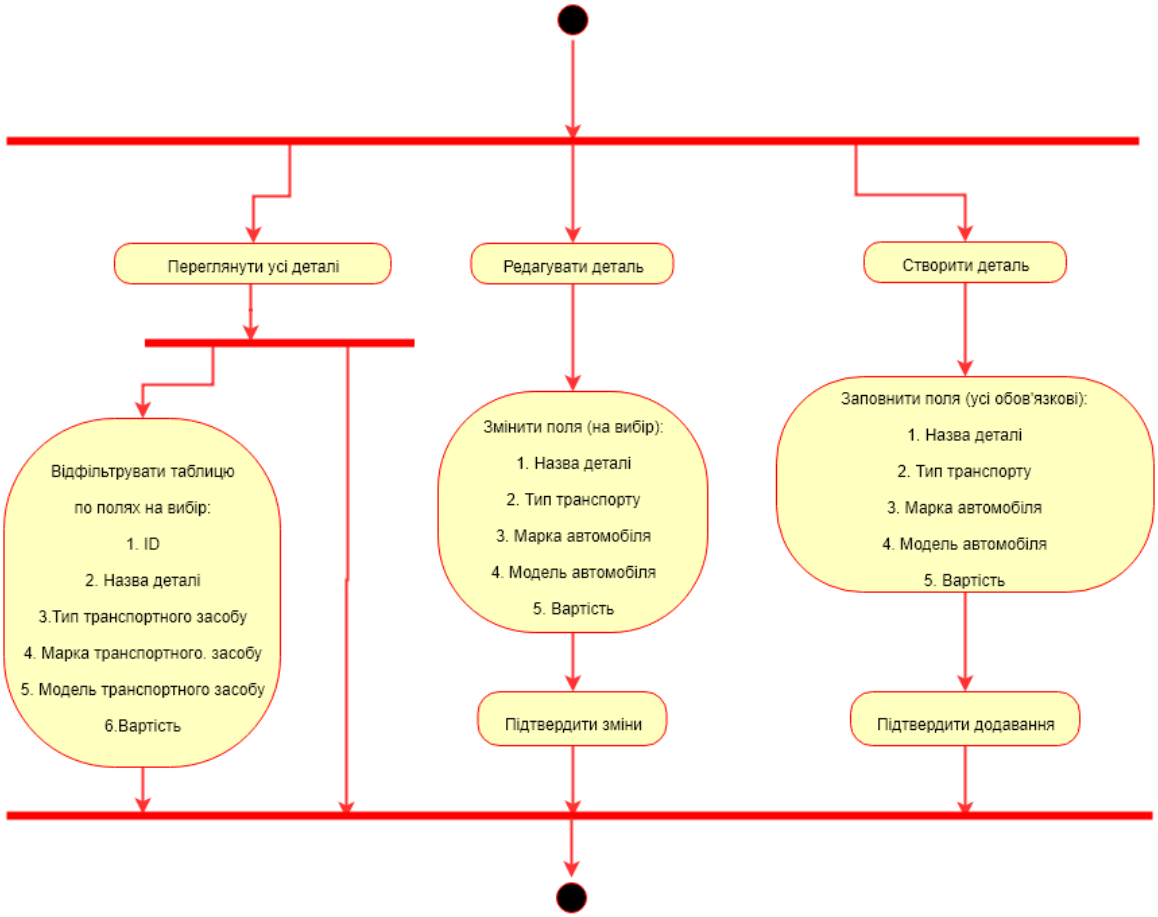
UML діаграма прецедентів модуля «Менеджер»



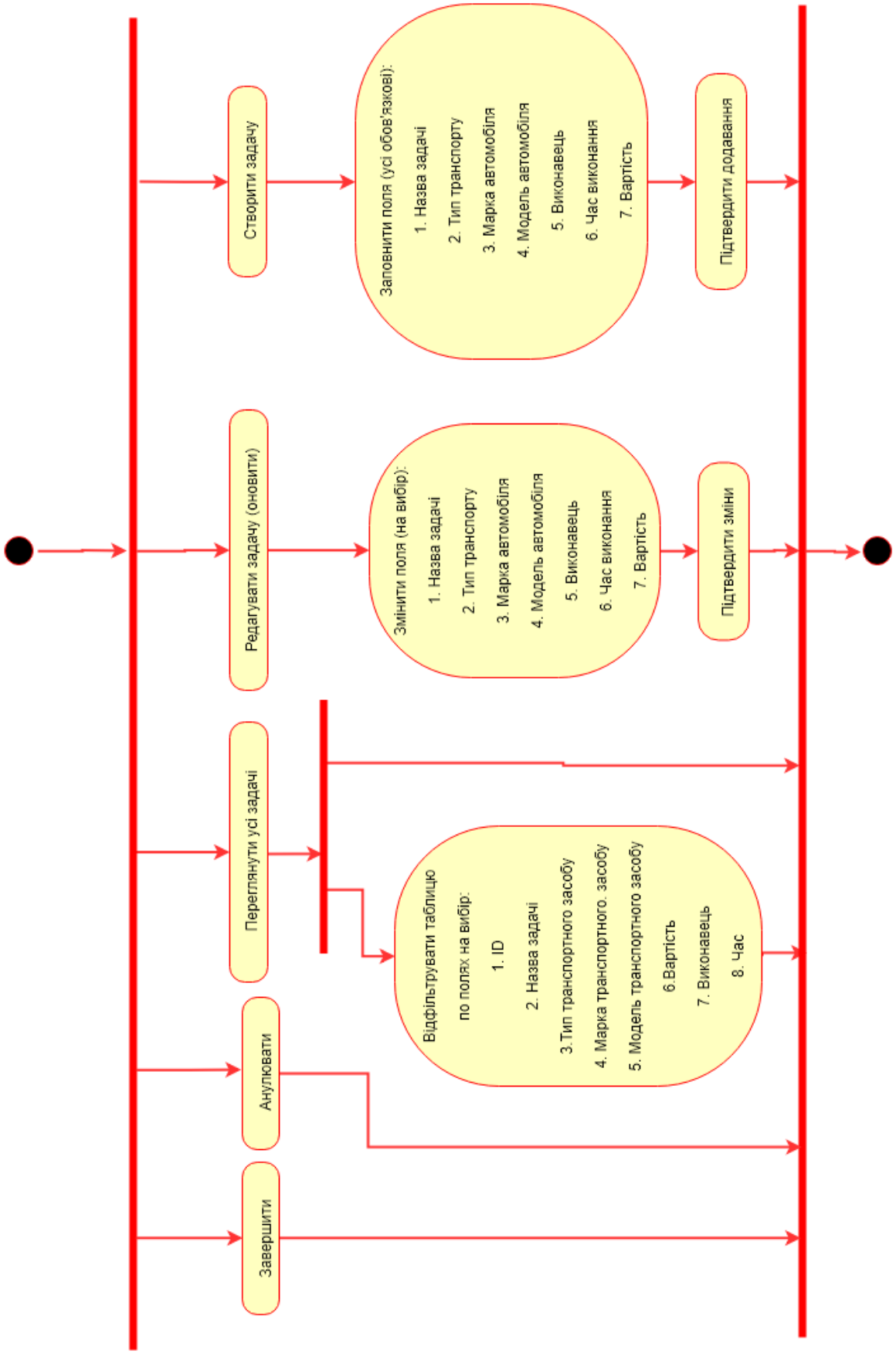
UML діаграма діяльності модуля «Менеджер»: користувачі



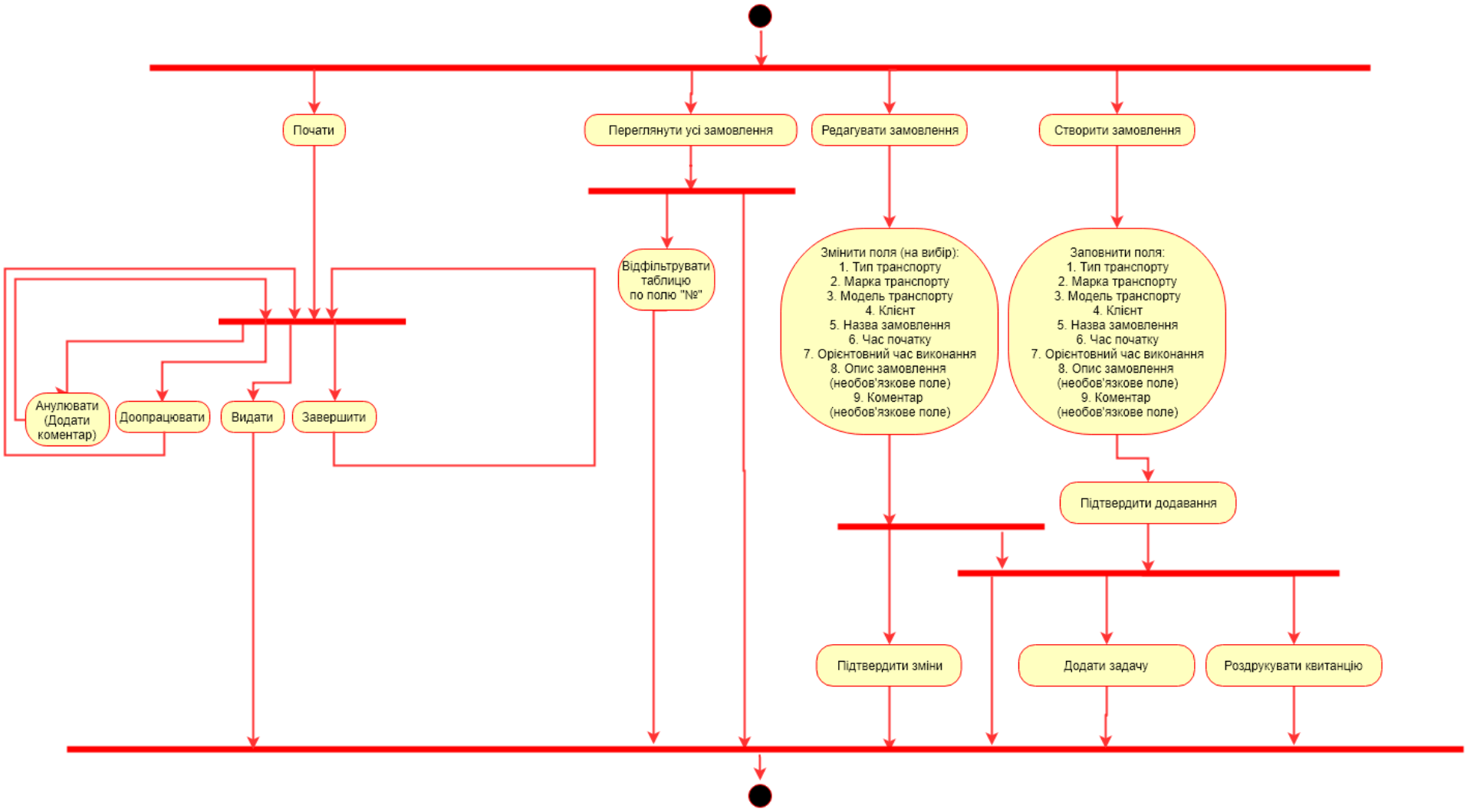
UML діаграма діяльності модуля «Менеджер»: деталі



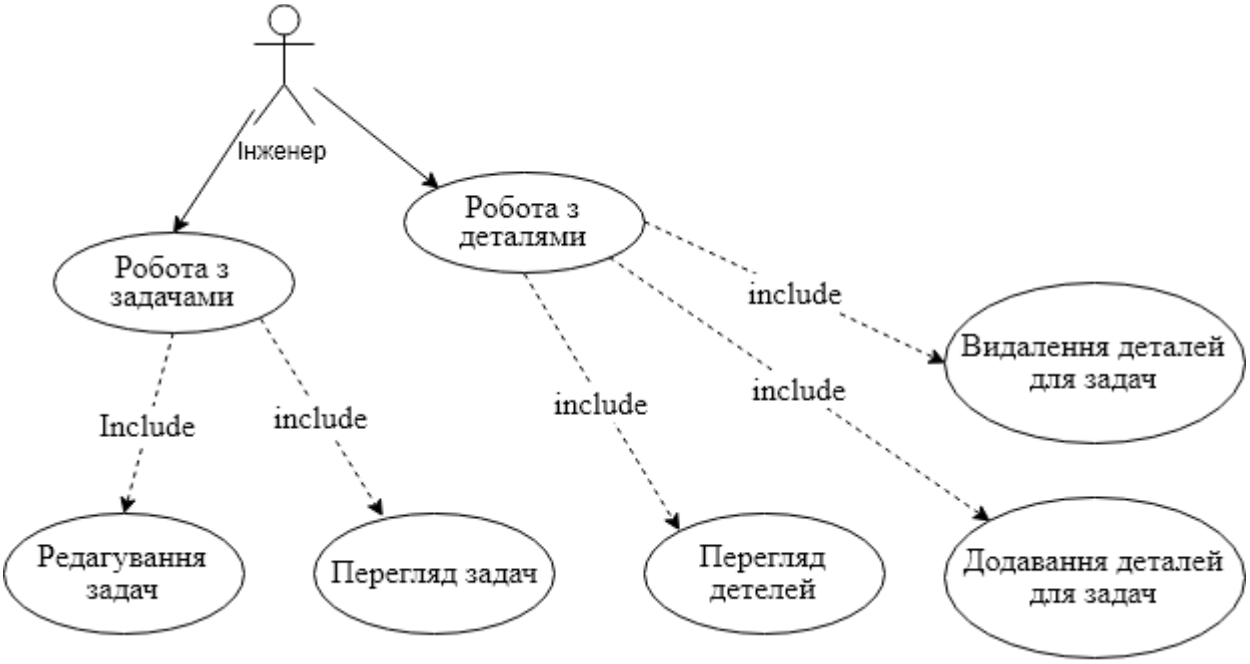
UML діаграма діяльності модуля «Менеджер»: задачі



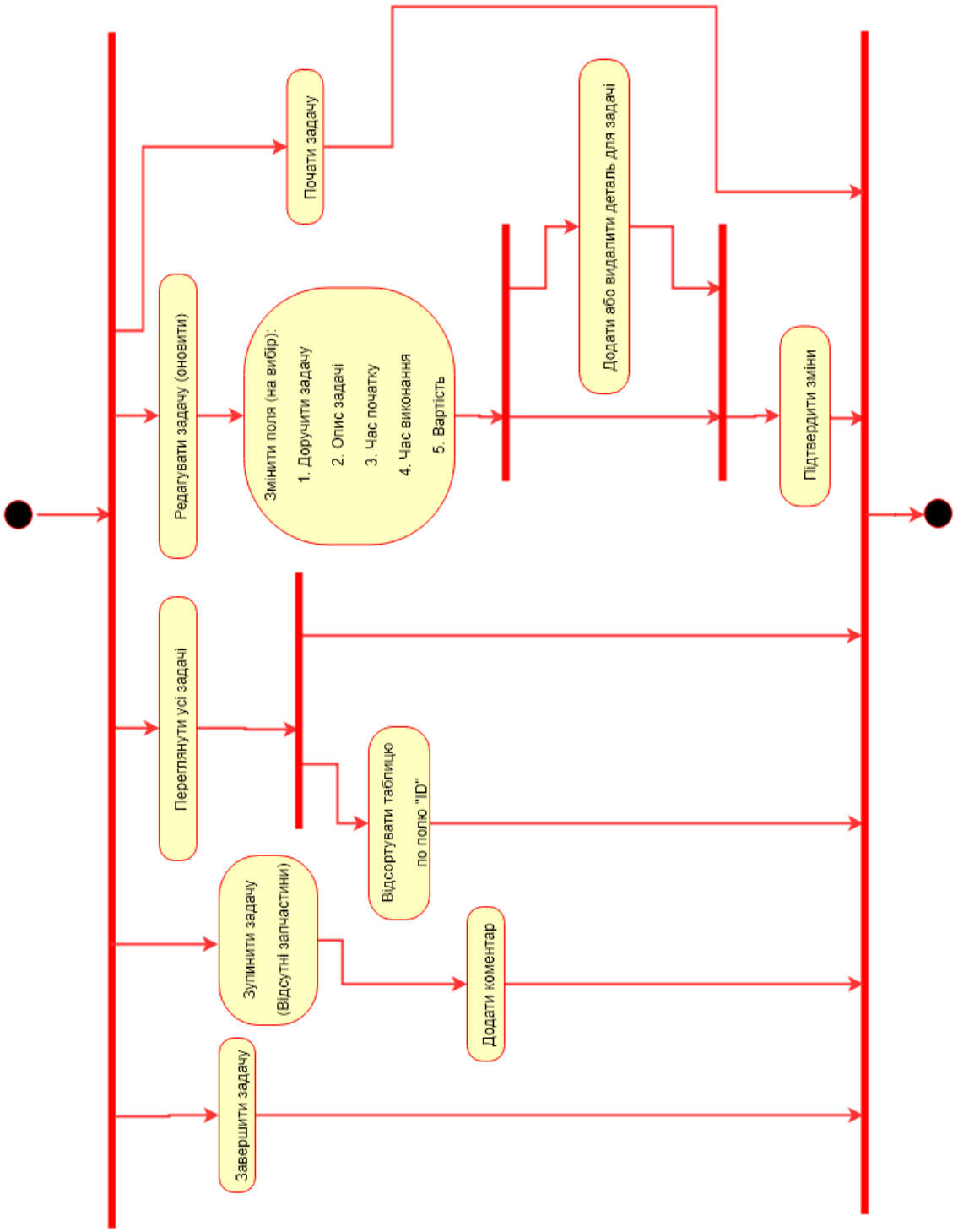
UML діаграма діяльності модуля «Менеджер»: замовлення



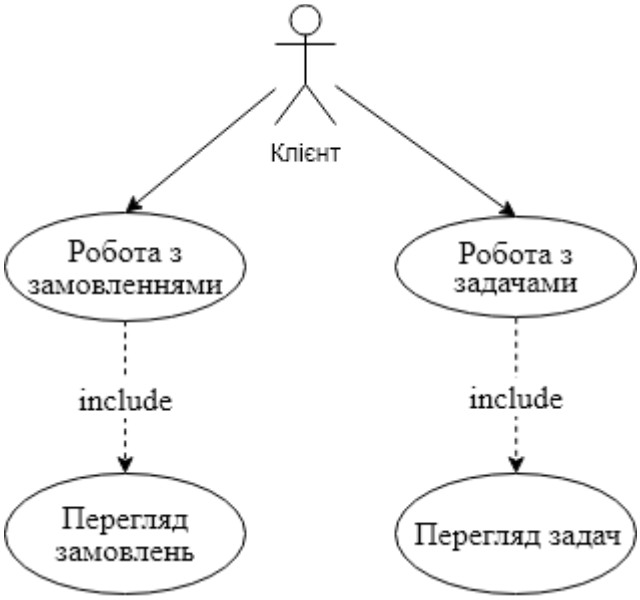
UML діаграма прецедентів модуля «Інженер»



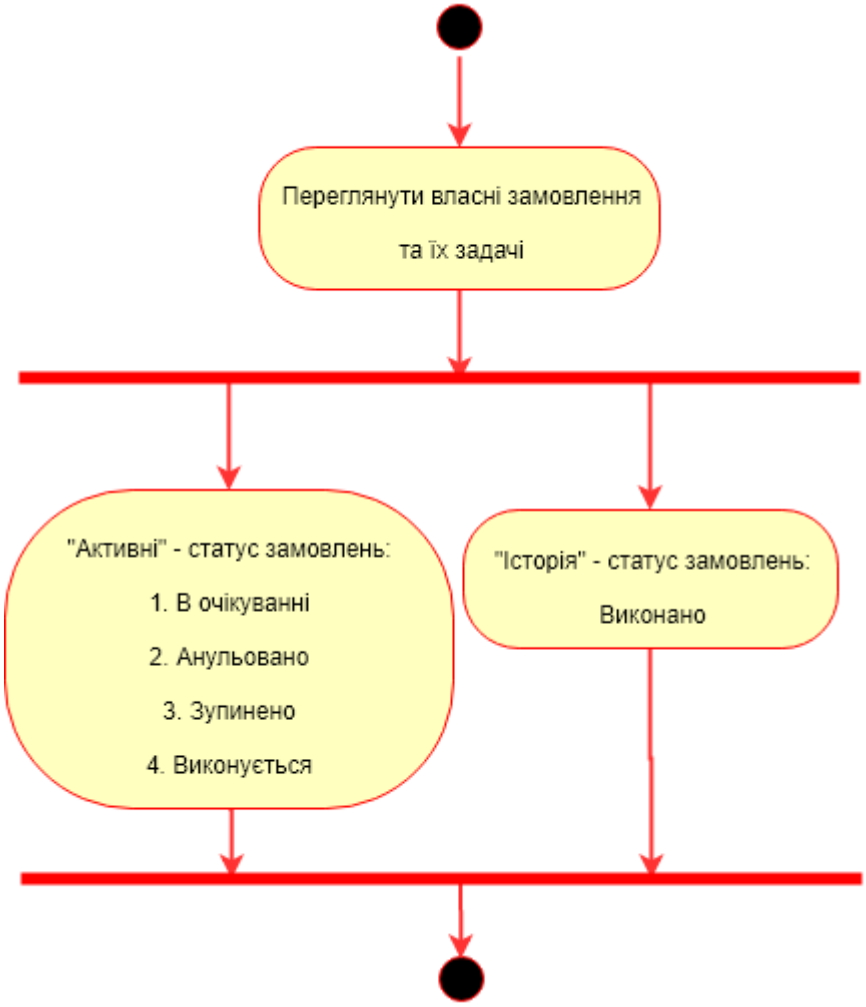
UML діаграма діяльності модуля «Інженер»: задачі



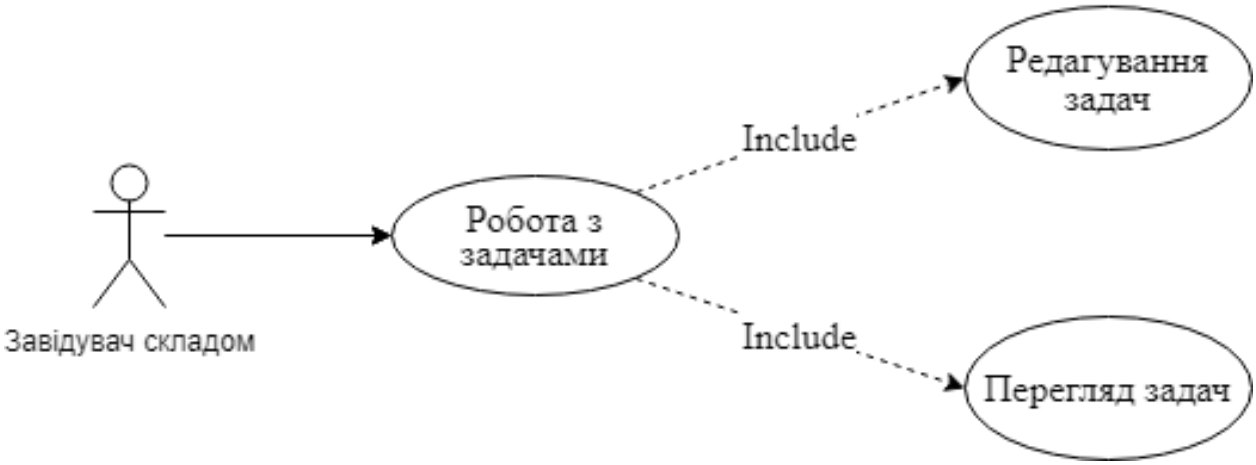
UML діаграма прецедентів модуля «Клієнт»



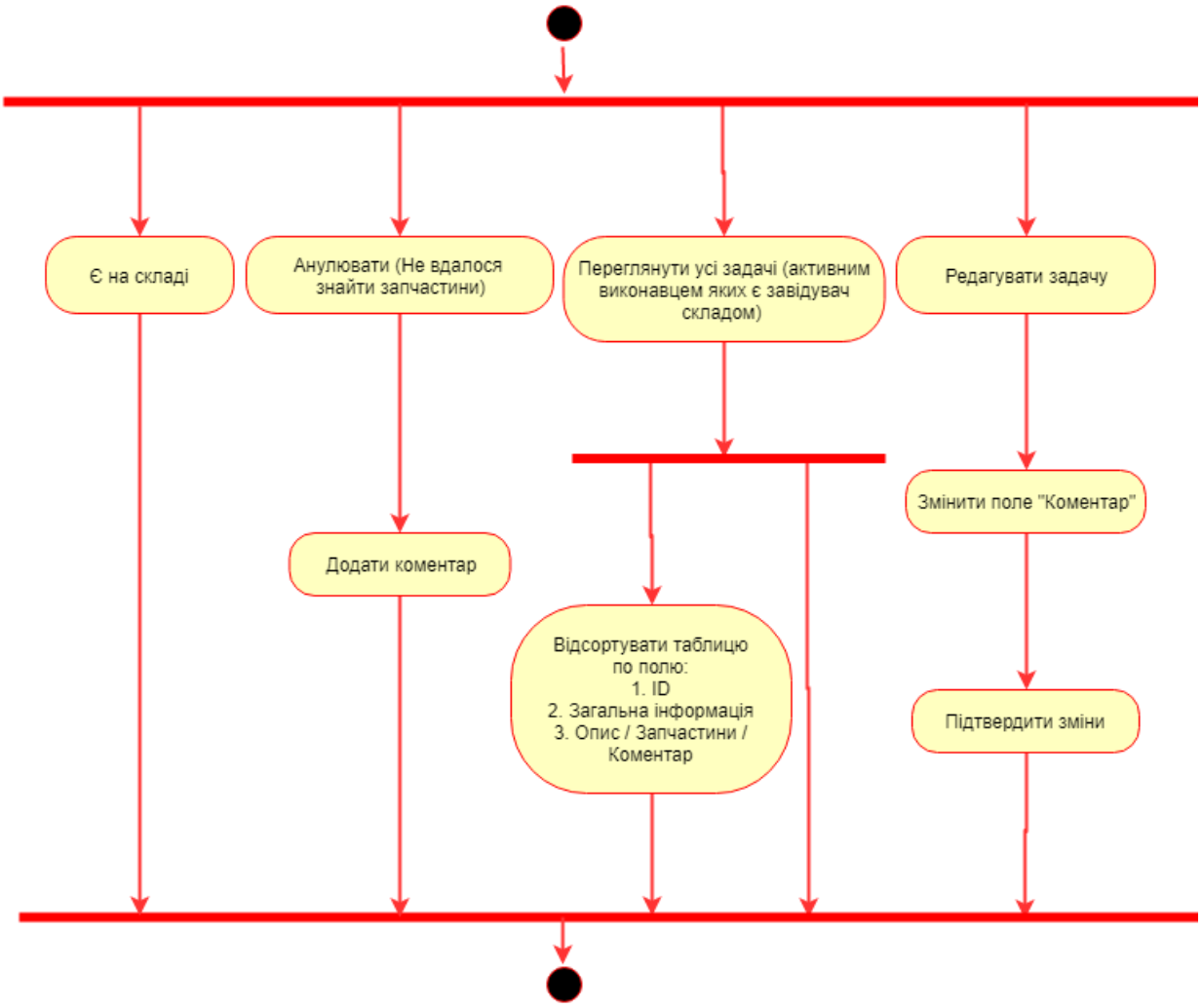
UML діаграма діяльності модуля «Клієнт»



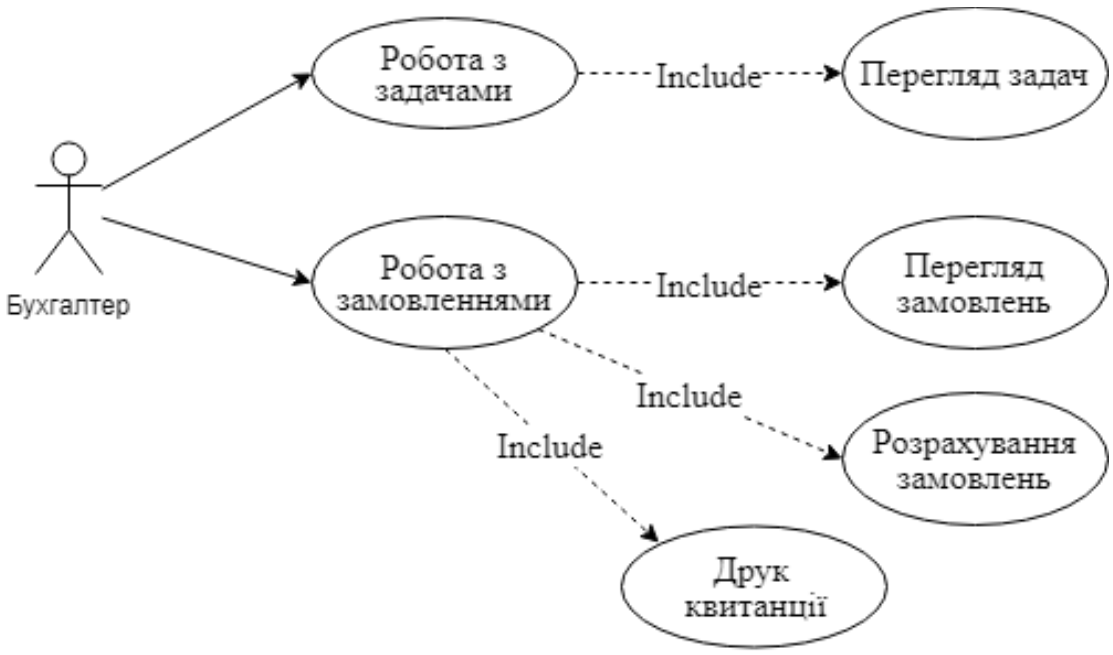
UML діаграма прецедентів модуля «Завідувач складом»



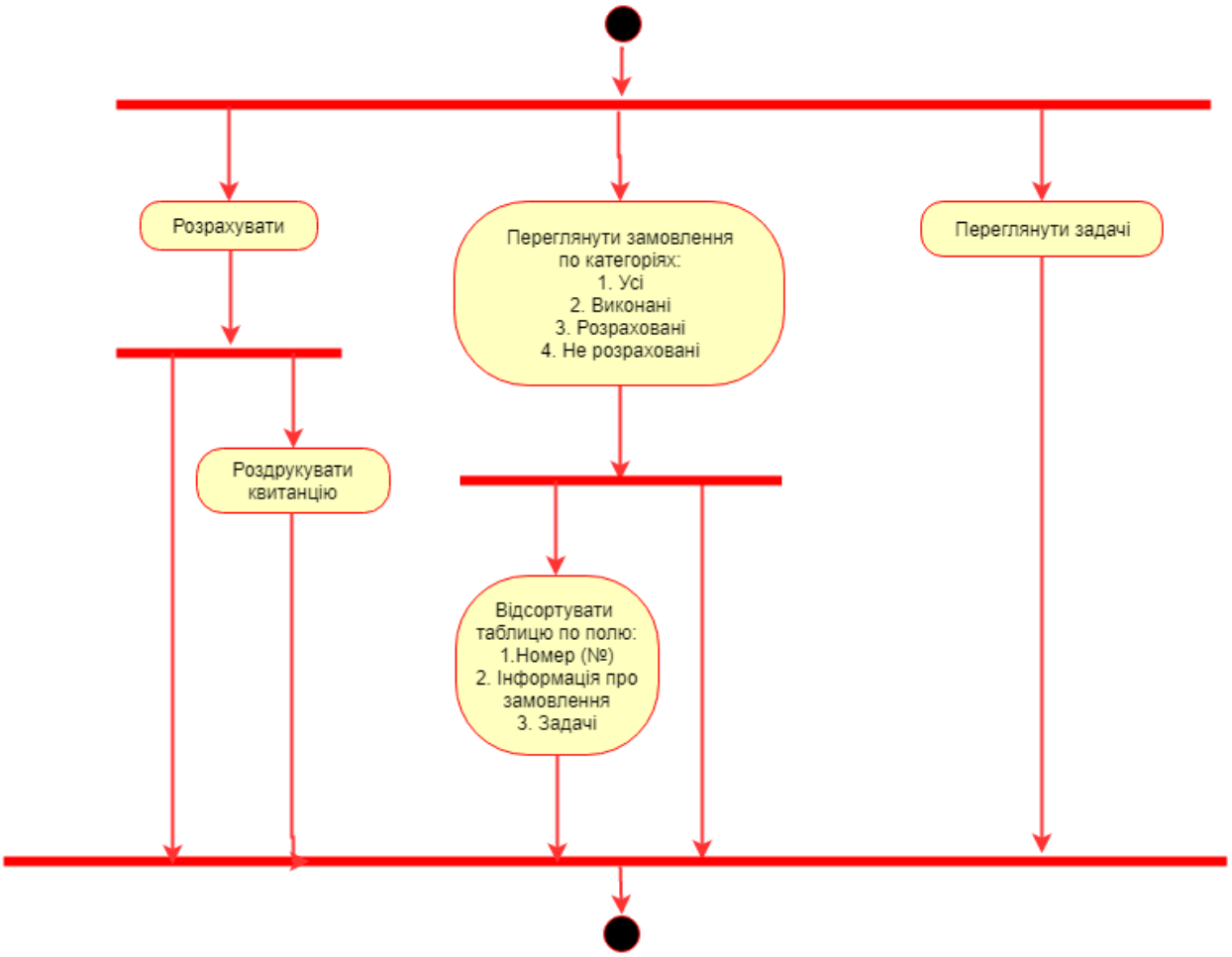
UML діаграма діяльності модуля «Завідувач складом»



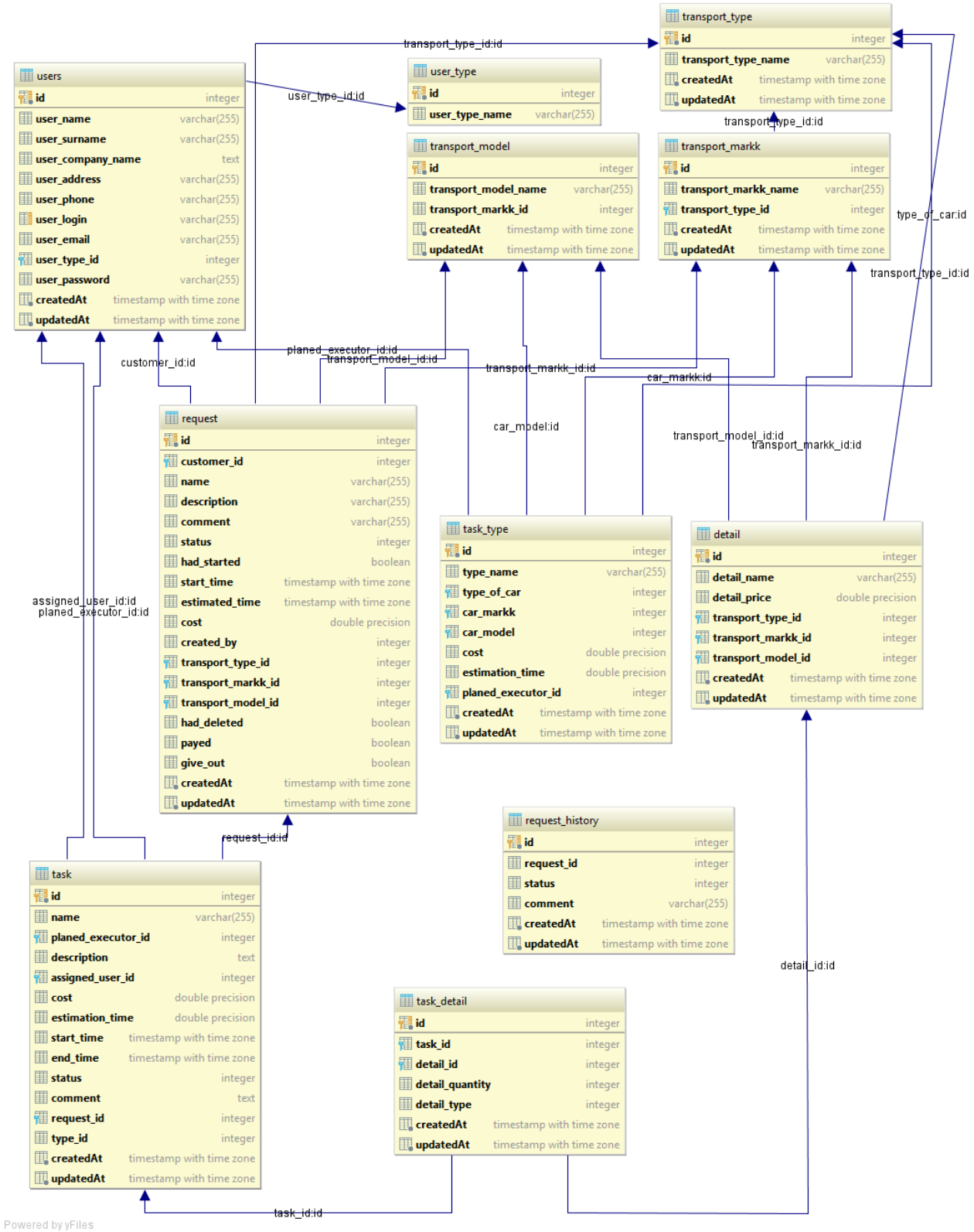
UML діаграма прецедентів модуля «Бухгалтер»



UML діаграма діяльності модуля «Бухгалтер»



UML діаграма бази даних



Додаток Д

Довідка про впровадження