# Програмне та ТЕХНічне забезпечення

## **Засоби розробки**

При створенні програмного продукту середовищем розробки було обрано Visual Studio 2010 для програмування мовою C#, використовуючи плагін ReSharper.

Дане середовище дозволяє розробляти як консольні застосунки, так і застосунки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторингу коду. Вбудовані інструменти включають в себе редактор форм для спрощення створення графічного інтерфейсу програми, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати і підключати сторонні доповнення (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду, додавання нових наборів інструментів.

Плагін ReSharper дуже зручно використовувати при розробці мовою C#. Він проводить статичний аналіз коду (пошук помилок в коді до компіляції) в масштабі всього рішення, передбачає додаткові засоби автозаповнення, навігації, пошуку, підсвічування синтаксису, форматування, оптимізації та генерації коду, надає 40 автоматизованих рефакторингів.

C# – сучасна мова програмування, яка має дуже багато переваг перед такими мовами, як C/C++ або Java. Ось неповний їх список:

* C# створювався паралельно з платформою Microsoft .NET Framework і повною мірою враховує всі його можливості;
* C# є повністю об'єктно-орієнтованою мовою, де навіть типи, вбудовані в мову, представлені класами;
* C# є потужною об'єктною мовою з можливостями наслідування і універсалізації;
* C# є спадкоємцем мов C/C++, зберігаючи кращі риси цих популярних мов програмування. Спільний з цими мовами синтаксис, знайомі оператори мови полегшують перехід програмістів від С++ до C#;
* зберігши основні риси C++, мова стала простіше і надійніше. Простота і надійність, головним чином, пов'язані з тим, що на C# хоча і допускаються, але не заохочуються такі небезпечні властивості С++ як покажчики, адресація, розіменування, адресна арифметика;
* завдяки платформі Microsoft .NET Framework, що стала надбудовою над операційною системою, програмісти C # отримують ті ж переваги роботи з віртуальною машиною, що і програмісти Java. Ефективність коду навіть підвищується, оскільки виконавча середа CLR є компілятором проміжної мови, в той час як віртуальна Java-машина є інтерпретатором байт-коду;
* потужна бібліотека платформи підтримує зручність побудови різних типів застосунків на C#, дозволяючи легко будувати Web-служби, інші види компонентів, достатньо просто зберігати і отримувати інформацію з бази даних.

Для збереження інформації було обрано систему управління реляційними базами даних Microsoft SQL Server 2008 R2.

Це сучасна відмовостійка СУБД, яку дуже зручно використовувати при розробці застосунків на основі платформи Microsoft .NET Framework. Платформа надає спеціальні засоби для роботи з Microsoft SQL Server.

Ця СУБД використовується для роботи з базами даних розміром від персональних до великих баз даних масштабу підприємства.

В якості системи контролю версій було використано розподілену систему керування версіями файлів git.

Використання розподіленої системи контролю версій дає наступні переваги (над централізованими):

* періодична синхронізація декількох комп'ютерів під управлінням одного розробника (робочого комп'ютера, домашнього комп'ютера, ноутбука і так далі). Використання розподіленої системи позбавляє від необхідності виділяти один з комп'ютерів в якості сервера, а синхронізація виконується за потребою;
* спільна робота над проектом невеликою територіально розподіленою групи розробників без виділення загальних ресурсів. Як і в попередньому випадку, реалізується схема роботи без головного сервера, а актуальність репозиторіїв підтримується періодичними синхронізації за схемою «кожен з кожним».
* великий розподілений проект, учасники якого можуть довгий час працювати кожен над своєю частиною, при цьому не мають постійного підключення до мережі. Такий проект може використовувати централізований сервер, з яким синхронізуються копії всіх його учасників. Можливі й більш складні варіанти - наприклад, зі створенням груп для роботи за окремими напрямками всередині більшого проекту. При цьому можуть бути виділені окремі «групові» сервери для синхронізації роботи груп, тоді процес остаточного злиття змін стає деревовидним: спочатку окремі розробники синхронізують зміни на групових серверах, потім оновлені репозиторії груп синхронізуються з головним сервером. Можлива робота і без «групових» серверів, тоді розробники однієї групи синхронізують зміни між собою, після чого будь-який з них (наприклад, керівник групи) передає зміни на центральний сервер.

Використання git дає наступні переваги (над аналогічними розподіленими системами контролю версій):

* висока продуктивність;
* продумана система команд;
* репозиторії git можуть поширюватися і оновлюватися загальносистемними файловими утилітами резервного копіювання та оновлення, завдяки тому, що фіксації змін і синхронізації не змінюють існуючі файли з даними, а лише додають нові (за винятком деяких службових файлів, які можуть бути автоматично оновлені за допомогою наявних у складі системи утиліт);
* для поширення репозиторію по мережі досить будь-якого веб-сервера.

## **Вимоги до технічного забезпечення**

Головною метою розробки даного програмного продукту є обробка даних про складські приміщення, об’єми товарів і замовлень з метою отримання плану перевезень між складами і споживачами. Він призначений для використання співробітником підприємства, що працює у відповідному напрямі.

Для можливості роботи з даним програмним продуктом до складу технічних засобів повинні входити:

1. комп’ютер з наступною конфігурацією:

1) процесор з тактовою частотою не нижче 1.4 ГГц;

1. об’єм оперативної пам’яті не менше 512 МБ;
2. наявність 20 Мб вільного простору на жорсткому диску;
3. графічна карта з об’ємом відео-пам’яті не менше 32 Мб;
4. додатково має бути встановлене наступне програмне забезпечення:
5. операційна система Windows XP/Vista/7;
6. СУБД Microsoft SQL Server 2008 R2;
7. платформа Microsoft .NET Framework 4.0;
8. комп’ютерна периферія, до складу якої входить:
9. монітор;
10. мишка;
11. клавіатура.

## **Архітектура програмного забезпечення**

### **Діаграма класів**

У додатку В, лист? (Схема структурна класів) представлена структурна схема класів, які відповідають за виконання основних функцій програми: обробка даних, складання плану і маршрутів перевезень.

Діаграма містить наступні класи:

* TaskController – клас, функціями якого є зв’язок користувацького інтерфейсу з класами алгоритмів (запуск, зупинка алгоритму), графічне відображення вузлів, збереження усіх вузлів, робота з файлами;
* Algorithm – клас алгоритму, в якому реалізована спільна для ітеративних алгоритмів логіка: запуск одної ітерації, запуск певної кількості ітерацій, проведення ітерацій, поки не буде досягнуто розв’язку з заданою точністю. Також даний клас має функцію ведення лог-файлу для оцінки ефективності алгоритму;
* BeesColony – клас, в якому реалізовано бджолиний алгоритм в загальному вигляді: початкова ініціалізація, перехід до кращого сусіднього розв’язку, генерація нових випадкових розв’язків;
* ClusteringAlgorithm - реалізує спільну логіку для алгоритмів кластеризації: збереження кластерів, додавання кластера, підрахунок цільової функції;
* KMeans – клас реалізує логіку алгоритму кластеризації k‑means;
* NearestNeighbourChain – клас реалізує логіку алгоритму кластеризації nearest-neighbor chain;
* ClusteringToTsp – клас, що складає маршрути обходу кожного з кластерів, що їх було отримано попередньо алгоритмами кластеризації, підраховує кінцеву оцінку витрат на перевезення і штрафи;
* Site – клас, який представляє розв’язок бджолиного алгоритму, а також реалізовує спільну логіку пошуку сусідніх розв’язків і переходу до кращого з них;
* SiteVrpTsp – клас розв’язку задачі VRP, що зводиться до TSP. Конкретизує логіку пошуку сусідніх розв’язків під цю задачу, обчислює відстані між вузлами;
* SiteClusteringVrp - клас розв’язку задачі кластеризації для VRP. Конкретизує логіку пошуку сусідніх розв’язків під цю задачу, обчислює суму квадратів відстаней вузлів від центру кластера для кожного з кластерів;
* SiteClusteringCvrp
* SiteClusteringCvrpp
* SiteClusteringCvrppNnc
* Node
* Connection
* Cluster
* SerializationData

### **Діаграма послідовності**

На рисунку 4.2 представлена схема структурна послідовності. На даній діаграмі представлена типова послідовність дій, які виконуються при складенні оптимального плану виготовлення продукції, надання підприємству прогнозів плану виготовлення продукції на майбутні періоди та формуванні звіту «План виробництва». Також визначено акторів та приналежність їм відповідних дій, необхідних для виконання поставленої задачі.

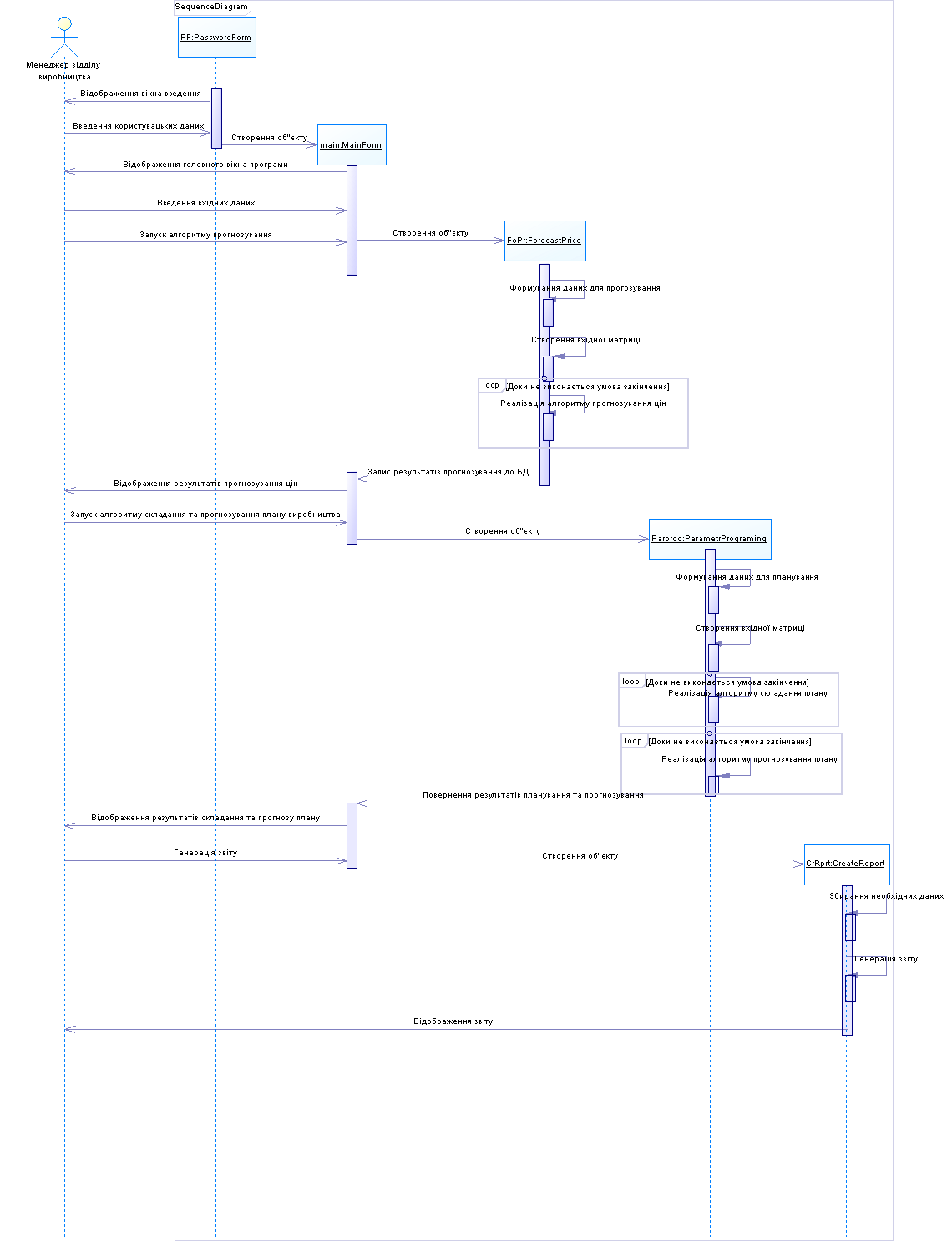


Рисунок 4.2 – Схема структурна послідовності

Далі, за допомогою діаграм послідовності опишемо дії, які виконуються при складанні плану випуску продукції та його прогнозуванні, а також при прогнозуванні ринкових цін.

На рисунку 4.3 представлена схема структурна послідовності дій роботи алгоритму. На даній діаграмі представлена типова послідовність дій, які виконуються при виконання алгоритму планування та прогнозування.



Рисунок 4.3 - Схема структурна послідовності дій роботи алгоритму планування та прогнозування

На рисунку 4.4 представлена схема структурна послідовності дій роботи алгоритму, який використовується при прогнозуванні ринкових цін за ретроспективними даними. На даній діаграмі представлена типова послідовність дій, які виконуються при виконання алгоритму прогнозування ринкових цін.

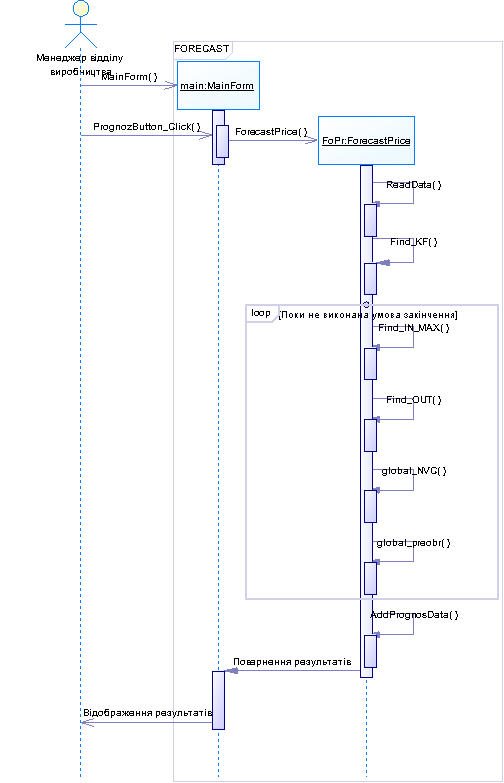


Рисунок 4.4 - Схема структурна послідовності дій роботи алгоритму прогнозування ринкових цін

### **Діаграма компонентів**

На рисунку 4.5 представлена схема структурна компонентів, на якій відображені компоненти, що використовуються в комплексі задач, та взаємозв’язки між ними. Основними компонентами в системі є співробітник підприємства, клієнтське застосування та СУБД Oracle.



Рисунок 4.5 – Схема структурна компонентів

### **Специфікація функцій**

Функції класів програмного забезпечення наведені в таблиці 4.9.

Таблиця 4.9 – Функції класів програмного забезпечення

| **Клас** | **Функція** | **Опис функції** |
| --- | --- | --- |
| Algorithm – абстрактний клас алгоритму, в якому реалізована спільна для ітеративних алгоритмів логіка | public abstract void DrawNodes() | Функція, яка реалізовує графічне відображення вузлів |
| public virtual string Info() | Функція повертає специфічну інформацію про поточний стан алгоритму |
| public virtual void StartingInitialize() | Ініціалізація на початку алгоритму бджолиних колоній для першого розв’язку |
| protected abstract void InnerIteration() | Специфічна ітерація для певного алгоритму |
| private void Iteration() | Проведення специфічної ітерації а також збір загальної статистики по алгоритму, ведення логу |
| public void Iterations (int count = 1) | Проведення певної кількості ітерацій |
| public void IterateToStop() | Виконання ітерації доки не буде досягнуто критерію зупину |
| private void OpenLogFile() | Відкриття лог-файлу для дозапису |
| public void SetNodes (List<Node> nodesForSet) | Задання вузлів для обробки у алгоритмі |
| public void Stop() | Зупинка алгоритму |
| BeesColony – клас-нащадок класу Algorithm. Клас, в якому реалізовано бджолиний алгоритм в загальному вигляді | private Site CreateNewSite() | Створює випадковий розв’язок задачі |
| public void CreateSites() | Генерація початкового набору розв’язків |
| public override void DrawNodes() | Функція, яка реалізовує графічне відображення вузлів |
| public override string Info() | Повертає інформацію про поточний стан алгоритму |
| protected override void InnerIteration() | Ітерація бджолиного алгоритму |
| Site – абстрактний клас, який представляє розв’язок бджолиного алгоритму а також реалізовує спільну логіку пошуку сусідніх розв’язків і переходу до кращого з них | public virtual int CompareTo(object obj) | Функція порівняння даного об’єкту з іншим |
| public void DrawNodes() | Графічне відображення |
| private List<Site> GenerateNeighbours (int count) | Генерація сусідніх розв’язків |
| protected abstract Site GetNeighbour() | Функція генерації сусіднього розв’язку |
| public bool GoToBestNeighbour (int countOfNeightbours) | Перехід до кращого сусіднього розв’язку, якщо такий є |
| protected abstract void GoToNeighbour (Site site) | Перехід до сусіднього розв’язку |
| public abstract List<Node> PrepareToDraw (Color connectionsColor) | Підготувати вузли до графічного відображення |
| SiteVrpTsp – клас-нащадок класу Site. Клас розв’язку задачі VRP, що зводиться до TSP | private void ConnectConsumerToDepot (Node node1, Node node2, Color connectionColor) | Графічно приєднати вузол-споживач до вузла-складу. |
| private static double[,] GeneratePricesByPositions (List<Node> nodes, int depotsCount, int consumersCount) | Згенерувати таблицю відстаней |
| private void GenerateSequence (int depotsCount, int clustersCount, int consumersCount) | Згенерувати послідовність обходження вузлів |
| private static double GetDistance (Node node1, Node node2) | Обчислення відстані від одного вузла до іншого |
| protected override Site GetNeighbour () | Згенерувати сусідній розв’язок |
| protected override void GoToNeighbour (Site site) | Перейти до сусіднього розв’язку |
| private static void Interchange (int[] arr, int i1, int i2) | Поміняти місцями елементи масиву |
| private static void Invert (int[] arr, int i1, int i2) | Інвертувати частину масиву |
| public void InvertRandomPartOfNodesSequence () | Інвертувати випадкову частину масиву |
| public override List<Node> PrepareToDraw (Color connectionsColor) | Підготувати вузли до графічного відображення |
| public override string ToString() | Функція повертає відображення розв’язку у вигляді рядку |
| SiteClusteringVrp – клас-нащадок класу Site. Клас розв’язку задачі кластеризації для VRP | protected List<Node> GenerateClusters() | Генерація кластерів випадковим чином |
| protected virtual List<Node> GenerateClusters (List<Node> nodesForClusters, Node \_depot) | Генерація кластерів випадковим чином (специфічна для нащадків) |
| protected static void MoveNodeFromOneClusterToAnother (Cluster cluster1, Cluster cluster2) | Перемістити вузол з одного кластеру в інший |
| public static void MoveNodeFromOneClusterToAnother (List<Cluster> clusters) | Перемістити вузол з одного кластеру у колекції до іншого |
| SiteClusteringCvrp – клас-нащадок класу SiteClusteringVrp. Клас розв’язку задачі кластеризаціїї для CVRP | private void AddCluster() | Додати пустий кластер до списку кластерів |
| protected static bool ExchangeNodesInClusters (Cluster c1, Cluster c2) | Перемістити вузол з одного кластеру в другий і з другого в перший |
| public static bool ExchangeNodesInClusters (List<Cluster> clusters) | Перемістити вузли між двома кластерами у списку |
| protected override List<Node> GenerateClusters (List<Node> nodesForClusters, Node \_depot) | Генерація кластерів випадковим чином |
| protected override Site GetNeighbour() | Згенерувати сусідній розв’язок |
| SiteClusteringCvrpp – клас-нащадок класу SiteClusteringCvrp. Клас розв’язку задачі кластеризації для CVRPP | private void FormAllClusters() | Формування списку усіх кластерів |
| protected override Site GetNeighbour() | Згенерувати сусідній розв’язок |
| public static double GetPrice (double length, double kilometerCost, double fines) | Функція повертає ціну перевезення з урахуванням штрафів |
| protected override void GoToNeighbour(Site site) | Перейти до сусіднього розв’язку |
| public override List<Node> PrepareToDraw (Color connectionsColor) | Підготувати вузли до графічного відображення |
| SiteClusteringCvrppNnc – клас-нащадок класу SiteClusteringCvrpp. Клас розв’язку задачі кластеризації алгоритмом Bees CVRPP+NNC | public override void StartingInitialize() | Функція, що виконується на початку бджолиного алгоритму. Обчислення початкового набору кластерів, використовуючи алгоритм near-neighbor chain |
| private void GenerateClustersFromNnc() | Генерація кластерів розв’язку на основі початкового набору кластерів, обчислених алгоритмом near-neighbor chain |
| Node – клас вузла. Зберігає інформацію, що стосується вузла | public void ConnectTo (Node node) | Приєднати поточний вузол до іншого вузла |
| public void ConnectTo (Node node, Color color) | Приєднати поточний вузол до іншого вузла, враховуючи колір з’єднання |
| public void DisconnectFromAll() | Від’єднати поточний вузол від усіх інших вузлів |
| ClusteringAlgorithm – абстрактний клас-нащадок класу Algorithm. Реалізує спільну логіку для алгоритмів кластеризації | protected void AddCluster() | Додати пустий кластер до списку кластерів |
| public override void DrawNodes() | Графічне відображення вузлів, оброблених алгоритмом |
| KMeans – клас-нащадок класу ClusteringAlgorithm. Клас реалізує логіку алгоритму кластеризації k‑means | private void GenerateClusters() | Генерація початкового набору кластерів |
| protected override void InnerIteration() | Ітерація алгоритму |
| private void SetCenters() | Перемістити центри кластерів відповідно до алгоритму |
| NearestNeighbourChain – клас-нащадок класу ClusteringAlgorithm. Клас реалізує логіку алгоритму кластеризації nearest-neighbor chain | private void GenerateClusters() | Генерація початкового набору кластерів |
| public override string Info() | Функція повертає рядок з необхідною інформацією щодо поточного стану алгоритму |
| protected override void InnerIteration() | Ітерація алгоритму |
| private int NearestCluster (int cluster) | Отримати номер найближчого кластера |
| Cluster – клас кластера. Зберігає інформацію, що стосується кластера | public void AddNode (Node node) | Додати вузол до кластера |
| public void AddNodes (List<Node> nodes) | Додати колекцію вузлів до кластера |
| public List<Node> GetDrawingNodes (Color connectionsColor) | Отримати вузли кластера, з’єднані відповідним кольором |
| public double GetPrice() | Обчислення суми квадратичних відстаней вузлів кластера від його центру |
| public void Merge (Cluster cluster) | Об’єднання кластера із поточним |
| public void RemoveNode (Node node) | Видалити вузол із кластера |
| public void RemoveNodes() | Видалити усі вузли з кластера |
| TaskController – статичний клас, функціями якого є зв’язок користувацького інтерфейсу з класами алгоритмів, графічне відображення вузлів, збереження усіх вузлів, робота з файлами | public static void AddNodeAtScreen (Node.NodeType nodeType, int posX, int posY, int volume = 1, int fine = 0) | Додавання нового вузла і відображення його на екрані |
| private static bool CheckVolume() | Перевірка розміру вузла |
| public static void CreateNewModel() | Створити нову модель (видалити усі старі вузли) |
| private static void DrawNode (Graphics g, Node node) | Графічне відображення вузла на відповідний графічний об’єкт |
| private static void DrawNodes (Graphics g, List<Node> nodes) | Графічне відображення вузлів на певний графічний об’єкт |
| public static void DrawNodes (int index) | Графічно відобразити збережену множину вузлів |
| public static void DrawNodes (List<Node> nodes = null) | Графічне відображення множини вузлів |
| public static Color GetDrawingColor (int i) | Функція повертає колір в залежності від індексу |
| public static bool LoadModel(string fileName) | Завантаження множини вузлів з файлу |
| public static bool SaveModel (string fileName = "") | Збереження множини вузлів до файлу |
| public static void SaveDrawnNodes() | Зберегти множину поточну вузлів для подальшого графічного відображення |
| private static void SetAlgorithm (Algorithm algorithm) | Встановлення поточного алгоритму |
| public static void StartBeesAlgorithm (BeesColony.ProblemType problemType, int clustersCount, int scoutsCount,  int goodSitesCount, int bestSitesCount, int neighboursForGoodSites, int neighboursForBestSites, int clusterCapacityLimit = -1, double kilometerCost = 0) | Запустити алгоритм бджолиних колоній з відповідними коефіцієнтами |
| public static void StartClusteringToTsp() | Запустити вирішення задачі комівояжера для кожного кластеру |
| public static void StartKMeansAlgorithm (int clustersCount) | Запустити алгоритм k‑means |
| public static void StartNearestNeighbourChainAlgorithm (int capacityLimit) | Запустити алгоритм nearest-neighbor chain |
| public static void StopAlgorithm() | Зупинити алгоритм |
| ClusteringToTsp – клас-нащадок класу Algorithm. Виконує вирішення задачі комівояжера для кожного заданого кластеру | public void Calculate (int scoutsCount, int goodSitesCount, int bestSitesCount, int neighboursForGoodSites, int neighboursForBestSites) | Виконати обчислення |
| public override void DrawNodes() | Відобразити графічно вузли задачі |
| SerializationData – клас, що тимчасово зберігає інформацію про вузли, а також записує і зчитує її з файлу | public static bool Serialize(string fileName, SerializationData data) | Серіалізувати інформацію в файл |
| public static bool Deserialize(string fileName, out SerializationData data) | Десеріалізувати інформацію з файлу |
|  |  |  |

## **Опис звітів**

За допомогою даного програмного продукту можливо генерувати звіти із планом перевезень продукції, який містить маршрути для кожного транспортного засобу, їх довжину, витрати на перевезення, об’єми перевезення по транспортному засобу і сумарні штрафи.

Звіт формується автоматично. Для цього необхідно обрати пункт меню “Reports – Generate report”. Після цього на екран буде виведено вікно, яке буде містити текстовий звіт щодо складених маршрутів перевезень.

Звіт можна зберегти у файл обравши пункт меню “File – Save report…”.

На рисунку 4. зображено сформований звіт.

Згенерований звіт має вигляд зображений на рисунку 4.6, де показано …

<Тут має бути зразок згенерованого звіту>

Рисунок 4.6 – Вигляд звіту «План виробництва»

## **Висновок до розділу**

*Дати висновок до цього розділу.*