Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем та програмної інженерії
(назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи)

магістр
(освітній ступінь (освітньо-кваліфікаційний рівень))

на тему: **Розробка інформаційної системи для організації логістичних перевезень з використанням клієнт-серверної архітектури**

Виконав: студент (ка) 6 курсу, групи СПм-61
спеціальності (напряму підготовки)
121 інженерія програмного забезпечення
(шифр і назва спеціальності (напряму підготовки))

	(підпис)	Корнієнко В.О. (прізвище та ініціали)
Керівник	(підпис)	(прізвище та ініціали)
Нормоконтроль	(підпис)	(прізвище та ініціали)
Рецензент	(підпис)	(прізвище та ініціали)

м. Тернопіль – 2019

Анотація

Робота містить ,сторінок , таблиць і рисунка , список літератури з найменувань та додатки.

У наш час актуальною постає тема транспортних перевезень. Логістика займає визначне місце серед усіх інших галузей і для її використання впроваджується усе більше програмних технологій та методів. Разом з тим варто зазначити що велика кількість транспортних компаній перебувають на рівні адаптації програмних продуктів в умовах складного та високонавантаженого ринку. В той час коли засоби програмування та інформаційні технології дозволяють використовувати значну кількість ресурсу для побудови схем руху і швидкодії автоматизованих систем у цілях збагачення і повного використання потенціалів обчислювальної техніки. Судячи з статистичних даних логістика займає одну з ключових ролей у формуванні ВВП України та світу, що означає високий попит на якісний програмний продукт у галузі.

Об’єктом дослідження є збільшення ефективності документообігу та показників швидкодії і координації між водієм та оператором сервісу вантажних перевезень з використанням сучасних технологій для веб-застосунків.

Метою роботи є реалізація архітектури, що може злагоджено виконувати функції та стане доповнення до вже існуючих технічних засобів логістичних програмних засобів , побудованих професійними командами розробників і надасть зручні інструменти для роботи оператора вантажних перевезень.

КЛЮЧОВІ СЛОВА: ПРОГРАМНА СИСТЕМ, ЛОГІСТИКИ, ТРАНСПОРТНІ ПЕРЕВЕЗЕННЯ, АВТОМАТИЗАЦІЯ

Abstract

The work contains pages , tables and figures , a list of references and titles .

Nowadays, the topic of transportation is becoming relevant. Logistics is a prominent place in all other industries and more and more software technologies and methods are being introduced to use it. However, it should be noted that a large number of transport companies are at the level of adaptation of software products in a complex and highly charged market.While software and information technology allow you to use a large amount of resources to build circuits and speed automated systems for the enrichment and full use of the potential of computing. According to statistics, logistics plays a key role in shaping the GDP of Ukraine and the world, which means high demand for quality software in the industry.

The object of the study is to increase the efficiency of document flow and performance and coordination between the driver and the freight carrier using modern web application technologies.

The purpose of the work is to implement an architecture that can perform functions in a coherent manner and complement the existing logistical software tools built by professional development teams and provide convenient tools for the freight carrier.

KEYWORDS: SOFTWARE, LOGISTICS, TRANSPORTATION, AUTOMATION

ЗМІСТ

ВСТУП

Бурхливий розвиток галузі вантажних перевезень є головною причиною формування запиту у сфері технічних засобів та комунікацій для створення відповідних програмних рішень і формуванні у суспільстві чіткого розуміння всієї важливості та актуальності побудови схем маршрутного планування і діяльності забезпечення розвитку руху комунікацій між постачальниками логістичних послуг та вантажними перевізниками, які працюють в умовах високої навантаженості і загальної напруги. На водія вантажних перевезень лягає непроста ноша здійснення комунікацій та власне самого перевезення і для більш злагодженої координації у сфері використовуються мобільні групи операторів, які забезпечують надійний тил для подальшої роботи групи.

Цілком природно, що в таких умовах виникає потреба в надійних і високопродуктивних інструментах з допомогою яких оператор вантажних перевезень зможе швидко та якісно виконувати свої завдання, які допоможуть координувати подальші дії водія вантажних автомобілів.

Питання якісної побудови маршрутних карт та фіксації часових швидкостей є ключовим не тільки при розумінні важливості розсередження та розподлі навантаженостей між різними учасниками процесу формування і налагодження орієнтації в полі часового планування та оптимізації графічних частин устаткування на картах і в графіках водіїв.

Модель структури матиме графічне представлення на основі уже готових архітектурних рішень та з використанням додаткових налагоджених механізмів впливу і ототожнення системи з високофункціональним механізмом робочого процесу з вишуканим ідентифікатором примусових і доцільних рішень , що забезпечуватимуть онлайн підтримку і безпосередню робочу структуру механізму з

визначеними навантаженнями , які можна оцінити як реалізовані виробничим процесом стандарти в тій чи іншій мірі сформовані потребами галузевого розширення і можливостями сучасної обчислювальної техніки. Глбиокий аналіз та вузькопрофільні спеціалізації програмних рішень , готових до впровадження, показали що значних змін потребують наявні продукти. Видно , що галузь програмування стала недосяжним для розуміння сучасних підприємців галузі логістики, які більше зосереджені на організації і впроваджені самих перевезень і які стикнулися зі складністю загальної картини і поєднані напрацювань сучасних методів інженерії програмного забезпечення з вимогами і складними потребами галузі вантажних перевезень. При всій досвідченості та освідомленості окремих власників фірм , які займаються організацією транспорту для вантажних перевезень їм стає на заваді обмеженості в належних фінансових ресурсах , які необхідні для розробки якісних інструментів та впровадження цих технологій в роботі. Разом з тим багато груп програмістів , які займаються розробкою подібного роду програмного забезпечення стикаються з проблемою недостатнього розуміння галузі та вузьких часових рамок впровадження , що призводить до діяльності в режимі швидких розробок і правок і веде до невідповідності потреб ринку та інструментів , що розробники таких продуктів реалізують для роботи працівників галузі.

Важливим нововведенням стануть архітектурні рішення пов’язані з оптимізацією часу графіку руху грузового транспорту і відображення поставлених пунктів за допомогою картографічних засобів програмного забезпечення.

На сьогодні існує безліч програмних рішень , які в тій чи іншій мірі виконують покладені на них зобов’язання, проте під час аналізу жодне з них не відзначилось високою продуктивністю , стабільністю та швидкодією, і саме з цієї причини було обрано галузь вантажної логістики для розробки програмного забезпечення і написання дипломної роботи.

Елементом новизни у даній роботі є оптимізація маршрутів пошуку та графіків руху водіїв. Сьогодні фірми , які займаються такою діяльністю використовують для

роботи спеціально навчених працівників , які в ручному режимі оптимізовують графіки руху водіїв. Ця робота є новим програмним рішенням для затратної та рутинної роботи операторів, з якою буде краще справлятися автоматизована система , при цьому заощаджуючи організаторам та перевізникам час , гроші і нерви.

Результатом роботи є веб-розширення з допомогою якого можна експортувати та трансформувати вхідні дані і будувати нові графіки маршрутних карт для тотожного розуміння і остаточного розрахунку найкращих типів дорожніх сполучень, типів вантажу, місць загрузки та вивантаження і побудови оптимальних сполучень.

1. АНАЛІЗ ВИМОГ ТА ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Широке використання інтернету в різноманітних сферах призвело до того, що об’єми даних почали стрімко рости, після чого виникло створення технологій великих даних. До великих даних можна віднести: інформаційні активи, великі в об’ємі, швидкості приросту та різноманітності, що включають в себе інноваційні та економічно ефективні методи обробки інформації, які сприяють покращенню процесу прийняття рішень та його оптимізації. На даний момент розвиток бізнесу сприяє пришвидшенню розвитку технологій управління даними . Фірми, що використовують технологію баз даних можуть спостерігати її чималі переваги .

Під великим даними часто розуміють соціально-економічний феномен, що утворився завдяки можливості обробляти та аналізувати велику кількість даних.

Існує так званий “принцип трьох v”, що визначає основні характеристики великих даних. Перше “v” це об’єм (volume) даних. Наступне “v ” це швидкість (velocity). Під цим також мжна розуміти велику швидкість приросту даних, так і вимоги до високошвидкісної обробки. Останнє “v ” це різноманітність (variety) цих даних і вимоги до їх своєчасної обробки.

З точки зору інформаційних технологій в різноманіття підходів та інструментів включались засоби масово-паралельної обробки невідомих структурованих даних, а саме – системи управління базами даних категорії NoSQL, алгоритмами Map Reduce та реалізованих ними програмними каркасами та бібліотека Hadoop. В подальшому до серії технологій великих даних стали відносити різноманітні інформаційно-технологічні рішення, що забезпечують в тому чи іншому ступені схожі по характеристиках можливості по обробці надвеликих масивів даних.

Джерелами великих даних включають в себе технології інтернет речей, соціальні медіа та внутрішні дані фірм інших різносторонніх організацій.

З розвитком технологій БД, вони почали охоплювати все більше сфер життєдіяльності та послідовно все більше компаній. Не обійшли стороною вони і область логістики, в якій вони мають шанс широкого та глибокого застосування.

Постачальники послуг вже збирають дані з різних джерел, включаючи походження товару, пункт прибуття, розмір, вага і місце перебування. Але однієї інформації недостатньо - необхідне вміння витягти з неї певні закономірності, ідеї.

Аналітика БД надає інформацію про моделі поведінки покупців, ринкові тенденції, циклах технічного обслуговування. Вона також пропонує методи скорочення витрат, оптимальну цінову стратегію, стратегію з оптимізації процесів і значно полегшує процес прийняття рішень .

Можна виділити три сфери діяльності логістики, розвитку яких сприяє технологія БД:

- Збільшення ефективності
- Покращення якості обслуговування клієнтів
- Реалізація ефективної або нової бізнес-моделі

Самий прямолінійний спосіб застосування БД в бізнесі - підвищення рівня ефективності операцій, особливо «останньої милі» Перешкодою в досягненні високої ефективності часто стає так звана «остання миля». Кінцевий етап ланцюжка поставок часто виявляється найбільш витратним, тому логістичні системи загострюють увагу на цьому питанні. Відома служба доставки DHL розбила рішення цієї проблеми на дві стадії. Перша - оптимізація маршруту руху транспорту в режимі реального часу. Друга має на увазі нову модель доставок, в якій використовують людей, які не працюють в компанії, але пересуваються за потрібне для компанії маршрутом. Це здійснюється за допомогою таких технологій БД, як обробка складних подій і геокорреляція через спеціальний додаток. Це рішення виявляється набагато ефективніше планованої і розподіленої робочої сили.

Логістика вже довгий час є індикатором макроекономічної ситуації. Транспортування товарів по всьому світу служить важливим фактором розвитку економіки. Типи і обсяги товарів, що доставляються в різні регіони, дозволяють визначити їх споживчий попит. Отримуючи і аналізуючи дані про глобальні поставки, логістичні компанії можуть отримати велику і детальну інформацію, необхідну для визначення найбільш ефективної бізнес-моделі.

Якими ж системами здійснюються технології БД в логістиці? Для цих цілей служать системи управління ланцюгами поставок (SCM) в інтеграції з системами управління взаємовідносинами з клієнтами (CRM). У ЦИХ системах здійснюється управління такими даними, як транзакції клієнтів, інвентаризація, реклама, відносини з клієнтами, їх переваги, інфраструктура менеджменту продаж, дані з фінансів і т.д. Як джерело даних використовуються системи планування ресурсів підприємства (ERP).

SCM використовують кількісні та якісні методи для прогнозування різних наслідків і вирішення проблем ланцюжків поставок. Використання даних GPS і радіочастотної ідентифікації (RFID), сенсорів на обладнанні зберігання допомагають точно відстежити статус товару. Один з перших користувачів SCM з БД - мережа супермаркетів Walmart. SCM вирішила багато проблем підприємства, особливо інвентаризацію, дозволяючи бачити точне число товарів на полицях магазинів в певний момент часу.

Розглянемо деякі можливості БД, що використовуються в логістиці:

- Аналіз обсягів поставок

Можливість прогнозу обсягів поставок в певний день тижня, місяця, року необхідна для оптимального розподілу бюджету. Аналіз існуючих даних допоможе спрогнозувати піки обсягів продукції і згенерувати рекомендації щодо оптимізації цих процесів.

- Дані про товари з особливими умовами зберігання і перевезення

Деякі товари вимагають особливих умов зберігання. Наприклад, деякі продукти і ліки необхідно зберігати при певній температурі, тендітні предмети вимагають особливих умов транспортування. Перевезення таких товарів в підсумку виявляється дорогою як для логістичної компанії, так і для клієнта, особливо в разі, коли з-за невірних умов зберігання і транспортування товар виявляється зіпсований. Тут допомагає технологія інтернету речей: сенсори сканують температурні умови, рівень запасів і т.д., дані аналізуються в оффлайн-режимі, формуючи інформацію про найбезпечніших і економічних способах транспортування і розміщення товарів.

- Економічний маршрут

Аналіз БД виявить найбільш надійні та економічні маршрути. Вихідна інформація допоможе у виборі кращих авіаліній або складських компаній.

- Аналіз ризику

Оцінка ризику значно допоможе скоротити витрати компанії. БД прогнозують різні непередбачені ситуації і пропонують альтернативні стратегії, які допомагають їх уникнути або способи мінімізації втрат.

- Аналіз часу доставки

Клієнтам завжди важливо знати точну дату поставки, що дуже складно спрогнозувати. Час доставки залежить від різних чинників: кількість замовлень, товарів, ситуація на дорозі, стан транспорту і т.п. За допомогою аналізу повного обсягу цих даних можна припустити приблизний час доставки. Також аналіз даних в режимі реального часу надасть точну інформацію про поточний перебування товару.

- Веб аналіз

Аналіз БД допоможе в наданні більш персоналізованого каталогу послуг відвідувачам веб-сайту. Залежно від інтересів користувача, йому можуть бути автоматично запропоновані різні послуги, кампанії, акції.Використання БД має

великі переваги, проте існують також і складності, пов'язані в першу чергу зі складністю отримання інформації, її зберіганням, пошуком, аналізом і візулізацією. Одне з найскладніших до подолання перешкод - недостатня швидкість ресурсів. Сучасні комп'ютери розвиваються досить швидко, проте кількість інформації, необхідної для обробки, зростає набагато швидше. Це обмежує обробку даних в режимі реального часу. Наступна проблема - суперечливість, неповнота і несвоєчасність даних. Отже, дані необхідно підготувати до обробки деякими процесами.

Такі процеси як чистка, об'єднання в кластери, перетворення в певний формат допоможуть позбутися від похибок і всілякого «сміття». З появою технологій БД почали змінюватися і технології збору та зберігання даних, включаючи пристрої зберігання даних, їх архітектуру, механізми доступу.

На даний момент БД ще знаходяться на початковій стадії свого розвитку, отже, поки що не здатні відповідати всім вимогам компаній. Для більш широкого поширення БД в логістиці потрібне їх подальший розвиток і подолання проблем, пов'язаних з якісними характеристиками вхідних даних. Доцільно припускати, що в майбутньому БД стануть часто використовуватися в логістичних системах.

1.2 Постановка задачі

Після глибокого аналізу предметної області було вирішено зробити системний аналіз з розробкою програмної системи, яка полягає у виконані основних функцій, а саме : планування маршруту подорожі водія, оптимізація графіку руху вантажоперевізника, позначення за допомогою картографічних схем шляху початку і кінця руху водія та визначення вартості пересування. Програмний додаток складається з таких функціональних складових:

- вхід в систему
- реєстрація нового користувача

- можливість додавання нового проекту
- можливість запису нових даних
- введення відомостей про замовника
- зміна та уточнення відомостей про замовника
- етап попередніх робіт
- можливість завантажити готові дані
- можливість пошуку найкращих пропозицій перевезень
- можливість перегляду даних про водія
- можливість зміни статусу водія
- можливість редагування проекту
- можливість видалення проекту

1.3 Пошук акторів та варіантів використання

Під час моделювання програмної системи було обрано засоби мови UML як основної для розробки проектного застосування і визначення основних проектних рішень з урахуванням архітектурних можливостей сучасних програмних систем. Такий архітектурний підхід дозволить мати чіткі уявлення про можливості продукту і визначить усі межі використання в рамках програмної системи.

В мові UML більшість уявлень про моделі складних систем відображаються у вигляді спеціальних графічних конструкцій, які отримали назву діаграм прецедентів.

Уніфікована мова моделювання (UML) - мова моделювання загального призначення. Основна мета UML - визначити стандартний спосіб візуалізації способу, який проектувала система. Він досить схожий на креслення, які використовуються в інших галузях техніки.

UML - це не мова програмування, це швидше візуальна мова. Ми використовуємо діаграми UML, щоб зобразити поведінку та структуру системи. UML допомагає інженерам-програмістам, бізнесменам та системним архітекторам у моделюванні, дизайні та аналізі. Група управління об'єктами (OMG) прийняла єдину мову моделювання в якості стандарту в 1997 році. Їм до цього часу керує OMG. Міжнародна організація зі стандартизації (ISO) опублікувала UML як затверджений стандарт у 2005 році. UML переглядався протягом багатьох років і періодично переглядається.

UML пов'язаний з об'єктно-орієнтованим дизайном та аналізом. UML використовує елементи та формує асоціації між ними для формування діаграм.

Структурні діаграми - Захоплення статичних аспектів або структури системи. Структурні діаграми включають: діаграми компонентів, діаграми об'єктів, діаграми класів та діаграми розгортання.

Діаграми поведінки – фіксує динамічні аспекти або поведінку системи. Діаграми поведінки включають: діаграми випадків, діаграми стану, діаграми діяльності та діаграми взаємодії.

Об'єктно-орієнтовані поняття, що використовуються в UML:

1. Клас - Клас визначає синій друк, тобто структуру та функції об'єкта.
2. Об'єкти - Об'єкти допомагають нам розкласти великі системи та допомагають модулювати нашу систему. Модульність допомагає розділити нашу систему на зрозумілі компоненти, щоб ми могли будувати нашу систему по частинах. Об'єкт - це фундаментальна одиниця (будівельний блок) системи, яка використовується для зображення сутності.

3. Спадщина - Спадщина - це механізм, за допомогою якого дочірні класи успадковують властивості своїх батьківських класів.

4. Абстракція - механізм, за допомогою якого деталі реалізації приховані від користувача.

5. Інкапсуляція - Зв'язування даних разом та захист їх від зовнішнього світу називається інкапсуляцією.

6. Поліморфізм - механізм, за допомогою якого функції чи утворення здатні існувати в різних формах.

Діаграми варіантів використання використовуються для відображення функціональності системи або частини системи. Вони широко використовуються для ілюстрації функціональних вимог системи та її взаємодії із зовнішніми агентами (суб'єктами). Випадок використання - це в основному схема, що представляє різні сценарії, де система може бути використана. Приклад діаграми використання дає нам уявлення про високий рівень того, що робить система чи частина системи, не вдаючись до деталей щодо впровадження.

Призначенням діаграм варіантів використання є захоплення основних функцій системи та візуалізація взаємодії різних речей, що називаються дійовими особами, із випадком використання. Це загальне використання діаграми випадку використання.

Діаграми випадку використання представляють основні частини системи та робочий процес між ними. У випадку використання деталі реалізації приховані від зовнішнього використання, лише представлений потік подій. За допомогою використаних діаграм на прикладі ми можемо з’ясувати умови до та після публікації після взаємодії з актором. Ці умови можна визначити, використовуючи різні тестові випадки.

У загальному випадку ці діаграми використовуються для:

- Аналіз вимог системи

- Проектування візуального програмного забезпечення високого рівня
- Захоплення функціональних можливостей системи
- Моделювання основної ідеї системи
- Передня та зворотна інженерія системи з використанням різних тестових випадків.

Діаграми варіантів використання потрібні для передачі потрібної функціональності, тому точний обсяг випадку використання може змінюватися залежно від системи та мети створення моделі UML.

Перед визначенням та проектуванням діаграми системи використання необхідно визначити такі компоненти:

- функціональні особливості і характеристики системи, які мають бути представлені у вигляді варіантів використання
- власне актори або прецеденти
- відношення між акторами в системі та варіантами їх використання

Актор використовується всередині таких діаграм. Актор - це сутність, яка взаємодіє із системою. Користувач - найкращий приклад актора. Актор - це суб'єкт, який ініціює випадок використання поза межами випадку використання. Це може бути будь-який елемент, який може викликати взаємодію із випадком використання. Один актор може бути пов’язаний із кількома випадками використання в системі.

За допомогою засобів представлення мовою UML визначено такі види відношення між представленими сутностями акторів та варіантами їх використання, а саме :

Відношення залежності - таке відношення між сутностями, при якому зміна стану першої (незалежної) впливає на стан іншої (залежної) сутності.

Відношення асоціації - структурне відношення, що описує зв'язок, тобто з'єднання між об'єктами (з різними типами зв'язків). На основі асоціацій виконується навігація по елементах відношення.

Відношення узагальнення (агрегування) - відношення показує що один елемент (дочірній, частина) є структурною частиною іншого (батьківського, цілого) .Можливо композитне агрегування. Сутність є частина цілого і має з ним єдиний час життя.

Відношення реалізації - відношення показує що один елемент визначає контракт (зобов'язання), а інший виконує цей контракт.

Для розробленої системи головним актором є користувач. Можливості його впливу на систему можна відобразити як діаграму варіантів використання. Дана система включає як і вільне так і локальне користування окремою фірмою чи її частиною, тому архітектурні особливості та варіанти використання підібрані належним чином. Згідно вимог додаток реалізується як веб ресурс з можливістю доступу до бази даних з замовленнями та користувачами, що спростить доступ до інформації та пришвидшить процес пошуку замовлень. В даній системі планується взаємодія між акторами, тому важливо зазначити, що потрібно правильно визначити акторів та їх ролі щоб методи взаємодії між ними були коректними. Система зосереджена в основному на логістичних компаніях, а це означає, що важливо врахувати особливості галузі логістики для правильного проектування діаграми варіантів використання та побудови сценаріїв їхньої взаємодії з системою.

В роботі з програмою користувачі поділятимуться на авторизованих та гостей. Авторизований користувач буде мати свій обліковий запис, для того щоб попередньо зареєструватись в системі. Також варто зазначити що у нього буде певна кількість компаній, з якими він і буде взаємодіяти. Гість буде відправлений на сторінку авторизації з можливістю зареєструватись.

1.4 Опис ключових варіантів використання

Як було розглянуто вище в даній системі є два актори виконавці, ними будуть Гість та Користувач. Можливості актора Гість будуть з обмеженими привілеями та функціональними можливостями, чого не скажеш про Користувача. Гість не зможе редагувати замовлення,а тільки переглядати їх та подавати заявки або пропозиції на створення нових замовлень. При запуску програми буде відбуватись визначення користувача та відповідна класифікація до певного варіанту використання, а саме до неавторизованого чи авторизованого користувача.

Для отримання більш чіткої оцінки корисності кожного з акторів, був складений план випадків використання із зображенням їх та можливостей доступних кожному з них.

На рисунку 1.1 зображено діаграму варіантів цих акторів.

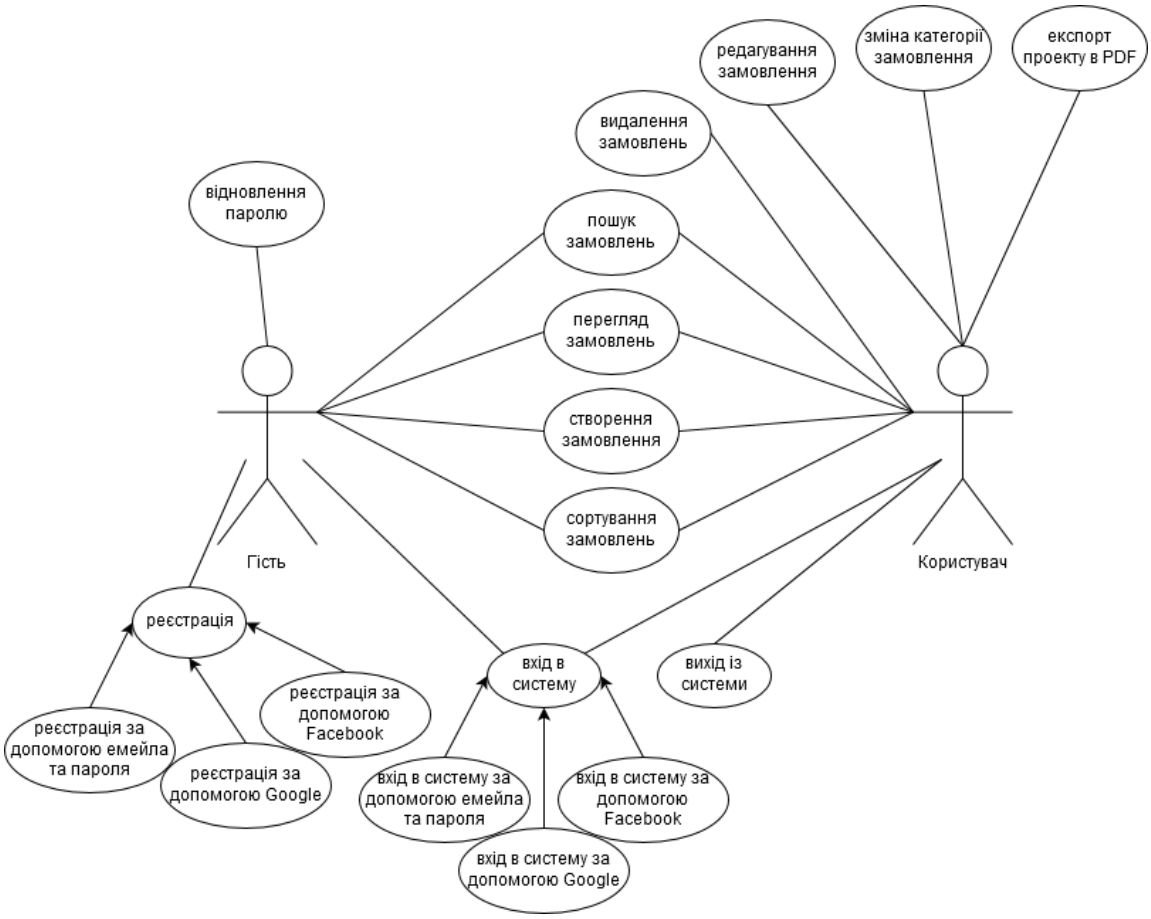

Рис 1.1

З детальним описом варіантів використання можна ознайомитись в таблиці 1.2

Кожне з полів "Логін" та "Вхід" можуть використовувати міркування про різні варіанти використання, які дозволять індивідуальними варіантам підтвердити специфіку їх механізмів. Деякі з варіантів використання доступні лише одному з акторів, це дозволяє нормувати процес доступу до замовлень та виключає непотрібну інформацію для кожного типу.

Таблиця 1.2 – Опис ключових варіантів використання системи

Код	Найменування	Основний сценарій	Альтернативний сценарій
НА1	Реєстрація за допомогою email та пароля	1. Гість переходить на екран реєстрації. 2. Заповнює поля email, password, repeat password, country, zip code 3. Натискає на кнопку “Sign up” 4. Гість авторизується в системі	1. Гість не ввів валідні дані. Виводиться повідомлення про те, що дані не валідні 2. Користувач з таким email вже зареєстрований у системі 3. На екрані з’являється вікно з помилкою
НА2	Реєстрація за допомогою Google	1. Гість переходить на екран реєстрації 2. Натискає кнопку “Sign up with Google” 3. З’являється вікно авторизації через сервіс Google 4. Гість авторизується в	1. Гість відхилив запит. Виводиться повідомлення про те, користувач відхилив запит на вхід

		системі	
НА3	Вхід в систему	1. Гість переходить на екран авторизації 2.Вводить свої дані та нажимає кнопку “Sign in” 3.Гість авторизується в системі	1. Введені дані не є валідними. Виводиться повідомлення про те, що дані не валідні
НА4	Сортування замовлень	1.Користувач вибирає в списку замовлень категорію по якій вони будуть сортуватись 2.Список замовлень сортується залежно від вказаних параметрів 3.При потребі можна змінити порядок сортування	
НА5	Створення замовлень	1.Користувач натискає кнопку “Створити замовлення” 2.Заповнює відповідні поля 3.Перевіряє надану інформацію 4.Підтверджує створення замовлення	1. Не всі поля є заповненими, програма просить користувача перевірити ще раз необхідні поля 2. Введені дані не є валідними. Виводиться повідомлення про те, що дані не валідні
НА6	Перегляд замовлень	1. Користувач може переглядати список наявних замовлень 2. Якщо потрібно він може переглянути	

		замовлення детально	
НА7	Пошук замовлень	1. Користувач може шукати замовлення в списку замовлень за допомогою “ключового слова” чи по певній категорії 2. Також пошук можна здійснювати залежно від певних параметрів замовлення	1. Якщо результат пошуку негативний, то виводиться повідомлення “Нічого не знайдено”
НА8	Редагування замовлень	1. Користувач вибирає замовлення із списку та нажимає кнопку “Редагувати” 2. Після чого вводить нові дані або змінює попередні, після чого натискає кнопку “Зберегти”	1. Не всі поля є заповненими, програма просить користувача перевірити ще раз необхідні поля 2. Введені дані не є валідними. Виводиться повідомлення про те, що дані не валідні
НА9	Експорт проекту в формат PDF.	1. Користувач переходить на екран експорту 2. Вводить email адресу, на яку буде надіслано результат експорту 3. Нажимає кнопку “Надіслати” 4. Проект експортується і відсилається	1. Користувач ввів невалідну email адресу. Виводиться повідомлення про те, що email адреса невалідна 2. Користувач не ввів email адресу. Виводиться повідомлення про те, що потрібно вказати email адресу

		системою на вказану email адресу в форматі PDF	
НА10	Вихід із системи	1. Користувач натискає на кнопку виходу з системи 2. Спрацьовує переадресація на екран входу у систему 3. При повторному запуску програми користувача системи буде ідентифіковано як Гостя	

З наведеної вище інформації у діаграм варіантів використання та таблиці можна детально ознайомитися із функціоналом системи та розподілом ролей між акторами і варіантами використання.

2. ПРОЕКТУВАННЯ ПРОГРМНОЇ СИСТЕМИ

2.1 Вибір процесу розробки

Управління розробкою програмного забезпечення — це тривале мистецтво розробки, керування та планування програмними проектами із проектуванням, різновид управління проектом, який включає розробку, реалізацію, відстеження та контроль за проектами, що включають розробку програмного забезпечення. Ці методи удосконалення включають використання різноманітних методик для втілення окремих підходів. Для розробки даного проекту було обрано спіральну модель розробки.

Спіральна модель розробки програмного забезпечення не так широко відома, як, наприклад, Scrum або Kanban. Причина в тому, що даний підхід може виявитися досить витратним в застосуванні. Саме тому він не дуже добре підходить для невеликих проектів. У спіральної моделі особлива увага приділяється управлінню ризиками. На практиці це означає, що фаза оцінки та дозволу ризиків є критичною для успіху проекту. Контроль ризиків, в свою чергу, вимагає проведення специфічного аналізу на кожній ітерації. Для регулярного огляду і аналізу поточного стану проекту необхідні додаткові навички та ресурси.

На перший погляд може здатися, що дана модель є складною, неповороткою і дорогої і немає ніяких вагомих причин для того, щоб розглядати її як один з можливих варіантів. Але, як і будь-який інший підхід до розробки програмного забезпечення, спіральна модель має, крім недоліків, також і свої сильні сторони. Наприклад, вона дозволяє додавати додатковий функціонал до програмного забезпечення на самих пізніх стадіях розробки. Оскільки постійний контроль за ризиками і, як наслідок, регулярні експертизи поточного стану проекту, є невід'ємною частиною даного підходу, загальне бачення проекту стає більш ясним.

Коротко спіральну модель можна описати як повторювану послідовність циклів розробки з безперервним контролем ризиків. ЇЇ зображено на рисунку 2.1

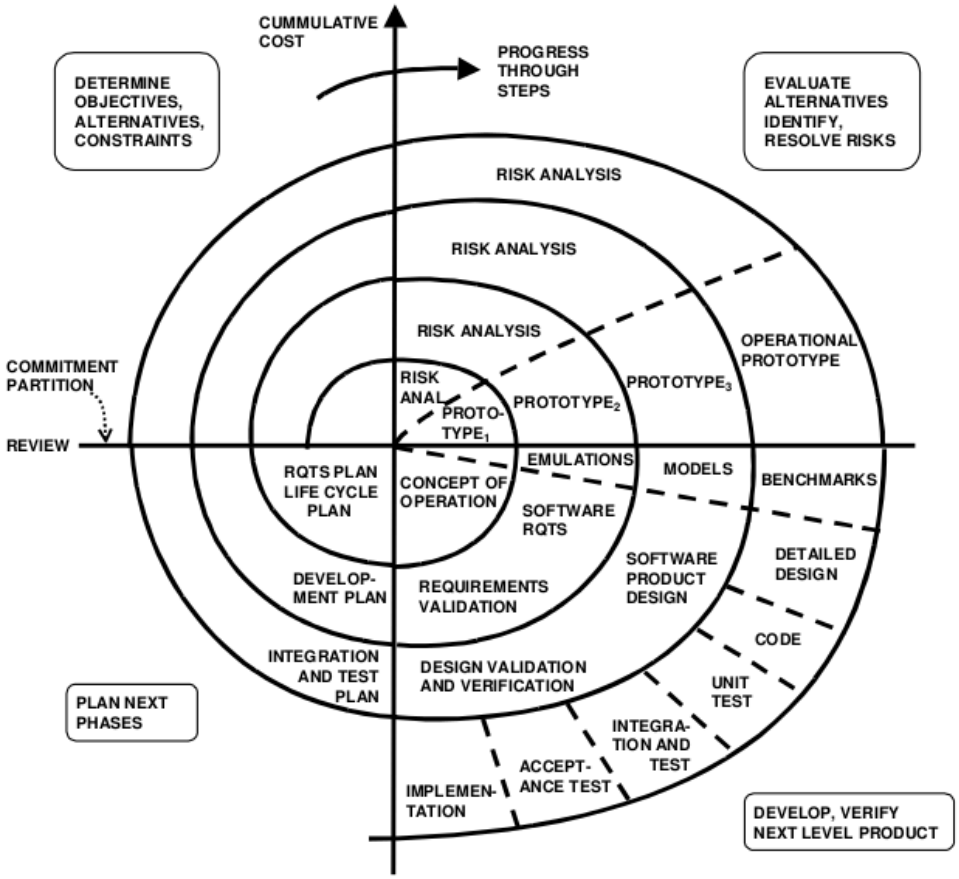

Рисунок 2.1 Спіральна модель розробки

Вона складається з чотирьох головних повторюваних стадій. В ході процесу розробки проект кілька разів проходить через всі ці фази. Кожна така ітерація називається спіраллю.

Чотири головні фази це:

1. Визначення цілей, альтернатив, обмежень, або фаза планування. З цієї стадії починається робота над проектом. Команда розробників формулює цілі проекту, основні вимоги (такі як, наприклад, Business Requirement Specifications, або BRS, System Requirement Specifications, або SRS), можливий дизайн і т.д. На наступних спіралях вимоги формуються відповідно до відгуками, отриманими від замовника. Саме тому постійна комунікація між замовником і командою вкрай важлива.

2. Аналіз, визначення і дозвіл ризиків є однією з найбільш значущих стадій розробки. В даному контексті, ризики - це можливі події і стани проекту, що перешкоджають досягненню командою розробників поставлених цілей. Існує досить великий діапазон можливих ризиків, від тривіальних і легко подоланні, до вкрай серйозних. Головним завданням для команди розробників є виявлення всіх можливих ризиків і присвоєння їм певного рівня пріоритету на основі їх значущості. Наступним кроком є розробка можливих стратегій подолання цих ризиків. В результаті цих дій можливі зміни в наступних стадіях розробки.В результаті цих дій можливі зміни в наступних стадіях розробки. Як результат роботи на цьому етапі створюється прототип.

3. Фаза розробки. На цьому етапі відбувається розробка і подальше тестування продукту. Під час першої ітерації, коли загальні вимоги ще не так чітко сформульовані, розробляється так званий концепція майбутнього продукту (Proof Of Concept), яка необхідна для отримання відгуку замовника. На наступних витках спіралі робочі версії продукту, або білди (builds), відправляються замовнику. Це дозволяє отримати більш детальний відгук і чіткіше сформулювати вимоги.

4. Планування наступної фази. На цьому етапі вся отримана інформація використовується для планування подальших етапів розробки.

Спіральну модель часто називають мета-моделлю, оскільки в ній використовуються два підходи: каскадний модель і модель прототипирования. Але вкрай важливо розуміти, що спіральна модель не є простою послідовністю етапів розробки, наступних каскадної моделі. Насправді, спіральна модель є досить гнучкою. Варто пам'ятати, що схема, наведена вище містить певні спрощення. Може здатися, що все стадії слідують одній спіральній послідовності. Але реальний життєвий цикл ПО більш гнучкий, ніж це

зображено на схемі.Існує навіть можливість повернутися до попередніх фаз в разі необхідності перегляду прийнятих рішень.

У будь-якої моделі розробки ПЗ є свої сильні і слабкі сторони. В цьому відношенні спіральна модель не є винятком. Давайте розглянемо її основні переваги та недоліки.

Переваги:

- Моніторинг ризиків є однією з головних особливостей, які роблять цю модель особливо привабливою в тому випадку, якщо вам належить управління великим, складним і дорогим проектом. Більш того, проект буде більш прозорим, оскільки спіральна модель спочатку була спроектована таким чином, щоб кожна ітерація ретельно аналізувалася;
- Замовник може побачити працюючу версію продукту вже на ранніх стадіях життєвого циклу ПЗ;
- Зміни можуть бути внесені на пізніх стадіях розробки;
- Проект може бути розділений на декілька частин і ті з них, які, згідно з аналізом, виявляться більш ризикованими, можуть бути реалізовані на ранніх стадіях. Такий підхід може знизити труднощі, пов'язані з управлінням проектом;
- Строгий контроль над документацією, як результат постійного аналізу ризиків.

Недоліки:

- Моніторинг ризиків вимагає додаткових ресурсів, а значить, ця модель може виявитися досить витратною. Кожна ітерація вимагає окремої експертизи, що

робить управління проектом складніше. Саме тому спіральна модель погано підходить для невеликих проектів;

- Велика кількість проміжних стадій розробки. Як наслідок - великий обсяг документації;
- На самих ранніх стадіях дата завершення роботи над проектом може бути невідома, що також ускладнює контроль над процесом розробки

2.2 Побудова UML діаграми діяльності програмної системи

Щоб зобразити поведінку досліджуваної системи, потрібно не тільки написати шляхи до зміни її станів, але й детально викласти основні моменти інтелектуальних та алгоритмічних адміністрацій , що можуть використовуватись в певних рамках в графічному представленні. Це є однією з причин використання блок-схем розрахунків. Вони підкреслюють достовірність перевірки інших базових чи істотних обмінів, які як і в цілому можуть мати ідеальні результати.

Передбачувані схематичні дії використовувались для виявлення завдань на мові UML. Спочатку повинна бути розроблена певна система, оскільки ми не зможемо просто написати якийсь код. Щоб грамотно впровадити такі рішення потрібна робота в команді та хороший колектив програмістів. Також варто зауважити, що оцінка вартості системи на ранніх етапах розвитку може бути досить складною та принести з собою деякі труднощі. UML діаграми – це один з інструментів, який допомагає нам це зробити та вирішити багато проблем на початковій стадії розробки. Також варто зауважити що це є корисним і на стадії аналізу, коли ми спілкуємось з замовником та дізнаємось, що справді йому потрібно та як ми можемо це реалізувати.

Діаграма діяльності - ще одна важлива діаграма в UML для опису динамічних аспектів системи. Це в основному блок-схема, що представляє потік від однієї діяльності до іншої діяльності. Діяльність може бути описана як робота системи.

Контрольний потік здійснюється від однієї операції до іншої. Цей потік може бути послідовним, розгалуженим або одночасним. Діаграми активності стосуються всіх типів управління потоком за допомогою різних елементів, таких як вилка, з'єднання тощо.

Основні цілі діаграм діяльності схожі з іншими чотирма діаграмами. Він фіксує динамічну поведінку системи. Інші чотири діаграми використовуються для відображення потоку повідомлень від одного об’єкта до іншого, але діаграма діяльності використовується для показу потоку повідомлень від однієї діяльності до іншої.

Діяльність - це певна операція системи. Діаграми активності використовуються не лише для візуалізації динамічного характеру системи, але й використовуються для побудови виконуваної системи за допомогою методів прямого та зворотного проектування. Єдине, що відсутнє на діаграмі діяльності - це частина повідомлення.

Він не показує жодного потоку повідомлень від однієї діяльності до іншої. Діаграма діяльності іноді розглядається як блок-схема. Хоча діаграми виглядають як блок-схема, вони не є. Він показує різні потоки, такі як паралельний, розгалужений, паралельний та одиничний.

Мета діаграми діяльності може бути описана як:

- Намалювати потік активності системи.
- Описати послідовність від однієї діяльності до іншої.
- Описати паралельний, розгалужений і паралельний потік системи.

Діаграми активності в основному використовуються як блок-схема, що складається з заходів, що виконуються системою. Діаграми активності не є точно

схемами, оскільки вони мають деякі додаткові можливості. До таких додаткових можливостей належать розгалуження, паралельний потік, плаваючий, тощо.

Перед тим, як скласти діаграму діяльності, ми повинні мати чітке розуміння елементів, які використовуються в діаграмі діяльності. Основним елементом діаграми діяльності є сама діяльність. Діяльність - це функція, яку виконує система. Після виявлення діяльності нам потрібно зрозуміти, як вони пов'язані з обмеженнями та умовами.

Перш ніж скласти діаграму діяльності, слід визначити наступні елементи:

- Діяльності
- Асоціації
- Умови
- Обмеження

Після того, як вищезазначені параметри будуть визначені, нам потрібно зробити ментальну компоновку всього потоку. Цей ментальний макет потім перетворюється на діаграму діяльності.

Особливим використанням є моделювання потоку управління від однієї діяльності до іншої. Цей потік управління не включає повідомлення.

Діаграма діяльності підходить для моделювання потоку активності системи. У додатку може бути декілька систем. Діаграма діяльності також фіксує ці системи та описує потік від однієї системи до іншої. Цього конкретного використання немає в інших діаграмах. Ці системи можуть бути базами даних, зовнішніми чергами або будь-якою іншою системою.

Зараз ми розглянемо практичні програми діаграми діяльності. З вищенаведеного обговорення видно, що діаграма діяльності складається з дуже високого рівня. Таким чином, це дає високий рівень перегляду системи. Цей вигляд

високого рівня в основному стосується бізнес-користувачів або будь-якої іншої особи, яка не є технічною особою. Ця діаграма використовується для моделювання діяльності, яка є не що інше, як бізнес-вимоги. Діаграма має більший вплив на розуміння бізнесу, а не на деталі впровадження.

Діаграму діяльності можна використовувати для:

- Моделювання робочого процесу за допомогою діяльності.
- Моделювання вимог бізнесу.
- Розуміння функціональних можливостей системи на високому рівні.
- Дослідження вимог бізнесу на більш пізньому етапі.

Отже , можна говорити про те , що діаграми діяльності є демонстраційними частинами того , які можливості для використання програми є у користувача.

У системі можна виділити наступні форми взаємодії з користувачем :

1. Авторизація
2. Реєстрація
3. Створення замовлення
4. Редагування замовлення
5. Перегляд наявних замовлень
6. Сортування замовлень
7. Видалення замовлень
8. Зміна категорії замовлень
9. Пошук замовлень

Дія “реєстрація” та “авторизація” дає користувачу можливість взаємодіяти з системою, відповідно до наявного статусу користувач має різні привілеї та можливості доступу в системі.

Створення замовлення – це можливість за допомогою якої користувач може створювати нові транспортні замовлення і відносити їх до певних категорій в загальному списку замовлень. Редагування замовлень – функція за допомогою якої користувач може вносити зміни у наявні замовлення. Перегляд замовлень – дозволяє користувачу чи гостю переглянути список замовлень та дані про ні замовлення. Сортування замовлень – це функція, що дозволяє користувачу відсортувати список замовлень згідно вибраних критеріїв. Це дозволить економити час та спростить взаємодію з системою. Видалення замовлень дозволить користувачам видаляти замовлення які не є актуальними з бази замовлень.

Зміна категорії замовлень – ця функція дозволить користувачу переносити замовлення з однієї категорії в іншу, що дозволить без значних змін швидко оновлювати інформацію. Пошук замовлень – одна з найважливіших функцій, оскільки дозволяє економити багато часу, якщо відома назва потрібного замовлення. Максимальний ефект цієї функції розкривається з використання у парі із сортуванням замовлень. З детальним планом проектованої системи можна ознайомитись на рисунку 2.2

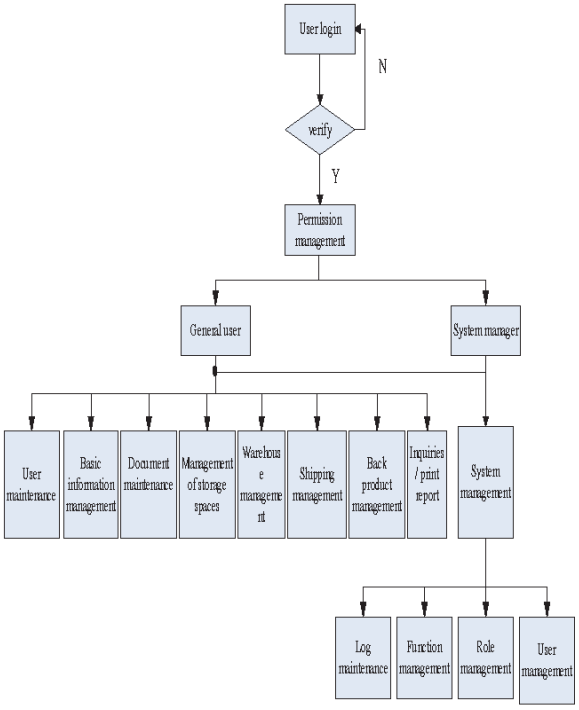

Рис 2.2

На цьому рисунку ми можемо оцінити наскільки розгалуженими та універсальними є можливі дії клієнта в додатку. Початком роботи є момент, коли користувач пробує увійти до веб-додатку. Наступний етап – визначення чи ідентифікація користувача з можливістю реєстрації в системі. Якщо користувач не є зареєстрованим у системі, то його буде направлено на сторінку з можливістю реєстрації. Після чого проводиться авторизація та присвоюється статус Користувача з розширенням усіх прав для використання ресурсу. Для незареєстрованих користувачів буде використовуватись гостьовий профіль з допомогою якого він зможе ознайомитись з додатком, сформувати замовлення чи провести повну реєстрацію аккаунта.

Зареєстровані користувачі будуть мати доступ до повного списку замовлень та можливості їх сортування, видалення чи редагування. Також варто зазначити, що буде можливість завантаження інформації про замовлення у форматі PDF. Під час ключових дій користувача буде здійснюватись запит до сервера який буде відправляти оновлену інформацію.

2.3 Моделювання архітектури системи

При розробці технічної бази проекту було вирішено, що проект за своїми характеристиками найкраще будувати на клієнт-серверній архітектурі з підтримкою баз даних.

Обчислювальна модель клієнт-сервер зайняла міцне місце серед методів розподілених обчислень. І хоча різні виробники пропонують різний програмне забезпечення, проте, що таке архітектура клієнт-сервер, цілком склалося єдиної думки.

Поділ програми на окремі завдання, що розміщуються на різних платформах для більшої ефективності. Як правило, це означає, що програма представлення даних знаходиться на машині користувача (на клієнті), а програма управління даними і самі дані – на сервері. В залежності від програми та використовуваного програмного забезпечення вся обробка даних може здійснюватися на клієнтській машині або розподілятися між клієнтом і сервером. Сервер з'єднується зі своїми клієнтами по мережі.

Серверне програмне забезпечення приймає запити від клієнтського програмного забезпечення і повертає йому результати Мережеве оточення, в якому управління даними здійснюється на серверному вузлі, а інших вузлів надається доступ до даних. Спільна з клієнтом обробка запитів клієнта серверові повернення результатів клієнту. У цій моделі обробка даних додатком розподілена (необов'язково порівну) між клієнтом і сервером. Обробка даних ініціюється і частково управляється клієнтом, але не в режимі «ведучий-ведений», а, скоріше, в режимі співробітництва

Клієнтські процеси посилають запити серверного процесу, що посилаєш назад результати цих запитів. Як випливає з назви, серверні процеси надають послуги і своїм клієнтам, як правило, виконуючи специфічну обробку, яку можуть виконати тільки вони. Клієнтський процес, звільнений від виконання складної обробки

транзакції, може виконувати іншу корисну роботу. Взаємодія між клієнтським і серверним процесами являє собою спільний 36истем изображ обмін, в якому активність виходить від клієнта, а сервер реагує на цю активність.

Кожен сервер в оточенні клієнт-сервер надає клієнтам набір послуг. Найбільш поширеним типом сервера в архітектурі клієнт-сервер є сервер баз даних, як правило, керуючий реляційної базою даних. Високопродуктивний сервер забезпечує колективний доступ декількох клієнтів до однієї і тієї ж бази даних. Крім клієнтів і серверів в оточення клієнт-сервер входить мережа. Обчислювальна модель клієнт-сервер за визначенням є розподіленою. Користувачі, додатки та ресурси розташовуються на різних комп'ютерах і з'єднані загальною локальної, глобальної або складовою мережею.

Традиційна архітектура клієнт-сервер складається з двох рівнів, або ланок: клієнтського і серверного. В останні роки все більшого поширення набуває трьохступенева архітектура клієнт-сервер. У даній архітектурі прикладне програмне забезпечення розподілено між системами трьох типів: користувальницької системою, проміжним сервером і сервером бази даних. Комп'ютер користувача являє собою клієнта, і в три ланкової моделі це, як правило, «тонкий» клієнт. Проміжні системи є, по суті, шлюзами між «тонкими» клієнтами і різноманітними серверами баз даних. Проміжні системи повинні вміти перетворювати протоколи і відображати одні типи запитів до баз даних на інші. Взаємодія між проміжним сервером і сервером баз даних також відповідає моделі к клієнт –сервер. Таким чином, проміжна система функціонує одночасно і як клієнт, і як сервер.

Для високорівневого огляду архітектури системи використаємо UML діаграму розгортання.

В рамках специфікації UML, діаграми розгортання моделюють фізичну архітектуру системи. Діаграми розгортання показують взаємозв'язок між програмним та апаратним компонентами системи та фізичним розподілом обробки.

Діаграми розгортання, яка зазвичай будується на етапі реалізації, показують фізичну структуру вузлів у розподіленій системі, артефакти, що зберігаються на кожному вузлі, а також компоненти та інші елементи, що використовуються артефактами. Вузли представляють апаратні пристрої, такі як комп'ютери, датчики та принтери, а також інші пристрої, які підтримують середовище виконання середовища системи. Шляхи зв'язку та розгортання відносин моделюють зв'язки в системі.

Amazon Web Services Elastic Compute Cloud – сервіс на якому розміщується сервер та база даних. Сервер працює на ОС Ubuntu 16.6 та з доступом через SSH протокол. На сервері розгортається Docker з контейнерами, в яких будуть запускатись процеси бази даних MongoDB, NodeJS сервер, Nginx та Let’s encrypt SSL.

Docker — інструментарій для управління ізольованими Linux- контейнерами. Docker доповнює інструментарій LXC більш високорівневим API, що дозволяє керувати контейнерами на рівні ізоляції окремих процесів.

Серверна частина Meteor додатку є надбудовою Node.js сервера.

MongoDB compute engine – сервер для баз даних MongoDB. Основна база даних системи запущена на такому сервері. База даних використовується NodeJS додатком, що запущений на основному сервері системи.

Client device – пристрій клієнта. Для взаємодії із системою, користувач має використовувати додаток, який встановления на пристрій, що працює на базі операційної системи типу iOS або Android. Підключення до системи відбувається через протокол HTTP, клієнт зв’язується із Meteor додатком запущеним на основному сервері, у відповідь клієнту встановлюється DDP з’єднання, що є надбудовою над Websocket.

Після встановлення з’єднання та ініціалізації клієнтської частини, взаємодія із сервером відбувається через протокол DDP.

Діаграму розгортання для розроблювальної 38системи зображено на рисунку 2.3.

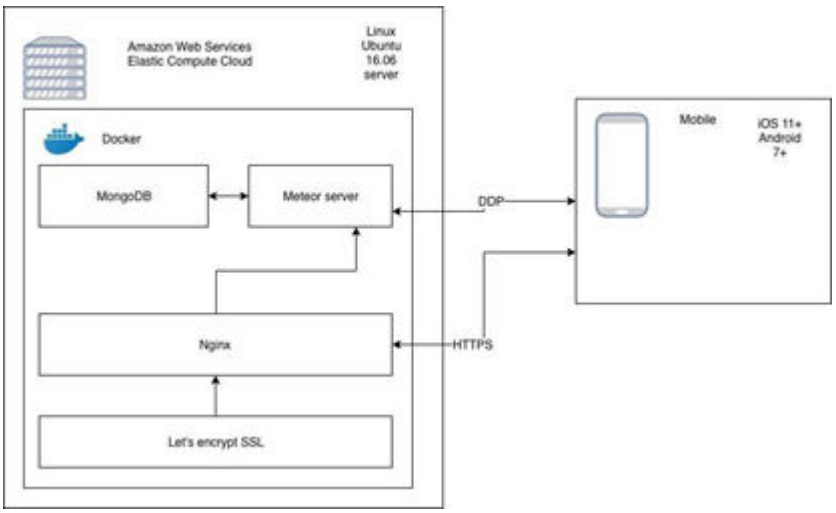

Рис 2.3 Діаграма розгортання системи

3. РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Вибір мови та середовища розробки

Відомо, що розроблювана система складається з кількох компонентів, які взаємодіють між собою. Щоб полегшити розробку і спростити подальшу підтримку додатків, взаємодії розробників різних модулів було обрано для розробки мову програмування Java Script на всіх рівнях.

JavaScript - це інтерпретована мова програмування з об'єктно-орієнтованими можливостями. З точки зору синтаксису базова мова Java Script нагадує C, C ++ і Java такими програмними конструкціями, як інструкція if, цикл while і оператор. Однак ця подібність обмежується синтаксичної схожістю. JavaScript - це нетипізована мова, в ній не потрібно визначати типи змінних. Об'єкти в JavaScript відображають імена властивостей на довільні значення. Цим вони більше нагадують асоціативні масиви Perl, ніж структури C або об'єкти C ++ або Java. Механізм об'єктно-орієнтованого успадкування JavaScript скоріше схожий на механізм прототипів в таких маловідомих мовах, як Self, і сильно відрізняється від механізму наслідування в C ++ і Java. Як і Perl, JavaScript - це мова, що інтерпретується, і деякі його інструменти, наприклад регулярні виражені кошти роботи з масивами, реалізовані за образом і подобою мови Perl.

Ядро мови JavaScript підтримує роботу з такими простими типами даних, як числа, рядки і булеві значення. Крім цього він має вбудовану підтримку масивів, дат і об'єктів регулярних виразів.

Зазвичай JavaScript застосовується в веб-браузерах, а розширення його можливостей за рахунок введення об'єктів дозволяє організувати взаємодію з користувачем, управляти веб-браузером і змінювати вміст документа, що відображається в межах вікна веб-браузера. Ця вбудована версія JavaScript запускає сценарії, впроваджені в HTML код веб-сторінок. Як правило, ця версія називається

клієнтським мовою JavaScript, щоб підкреслити, що сценарій виконується на клієнтському комп'ютері, а не на веб-сервері. В основі мови JavaScript і підтримуваних ним типів даних лежать міжна рідні стандарти, завдяки чому забезпечується прекрасна совместімостьмежду реалізаціями. Деякі частини клієнтського JavaScript формально стандартизовані, інші частини стали стандартом де-факто, але є частини, якіє специфічними розширеннями конкретної версії броузера. Сумісність реалізацій JavaScript в різних браузерах часто приносить немалобеспокойств програмістам, які використовують клієнтський мову JavaScript.

Коли інтерпретатор JavaScript вбудовується в веб-браузер, результатом є клієнтський JavaScript. Це, безумовно, найбільш поширений варіант JavaScript, і більшість людей, згадуючи JavaScript, зазвичай мають на увазі саме клієнтський JavaScript. У цій книзі клієнтський мову JavaScriptопісивается разом з базовимJavaScript, який представляє собою підмножину клієнтського JavaScript.

Клієнтський JavaScript включає в себе інтерпретатор JavaScript і об'єктнуюмодель документа (Document Object Model, DOM), яка визначається веб-браузером.Документи можуть містити JavaScript сценарії, які в свою чергу можуть використовуватися модель DOM для модифікації документа або управління способоми відображення. Іншими словами, можна сказати, що клієнтський JavaScript дозволяє визначити поведінку статичного вмісту веб-сторінок. Клієнтський JavaScript є основою таких технологій розробки веб-додатків, як DHTML, і таких архітектур, як Ajax. Поява AJAX змінила ситуацію та повернула увагу професійної спільноти до мови. В результаті, були розроблені та покращені багато практик використання JavaScript (зокрема, тестування та налагодження), створені бібліотеки та фреймворки , поширилося використання JavaScript поза браузером.

Коли говорять про наслідування, то JavaScript має тільки одну конструкцію: об'єкти. Кожен об'єкт має приховану властивість (про яку кажуть [[Prototype]]), яка утримує посилання на інший об'єкт, з назвою prototype. Цей prototype свою чергу

має свій prototype , і так далі поки такий об'єкт не стане дорівнювати null в значені prototype. За визначенням, у значення null не може бути prototype, і це є останнім значенням у такому ланцюгу з прототайпів. Майже усі об'єкти в JavaScript є екземплярами Object який є першою ланкою в ланцюгі зі значень prototype.

Хоча це розцініються як одна зі слабкостей JavaScript, тим не менше таке наслідовування, фактично, більш потужніше, ніж класичне. Наприклад, це доволі поширена практика будувати класичну модель на базі прототипної моделі.

Клієнтська частина є мобільним застосунком з підтримкою операційних систем Windows. Внаслідок того, що смартфони та мобільні додатки стають настільки популярними, веб-розробники шукають способи створення мобільних додатків за допомогою JavaScript. Ця популярність призвела до розробки багатьох фреймворків для цієї мови програмування, здатних виконувати нативний код на мобільних пристроях.

Завдяки зростаючій популярності, мова JavaScript в нашому часі активно розвивався, а сучасний веб-розробник драматично змінює за порівнянням з недавнім минулим. Ви працюєте у Веб при допомозі JavaScript, працюючи на серверах, а також в браузері, було складно себі уявити ще кілька років тому - в найкращому випадку, такі можливості можна було лише в пісочних версіях Flash або Java Applets.

Node.js - це пакет, що включає в себе JavaScript V8 від Google, увімкнення абстракції платформи - бібліотека libuv, а також базову бібліотеку, яка сама написана на JavaScript." створюють веб-сайти, працюють у реальному часі та оснащені push-функції, під враженням від таких пропозицій як Gmail. У Node.js він запропонував програмний інструмент для роботи з парадигмою неблокирующего, подієво-орієнтованого вводу / виводу. Через 20 років парадигми «Веб-додаток без збереження стану на базі комунікації запросити-відключити без збереження стану» у нас накопичувались - це додатки з двонаправленою зв'язкою в реальному часі.

Коротше: Node.js уявляється у застосуванні реального часу, так як працює push-технологія через веб-сайти. Що таке в цьом такому революційному? Ну, як вже говорилося, з 20-ти років использование вищої парамігмі з'являються Такі двонаправлені Додатки, де зв'язок может ініціюваті як кліент, так и сервер, и забувається Присутні до вільного обміну данімі. Така технологія різко контрастує з типовою парадигмою веб-відключніків, де комунікацію завжди ініціює кліент. Окрім цього, будь-яка технологія засновано на відкрітому веб-стеці (HTML, CSS та JS), працює Ідеально через стандартний порт 80.

Основна ідея Node.js: використовувати неблокуюче подієво-орієнтований введення / виведення, щоб залишатися легковажним і ефективним при поводженні з додатками, що обробляють великі обсяги даних в реальному часі і функціонуючими на розподілених пристроях. Node.js не є платформою на всі випадки життя, яка буде домінувати в світі веб-розробки. Навпаки, це платформа для вирішення строго визначених завдань. Розуміти це абсолютно необхідно. Зрозуміло, не варто використовувати Node.js для операцій, інтенсивно навантажують процесор, більш того - застосування Node.js в важких обчисленнях фактично анулює всі його переваги. Node.js дійсно хороший для створення швидких масштабованих мережних додатків, оскільки дозволяє одночасно обробляти величезну кількість з'єднань з великою пропускною здатністю, що рівноцінно високою масштабованості.

Тонкощі роботи Node.js «під капотом» досить цікаві. У порівнянні з традиційними варіантами веб-сервісів, де кожне з'єднання (запит) породжує новий потік, навантажуючи оперативну пам'ять системи і, врешті-решт, розбираючи цю пам'ять без залишку, Node.js набагато економічніше: він працює в єдиному потоці, при викликах використовує неблокуюче введення / виведення, дозволяє підтримувати десятки тисяч конкурентних з'єднань (які існують в циклі подій).

Хоча Node.js особливо хороший в контексті додатків, що працюють в реальному часі, він цілком підходить і для надання інформації з об'єктних баз даних.

Дані, збережені у форматі JSON, дозволяють Node.js функціонувати без втрати відповідності та без перетворення даних.

Також можна розглянути Cordova та React Native, що є досить популярними сьогодні. Cordova дозволяє займатись розробкою на мобільних платформах типу Android , iOS та Windows Phone. З іншої сторони React Native дозволяє підтримувати різні операційні системи типу Android чи iOS. Для вирішення, який з фреймворків буде кращим для розробки слід їх оцінити та порівняти їх переваги та недоліки.

Cordova дозволяє використовувати стандартні веб-технології, такі як HTML5, CSS3 і JavaScript для крос платформної розробки, уникаючи рідної мови розробки для кожної з мобільних платформ. Додатки виконуються всередині обгортки націленої на кожну платформу і покладаються на стандартні API для доступу до датчиків пристрою, даними і станом мережі.

Apache Cordova використовується для:

- Мобільний розробник і хочете розширити додаток на більш ніж одну платформу, без необхідності повторно реалізувати його для кожної мови розробки платформ і набору інструментів.
- веб-розробник і хочете, щоб розгорнути веб-додаток, який упаковано для поширення в різних магазинах додатків.
- Мобільний розробник, зацікавлений в змішування компонентів власного додатка з WebView можна отримати доступ до API рівні пристрою, або якщо ви хочете розробити плагін інтерфейс між рідною і WebView компонентами.

Додаток Cordova покладаєтьься на загальний файл config.xml, який містить інформацію про програму та визначає параметри, що впливають на те як додаток працює, такі як, чи реагує воно на зміну орієнтації пристрою. Цей файл відповідає специфікації W3C Упакування веб-додатки, або widget,. Сам додаток реалізований як веб-сторінки, за замовчуванням локальний файл під назвою index.html, який посилається на будь-який CSS, JavaScript, зображення, файли мультимедіа або інші

ресурси необхідні для його запуску. Додаток виконує як WebView в межах оболонки додатки, яку ви поширюєте в магазини додатків.

WebView з підтримкою Cordova може забезпечувати додатки і він повністю призначений для користувача інтерфейс. На деяких платформах вона також може бути компонентом у великих, гібридні програми, який об'єднують WebView з іншими компонентами програми. Інтерфейс плагіна доступний для Cordova і інших компоненти, для взаємодії один з одним. Це дозволяє викликати код на мові платформи з JavaScript. В ідеалі на декількох платформах пристроїв узгоджуються JavaScript API, щоб цей машинний код. Починаючи з версії 3.0 плагіни надають прив'язки до стандартних API інтерфейсів пристрою.

Сторонні плагіни надають додаткові прив'язки для функції не обов'язково доступних на всіх платформах. Можна знайти ці сторонні плагіни в реєстрі плагінів і використовувати їх в своєму додатку. Ви можете також розробити свої власні плагіни, як описано в розділі "Керівництво по розробці плагінів". Модулі можуть виявитися необхідними, наприклад, для зв'язку між Cordova і власними компонентами. Cordova не надає будь-яких віджетів призначеного для користувача інтерфейсу або MV фреймворків. Cordova надає тільки під час виконання, в якій ті можуть виконувати. Якщо ви хочете використовувати UI-віджети або MV фреймворк, вам потрібно буде вибрати їх і включити їх в додаток самостійно, як ресурси третьої сторони.

В Cordova можна використовувати два основних робочих процесу для створення мобільних додатків. Хоча ви можете використовувати будь-який робочий процес для виконання однієї і тієї ж задачі, кожен з цих шляхів має свої переваги:

- Крос платформний робочий процес: Використовуйте цей робочий процес якщо ви хочете, щоб ваш додаток запускався на максимально можливій кількості мобільних платформ, з мінімальними потребами для платформо-специфічної розробки. Цей робочий процес формується біля утиліти cordova, також відомому як Cordova CLI, який був введений починаючи з Cordova 3.0.

CLI це високо рівневий інструмент який дозволяє побудувати проекти для якомога більшої кількості платформа одночасно, абстрагуючись якомога більше функціональності низько-рівневий скриптів. CLI копіює загальний набір web ресурсв в підкаталоги для кожної мобільної платформи, робить будь-які необхідні зміни в конфігурацію для кожної платформи, запускає сценарії збірки для створення виконуваних файлів програми. CLI також надає загальний інтерфейс для застосування плагінів для вашого застосування.

- Від платформи орієнтований процес розробки: Використовуйте цей процес якщо ви хочете сфокусуватися на побудові програми для однієї платформи і вам буде необхідно вносити модифікації на низькому рівні. Ви повинні використовувати цей підхід, наприклад, якщо ви хочете щоб ваше додаток поєднувало власні компоненти з веб-орієнтованими компонентами Cordova. Як правило цей робочий процес використовується, якщо необхідно змінити проект за допомогою SDK. Цей робочий процес спирається на набір скриптів нижчого рівня, які пристосовані для кожної підтримуваної платформи і окрему утиліта Plugman, яка дозволяє застосовувати плагіни індивідуально до обраної платформі.

Недоліком є те, що додатки на Cordova є веб сторінками, які формують нативний інтерфейс. Внаслідок цього з’являється значне навантаження на CPU, яке призводить до зниження продуктивності застосунка і може бути не зручним для користувача.

В основі React Native лежить React (JavaScript фреймворк) для розробки інтерфейсів додатків, що використовують JavaScript. В основному програми React Native не використовують WebView для відтворення інтерфейсу програми, як у Cordova. Весь інтерфейс у React Native відображається за допомогою нативних компонентів, які працюють в окремому потоці і обробляються за допомогою GPU, а не CPU. Інтерпретатор JavaScript вбудований для розробки бізнес-логіки, обробки даних та управління інтерфейсом, в той час як рендеринг бере на себе

нативна частина. Недоліком є те, що в деяких випадках необхідно мати справу з модулями, написаними на Java та Objective-C. Перевагою є те, що React Native є швидким середовищем, яке керується фреймворком React, що добре зарекомендувало себе у спільноті розробників.

React дозволяє розробникам створювати великі веб-застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-застосунків.

В даний час React використовують Khan Academy, Netflix, Yahoo, Airbnb, Sony, Atlassian та інші.

Для розроблюваної системи важливим є простота в користуванні і продуктивність. Тому для роботи було вибрано технологію React Native, яка буде в змозі задовільнити потреби і користувача і розробника.

Для серверної частини вибрано JavaScript-фреймворк, що носить назву Meteor.

Meteor − платформа, що побудована на основі Node.js для створення у режимі реального часу веб-додатків. ЇЇ можна вважати певним прошарком поміж вашою базою даних і інтерфейсом програм, які стежать за їхньою синхронізацією.

Ця платформа була обрана через такі особливості:

- програми написані на платформі Meteor є по-замовчуванню додатками реального часу;

- клієнтська і серверна частина розробляється за допомогою однієї мови - Javascript;
- велика швидкість розробки через наявність смарт-пакетів;
- нова технологія, яка стрімко розвивається.

На виході ми маємо дуже потужну та відносно просту програму для створення додатків з можливістю створення в реальному часі, яка дозволяє абстрагуватись від багатьох звичних неприємних речей та підводних каменів розробки.

Вибір цього фреймворку стався тому, що він дозволить швидко розробляти веб-додатки та в майбутньому дана система зможе впроваджуватись не тільки для мобільних операційних систем, але й як розширення для браузера. Meteor підходить як для невеликих проектів чи стартапів, так і для великих за обсягом проектів завдяки своїй масштабності. Також варто зауважити, що серверна частина фреймворку побудована на Node.js, що э безумовною перевагою. Це дозволяэ використовувати сторонній пакет из публічною репозицією NMP.

NPM (Node Package Manager) - це менеджер пакунків для мови програмування JavaScript. Для середовища виконання Node.js є менеджером пакунків за замовчуванням. Включає в себе клієнт командного рядка, який також називається npm, а також онлайн-базу даних публічних та приватних пакунків, яка називається реєстром npm. Реєстр доступний через клієнт, а доступні пакунки можна переглядати та шукати через веб-сайт npm. Менеджер пакунків та реєстр керуються npm, Inc.

Як середовище розробки було обрано Microsoft Visual Studio Code (VS Cod) — відносно новий текстовий редактор, випущений Microsoft. Він, також як і Atom, ґрунтується на движку Chromium, але володіє своїми унікальними фичами, такими, як, наприклад, IntelliSense "з-коробки".

За останні декілька років популярність Visual Studio Code зросла на ринку IDE з відкритим сирцевим кодом. VS Code був офіційно випущений у 2015 та зараз

використовується приблизно 35% усіх розробників, згідно з опитуванням Stack Overflow 2018. VS Code підтримує мови JavaScript, Babel, CoffeeScript, TypeScript та інші. Варто зауважити, що в середовищі також підтримується авто доповнення, навігація по коду, авто виправлення інтеграцію системи управління та ре факторинг. Також особливістю інтегрованого середовища розробки VS Code є робота з проектами(включаючи ре факторинг коду JavaScript, що міститься в різних тегах та файлах проекту, а також вкладеного в HTML). Підтримується і множинна вкладеність, яка дозволяє документу на HTML, вкладеному в скрипт на Javascript, в який вкладено інший код, всередині якого вкладений Javascript, з підтримкою рефакторинга в конкретних конструкціях.

3.2 Вибір СКБД та опис її фізичної моделі

Основна функції СКБД – безпосереднє керування даними у зовнішній пам'яті. Ця функція включає збереження та ведення структурної інформації (даних), перетворення даних за запитом на структурну інформацію для користувача. При цьому повинні забезпечуватися: простота використання системи; можливість автономного функціонування при порушеннях мережі або при адміністративних потребах; високий ступінь ефективності.

Існує декілька видів СКБД, які вирізняються моделями зберігання та доступу до інформації. Відомі ієрархічні, мережні та реляційні моделі даних. Дані у мережних моделях мають файлову структуру, а не груп таблиць єдиної структури БД. Більшість сучасних СКБД використовують реляційну модель, оскільки складаються з набору зв'язаних між собою об'єктів-таблиць (від англ, relation – відношення). У реляційній моделі даних зв'язок між відношеннями, що являють собою відповідні об'єкти, здійснюється за допомогою атрибутів. Атрибут – це поіменована характеристика об'єкта, за допомогою якої моделюється його властивість . Значення кожного атрибута вибирають з відповідної множини значень, що включає всі потенційні значення, які можуть бути присвоєні атрибуту. Ця множина називається доменом. Атрибути поділяються на прості (атомарні) й

складені. Прості атрибути не можуть бути розділені на більш дрібні складові. Реляційна модель складається із трьох частин – структурної, маніпуляційної та цілісної. У структурній частині фіксується, що єдиною структурою, яка використовується у реляційній базі, є нормальне я-арне відношення (л зв'язків між об 'єкта/uu БД). У маніпуляційній частині моделі затверджуються два фундаментальні механізми маніпулювання реляційними БД – реляційна алгебра та реляційне обчислення**.** Перший механізм базується в основному на класичній теорії множин, а другий – на класичному логічному апараті обчислення предиктів першого порядку .

З іншого боку, СКБД – це численні програмні засоби, більшість яких є не закінченими програмами, а спеціалізованими мовами програмування, за допомогою яких можна створювати такі структури, які потрібні для роботи та вводити в них необхідні елементи управління. До таких мов програмування належать Clipper, Paradox, FoxPro та багато інших, що вирізнялися інтерфейсом, складом операторів та рівнем предметної орієнтації. При використанні цих мовних засобів для створення конкретної системи під потреби підприємства необхідні програмісти. Як правило, кожна група програмістів розробляють своє власне програмне середовище (СКБД першого покоління), що призвело до появи численних несумісних програм. У системах керування другого покоління файли взаємопов'язаних даних об'єднуються у бази даних з певними закономірностями, що дозволило дещо уніфікувати програми для різних задач.

Стан справ значно змінився після появи у складі пакета Microsoft Office системи управління базами даних Access з інтерфейсом, орієнтованим на будь-якого користувача. За допомогою Access користувачі отримали зручний спосіб для створення та експлуатації досить потужних баз даних без необхідності щось програмувати відповідними мовами. Таким чином, системи керування даними третього покоління мають більш розвинуті можливості, доступні звичайним користувачам, з'явилась можливість уникнення надлишкових даних. До СКБД

третього покоління належать такі як Visual FoxPro, DBasa – системи, Oracle Application і низка інших. Вибір типу СКБД для розв'язання задач керування даними залежить від обсягів інформації, з якою працює користувач, кількості функціональних задач, що використовують дану базу. Відповідно до потужності та функціональних можливостей СКБД поділяють на класи. СКБД класу А типу FOXPRO, CLIPPER, ACCESS мають порівняно невеликі можливості щодо зберігання та обробки інформації. Фізично така СКБД може розміщуватись на одному автоматизованому робочому місці (АРМ) фахівця. Для великих підприємств можливості таких систем недостатні. СКБД MS Access входить до пакета програм MS Office і через широке розповсюдження пакета СКБД Access знаходить повсюдне використання як настільна система для індивідуального та офісного використання. СКБД MS Access підтримує реляційну модель БД і містить усі необхідні інструментальні засоби як для створення локальної бази, так і бази даних у локальній мережі з файловим сервером або бази даних на SQL Server і додатків користувача для роботи із цими БД. У системі є різні способи керування даними – через систему меню, панель інструментів, контекстне меню, покажчиком миші, комбінацію клавіш. Ці засоби, завдяки людино- орієнтованому інтерфейсу, створені для непідготовленого користувача і можуть бути швидко освоєні у процесі використання програми.

В Access база даних – це файл, який містить дані у вигляді однієї або кількох таблиць. База даних, створена на локальному комп'ютері, зберігається разом з елементами додатка й програмним кодом в одному mdb-файлі, що спрощує як створення, так і розповсюдження додатків БД.

Інструментальні засоби СКБД MS Access включають:

- засоби графічного конструювання, що дозволяють користувачеві створювати об'єкти БД і об'єкти додатка, не використовуючи програмування;

- майстри, що дозволяють створювати об'єкти БД і об'єкти додатка в діалоговому режимі, а також виконувати різноманітні функції з реорганізації та перетворення БД;
- засоби програмування, куди входять мова структурованих запитів SQL, мова макрокоманд і об'єктно-орієнтована мова програмування високого рівня Visual Basic for Application (VBA).

Серед засобів графічного конструювання та діалогових засобів Access варто виділити:

- засоби для створення таблиць і схем даних;
- засоби конструювання запитів вибірки й запитів на зміну даних бази;
- засоби створення екранних форм, призначених для введення, перегляду й обробки даних у діалоговому режимі;
- засоби створення звітів, призначених для перегляду та виведення на друк даних з бази і результатів їх обробки;
- засоби створення сторінок доступу до даних, що забезпечують роботу з БД у середовищах Інтернет і Інтранет;
- засоби конструювання інтерфейсу користувача – меню, панелей керування додатком, що дозволяють об'єднати операції з роботи з БД у єдиний технологічний процес.

За необхідності створення нової бази даних користувач може використати численні заготовки – шаблони різних об'єктів бази, що входять до складу Access, або створювати свої об'єкти для конкретних задач.

Етапи створення бази даних у середовищі MS Access:

- визначення мети створення бази даних, яка визначатиме перелік таблиць БД;

- визначення структури таблиць (полів та їх типів);
- призначення ключів таблиць та визначення зв'язків між таблицями;
- завантаження даних до таблиці;
- створення інших об'єктів бази даних – запитів, форм, звітів, макросів та модулів.

У MS Access для обслуговування, як і в інших СКБД, використовується спеціалізоване лінгвістичне забезпечення. У більшості випадків це спеціалізована мова для взаємодії з структурованою інформацією БД – SQL (Structered Query Language – структурована мова запитів), що була розроблена у середині 70-х років у рамках проекту експериментальної реляційної СКБД System R. Назва мови частково відображає її суть як зручного інструмента для формулювання запитів до реляційних баз даних. Розроблена мова SQL являє собою деяку комбінацію реляційного обчислення кортежів і реляційної алгебри. Разом з тим можливості мови SQL для окремих операцій ширші, ніж у реляційній алгебрі. Базовий набір мовних операторів SQL уміщує оператори визначення схеми БД, вибірки і маніпулювання даними, авторизації доступу до даних, підтримки вбудування SQL в інші мови програмування. Мова допускає три типи синтаксичних конструкцій, які починаються з ключового слова SELECT (вибрати):

- специфікації курсора (cursor specification) для тимчасового зберігання результатів запиту у курсорі;
- оператора вибірки (select statement) – список полів, що вибираються;
- підзапит (subquery) на виконання певних операцій з даними полів.

Основою всіх них є синтаксична конструкція "табличний вираз (table expression)". Семантика табличного виразу складається з того, що на основі послідовного використання розділів from (список таблиць), where (умова вибірки), group by

(умова групування) і having, із заданих у розділі from таблиць будується нова результуюча таблиця з невизначеною послідовністю рядків. СКБД класу В типу Informix, SyBase, IT-підприєство, 1С: Предприятие 8.0 вимагають значних обсягів пам'яті для розміщення інформації та потужного програмного забезпечення, яке не вміщується на робочому комп'ютері АРМ. Більш того, при використанні потужних баз даних на великих підприємствах інформацію бази використовують десятки фахівців і керівників, що вимагає певної організації доступу до даних – архітектури програмного забезпечення.

При проектуванні документо-орієнтованої бази даних важливим моментом є створення так званої схеми. Схема бази даних - це структура системи баз даних описана формальною мовою, яка підтримується системою управління баз даних (СУБД) і відноситься до організації даних для створення плану побудови бази даних з розподілом на таблиці. Формально схема баз даних являє собою набір правил, які називаються обмеженнями цілісності. Кінцевою метою створення схеми є зменшення потенційної суперечливості збереженої в базі даних інформації.

Розроблювальна система використовує базу даних MongoDB. MongoDB — документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів. Код MongoDB написаний на мові C++ і поширюється в рамках ліцензії AGPLv3.

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій. У MongoDB є вбудовані засоби із забезпечення

шардінгу (розподіл набору даних по серверах на основі певного ключа), комбінуючи який з реплікацією даних можна побудувати горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера на кластер проводиться без зупинки роботи БД простим додаванням нових машин.

Основні можливості MongoDB:

- Документо-орієнтоване сховище (проста та потужна JSON-подібна схема даних)
- Досить гнучка мова для формування запитів
- Динамічні запити
- Повна підтримка індексів
- Профілювання запитів
- Швидкі оновлення «на місці»
- Ефективне зберігання бінарних даних великих обсягів, наприклад, фото та відео
- Журналювання операцій, що модифікують дані в БД
- Підтримка відмовостійкості і масштабованості: асинхронна реплікація, набір реплік і шардінг
- Може працювати відповідно до парадигми MapReduce

MongoDB реалізує новий підхід до побудови баз даних, де немає таблиць, схем, запитів SQL, зовнішніх ключів і багатьох інших речей, які притаманні об'єктно-реляційних баз даних. З часів динозаврів було звичайною справою зберігати всі дані

в реляційних базах даних (MS SQL, MySQL, Oracle, PostgresSQL). При цьому було не так важливо, а чи підходять реляційні бази даних для зберігання даного типу даних чи ні. На відміну від реляційних баз даних MongoDB пропонує документо-орієнтовану модель даних, завдяки чому MongoDB працює швидше, має кращу масштабованість, її легше використовувати.

Але, навіть враховуючи всі недоліки традиційних баз даних і гідності MongoDB, важливо розуміти, що завдання бувають різні і методи їх вирішення бувають різні. В якійсь ситуації MongoDB дійсно поліпшить продуктивність вашої програми, наприклад, якщо треба зберігати складні за структурою дані. В іншій же ситуації краще буде використовувати традиційні реляційні бази даних. Крім того, можна використовувати змішані підхід: зберігати один тип даних в MongoDB, а інший тип даних - в традиційних БД.

Вся система MongoDB може представляти не тільки одну базу даних, що знаходиться на одному фізичному сервері. Функціональність MongoDB дозволяє розташувати кілька баз даних на декількох фізичних серверах, і ці бази даних зможуть легко обмінюватися даними і зберігати цілісність.

Одним з популярних стандартів обміну даними та їх зберігання є JSON (JavaScript Object Notation). JSON ефективно описує складні за структурою дані. Спосіб зберігання даних в MongoDB в цьому плані схожий на JSON, хоча формально JSON не використовується. Для зберігання в MongoDB застосовується формат, який називається BSON (Бісон) або скорочення від binary JSON. MongoDB написана на C ++, тому її легко перенести на найрізноманітніші платформи. MongoDB може бути розгорнута на платформах Windows, Linux, MacOS, Solaris. Можна також завантажити вихідний код і самому скомпілювати MongoDB, але рекомендується використовувати бібліотеки з офсайта. Якщо реляційні бази даних зберігають рядки, то MongoDB зберігає документи. На відміну від рядків документи можуть зберігати складну за структурою інформацію. Документ можна уявити як

сховище ключів і значень.Ключ являє просту мітку, з якою асоційований певний шматок даних.

Однак при всіх відмінностях є одна особливість, яка зближує MongoDB і реляційні бази даних. У реляційних СУБД зустрічається таке поняття як первинний ключ. Це поняття описує якийсь стовпець, який має унікальні значення. У MongoDB для кожного документа є унікальний ідентифікатор, який називається _id. І якщо явно не вказати його значення, то MongoDB автоматично згенерує для нього значення.

Кожному ключу зіставляється певне значення. Але тут також треба враховувати одну особливість: якщо в реляційних базах є чітко окреслена структура, де є поля, і якщо якесь поле не має значення, йому (в залежності від налаштувань конкретної бд) можна привласнити значення NULL. У MongoDB все інакше. Якщо якомусь ключ не порівнювати значення, то цей ключ просто опускається в документі і не вживається. Якщо в традиційному світі SQL є таблиці, то в світі MongoDB є колекції. І якщо в реляційних БД таблиці зберігають однотипні жорстко структуровані об'єкти, то в колекції можуть містити найрізноманітніші об'єкти, що мають різну структуру і різний набір властивостей.

Система зберігання даних в MongoDB представляє набір реплік. У цьому наборі є основний вузол, а також може бути набір вторинних вузлів. Всі вторинні вузли зберігають цілісність і автоматично оновлюються разом з оновленням головного вузла. І якщо основний вузол з якихось причин виходить з ладу, то один з вторинних вузлів стає головним. Відсутність жорсткої схеми бази даних і в зв'язку з цим потреби при щонайменшій зміні концепції зберігання даних пересоздавать цю схему значно полегшують роботу з базами даних MongoDB і подальшим їх масштабуванням. Крім того, економиться час розробників. Їм більше не треба думати про пересоздании бази даних і витрачати час на побудову складних запитів. Однією з проблем при роботі з будь-якими системами баз даних є збереження даних великого розміру. Можна зберігати дані в файлах, використовуючи різні мови

програмування. Деякі СУБД пропонують спеціальні типи даних для зберігання бінарних даних в БД (наприклад, BLOB в MySQL).

На відміну від реляційних СУБД MongoDB дозволяє зберігати різні документи з різним набором даних, однак при цьому розмір документа обмежується 16 мб. Але MongoDB пропонує рішення - спеціальну технологію GridFS, яка дозволяє зберігати дані за розміром більше, ніж 16 мб. Система GridFS складається з двох колекцій. У першій колекції, яка називається files, зберігаються імена файлів, а також їх метадані, наприклад, розмір. А в іншій колекції, яка називається chunks, у вигляді невеликих сегментів зберігаються дані файлів, зазвичай сегментами по 256 кб. Для тестування GridFS можна використовувати спеціальну утиліту mongofiles, яка йде в пакеті mongodb. Подібно до інших документо-орієнтованих СКБД (CouchDB тощо), MongoDB не є реляційною СКБД. Є докладна і якісна документація, велике число прикладів і драйверів під популярні мови Java, C++, C#, PHP, Python, Perl, Ruby.

При випуску одразу було заявлено, що реліз MongoDB 1.0 готовий до використання у виробництві як в якості одиничного хоста, так і в зв'язках master/slave. Код цього релізу досить стабільний і перевірений в промисловій експлуатації протягом 1,5 років. За можливості MongoDB має бути розгорнута мінімум на двох серверах, використовуючи реплікацію Master/Slave. Це забезпечує наявність актуальних даних при виході з ладу однієї з СКБД.

MongoDB — продукт досить молодий, і відтак у ньому зустрічаються помилки, з'являються нові можливості тощо. Характерний високий темп розробки (проект пишуть не тільки волонтери, а й компанія людей на повній зайнятості). MongoDB є невідємною частиною фреймворку Meteor, оскільки завдяки їй і реалізується реактивість – одна із головних особливостей фреймворку. Звісно є рішення для використання інших баз даних в Meteor, але вони не практикуються по причині того, що потреби в цьому не має, бо в такому випадку пропаде реактивість, ну і сама по собі MongoDB – це популярна і ефектифна база даних, яка добре підходить для вирішення задач, притаманних для систем накшталт такої яка зараз розробляється.

Для створення розроблюваної системи було обрано саме цю базу даних за її явні переваги описані вище. Це документо-орієнтована база даних, де дані зберігаються у вигляді документів в колекціях. Це дозволяє MongoDB працювати значно швидше тамати краще масштабування у порівнянні з реляційними базами даних. Бінарний формат зберігання даних дозволяє пришвидшити процеси, але вимагає більшої кількості місця.

Після проведеного аналізу ми можемо виділити основні сутності предметної області, що відповідають базі даних.

Лістинг 3.1 - завантаження даних

```
def load_imdb_sentiment_analysis_dataset(data_path, seed=123):
    """Loads the IMDb movie reviews sentiment analysis dataset.
    # Arguments
        data_path: string, path to the data directory.
        seed: int, seed for randomizer.
    # Returns
        A tuple of training and validation data.
        Number of training samples: 25000
        Number of test samples: 25000
        Number of categories: 2 (0 - negative, 1 - positive)
    # References
        Mass et al., http://www.aclweb.org/anthology/P11-1015
        Download and uncompress archive from:
        http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
    """
    imdb_data_path = os.path.join(data_path, 'aclImdb')
    # Load the training data
    train_texts = []
    train_labels = []
    for category in ['pos', 'neg']:
        train_path = os.path.join(imdb_data_path, 'train', category)
        for fname in sorted(os.listdir(train_path)):
```

```
            if fname.endswith('.txt'):
                with open(os.path.join(train_path, fname)) as f:
                    train_texts.append(f.read())
                train_labels.append(0 if category == 'neg' else 1)
    # Load the validation data.
    test_texts = []
    test_labels = []
    for category in ['pos', 'neg']:
        test_path = os.path.join(imdb_data_path, 'test', category)
        for fname in sorted(os.listdir(test_path)):
            if fname.endswith('.txt'):
                with open(os.path.join(test_path, fname)) as f:
                    test_texts.append(f.read())
                test_labels.append(0 if category == 'neg' else 1)
    # Shuffle the training data and labels.
    random.seed(seed)
    random.shuffle(train_texts)
    random.seed(seed)
    random.shuffle(train_labels)
    return ((train_texts, np.array(train_labels)),
            (test_texts, np.array(test_labels)))
```

Лістинг 3.2 - Код порівняння алгоритмів, та способів представлення даних.

```
def classifier_comparator(vectorizers, classifiers, data):
    target = [1 if i < 25000 else 0 for i in range(50000)]
    result = []
    for classifier_name, classifier in list(classifiers):
        for vectorizer, vectorizer_name in vectorizers:
            pipeline = Pipeline([(vectorizer), (classifier)])
            clf_accuracy, tt_time = accuracy_summary(pipeline, x_t, y_t, x_v, y_v)
            result.append((classifier_name, vectorizer_name, clf_accuracy, tt_time))
    return result
def accuracy_summary(pipeline, data, target):
    t0 = time()
```

```
x_train, x_test, y_train, y_test = train_test_split(data, target, train_size=0.75)
sentiment_fit = pipeline.fit(x_train, y_train)
train_test_time = time() - t0


y_pred = sentiment_fit.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy {accuracy}")
print(classification_report(y_test, y_pred, target_names=['class 0', 'class 1']))
scores = cross_val_score(sentiment_fit, data, target, cv=5)
print(scores)
print("cross-validation accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
return accuracy, scores.mean(), train_test_time
```

Приклад форми довідки на рис 3.1

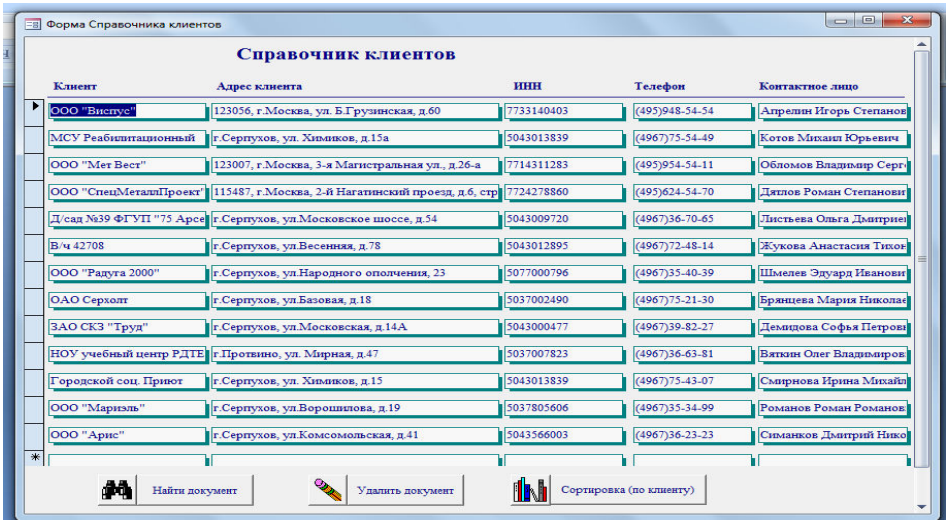

Рис 3.1 Форма-довідка

Приклад форми поставки на рис 3.2

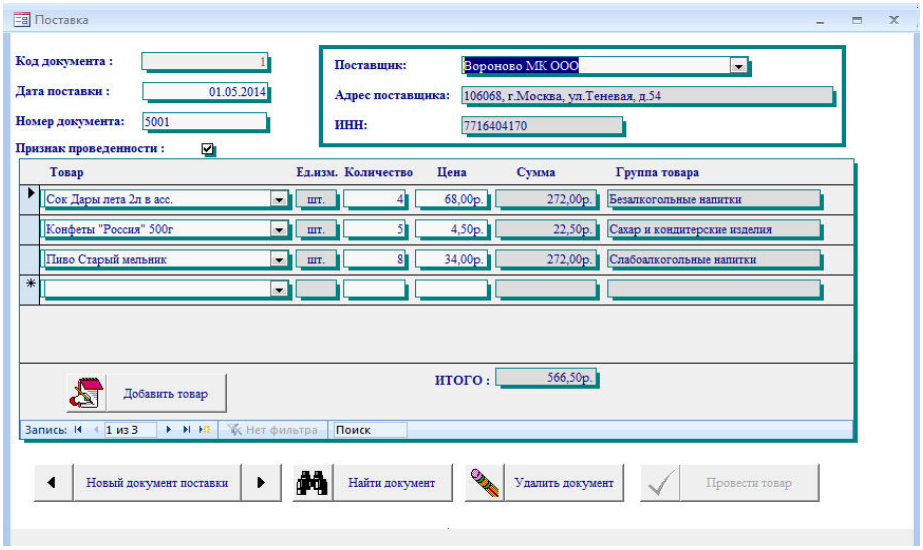

Рис 3.2 Форма поставки

4. ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИНА

Метою цього розділу дипломної роботи є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності симульованої екосистеми для систем типу Windows, Linux та MacOS.

Для здійснення розрахунків було використано реальні дані проведення науково-дослідної роботи, був описаний технологічний процес розробки із зазначенням трудомісткості кожної операції, враховано суму матеріальних затрат, та витрат на оплату праці основного і допоміжного персоналу, включно з відрахуванням на соціальні заходи, обчислено додаткові затрати, а також було визначено економічну ефективність та термін окупності продукту.

4.1 Планування стадій та етапів проектування програмного забезпечення

Відповідно до ст. 1 Закону № 3792, комп'ютерна програма – це набір інструкцій у вигляді слів, цифр, кодів, схем, символів чи у будь-якому іншому вигляді, виражених у формі, придатній для зчитування комп'ютером, які приводять його в дію для досягнення певної мети або результату. В такому самому значенні розуміється цей термін і в контексті податкового обліку (пп.8.2.2 Закону про прибуток). Статтею 8 Закону № 3792 передбачено, що комп'ютерні програми є об'єктом авторського права [7].

Діяльність з програмування — це процес розробки програми, який є представлений послідовністю таких кроків: формулювання вимог до програми та розробка алгоритмів; кодування (запис алгоритму на необхідній мові програмування), відлагодження коду, тестування програмного продукту, доопрацювання програми, розробка довідкової системи.

Проектування програмного забезпечення складається з послідовних, цілеспрямованих, взаємоповязаних та взаємообумовлених етапів, на які можна розділити весь часовий відрізок, який відводиться на розробку програмного продукту (таблиця 4.1).

Таблиця 4.1 — Етапи розробки проекту

Найменування		Вид роботи	
стадії	Етапу	шифр	зміст роботи
1 Підготовча стадія	1.1 Вивчення стану предметної області	1.1.1	Дослідження проблеми
		1.1.2	Вибір платформи для реалізації проекту
		1.1.3	Вивчення та аналіз аналогічних розробок
		1.1.4	Економічне обґрунтування доцільності виконання проекту
	1.2 Розробка технічного завдання	1.2.1	Складання технічного завдання
		1.2.2	Узгодження технічного завдання із зацікавленими сторонами
		1.2.3	Складання плану та розрахунок розробки
2 Технічна пропозиція	2.1 Аналіз технічного завдання та техніко-економічне обґрунтування проекту	2.1.1	Доведення техніко-економічного обґрунтування
		2.1.2	Аналіз технічного завдання та визначення пріоритетних аспектів розробки
		2.1.3	Доведення та уточнення загального обсягу робіт, строків виконання та витрат
3 Теоретична розробка	3.1 Теоретичне вивчення задачі	3.1.1	Дослідження технічних особливостей інформаційної системи
		3.1.2	Визначення переліку технологій, які використовуватимуться при розробці та мови програмування
		3.1.3	Визначення форматів даних та запитів і їх узгодження із розробниками проекту
		3.1.4	Розробка алгоритмів роботи програми на високому рівні
		3.1.5	Розробка структури систем забезпечення та схеми взаємодії її компонент
4 Практична реалізація	4.1 Реалізація окремих модулів ПЗ	4.1.1	Розробка алгоритмів роботи програми на низькому рівні
		4.1.2	Розробка окремих класів та компонування їх у модулі
		4.1.3	Розробка графічного користувацького інтерфейсу для системи
	4.2 Відладка ПЗ	4.2.1	Автономна відладка окремих модулів системи
		4.2.2	Комплексна відладка ПЗ системи

Продовження таблиці 4.1

Найменування		Вид роботи	
	4.3 Розробка субмодуля	4.3.1	Розробка структурної схеми субмодуля
		4.3.2	Розробка функціональної схеми субмодуля
		4.3.3	Розробка алгоритму функціювання субмодуля
		4.3.4	Обґрунтування вибору елементного базису та розробка схеми
5 Доробка всього комплексу	5.1 Тестування всієї системи	5.1.1	Тестування системи у реальних умовах
		5.1.2	Доробка систем із урахуванням результатів тестування
	5.2 Підготовка документації по системі	5.2.1	Підготовка звіту про розробку системи
		5.2.2	Підготовка технічної документації
		5.2.3	Розробка довідкової системи, допоміжної і супроводжувальної документації, запис CD/DVD диска
6 Заключна стадія	6.1 Ознайомлення зацікавлених осіб з проектом	6.1.1	Підготовка презентації
		6.1.2	Демонстрація системи

Головна мета планування процесу розробки продукту — це визначення ресурсів які будуть необхідні на всіх етапах розробки програмного забезпечення (далі ПЗ).

Традиційний процедурний підхід для розробки ПЗ в основі якого лежать алгоритми, процедури та функції передбачає розробку ПЗ як монолітного композиту, що в подальшому вимагає великих витрат на супровід та модернізацію проекту.

Об'єктно-орієнтований підхід, який ґрунтується на основі об'єктів певних класів, котрі описують певну область, описують методи та володіють певними атрибутами, орієнтовані на варіанти використання та покроковий, покомпонентний процес розробки.

Підготовча стадія, технічна пропозиція і заключна стадія при розробці з об’єктно-орієнтованим підходом залишаються незмінними. Стадії теоретичної розробки, практичної реалізації та модернізація всього комплексу програмного продукту описані з точки зору специфіки об'єктно-орієнтованого підходу із врахуванням життєвого циклу розробки ПЗ і представлені із уточненням фахівців залучених у кожний робочий процес розробки (таблиця 4.2) [8].

При створенні ПЗ за об'єктно-орієнтованим підходом необхідно залучити більшу кількість фахівців, більшу увагу приділяти покроковій компонентній розробці, що може збільшити робочий час і витрати. Однак цей підхід значно скорочує витрати на подальший супровід і модернізацію, що в загальному призведе до зменшення витрат усього проекту.

Таблиця 4.2 — Робочі процеси життєвого циклу розробки ПЗ

№	Робочі процеси	Діяльності	Співробітники	Артефакти
1	“Визначення вимог”	Ідентифікація актантів (А) і варіантів використання (ВВ), Модель ВВ Описи (формалізація) ВВ і сценаріїв реалізації, Визначення пріоритетності ВВ Створення прототипів інтерфейсів користувача (ІК)	Системний аналітик Специфікатор ВВ	Модель ВВ Актанти Глосарій ВВ Прототип ІК
2	“Проектування”	Проектування архітектури, Визначення вузлів та мережевих конфігурацій Визначення систем та їх інтерфейсів Визначення підсистем та зв’язків між ними Визначення інтерфейсів підсистем Визначення архітектурно значущих класів та класів проектування,Визначення загальних механізмів проектування Проектування ВВ Проектування класів Визначення вимог до реалізації програмного продукту	Архітектор Інженер з ВВ	Модель проектування Модель розміщення Опис архітектури Проекти реквізитів ВВ Класи проектування

Продовження таблиці 4.2

№	Робочі процеси	Діяльності	Співробітники	Артефакти
3	“Реалізація”	Модель реалізації, Компоненти, Інтерфейси компонентів, Опис архітектури План збирання системи, Реалізація архітектури, Ітеративна реалізація класів Проектування з генерацією програмного коду, Збірка системи, Планування білдів Реалізація білдів і системи в цілому	Системний інтегратор Інженер-програміст	Модель реа-лізації Опис архітекту-ри План збірки Компоне-нти Підсист-еми реалізації Інтерфейси
4	“Тестування”	Модель тестування Тестові приклади Процедура тестування Тестові компоненти План і оцінка тестування Розробка тестів Тестування цілісності Тестування системи Оцінка результатів тестування	Інженер з тестування Системний тестер	Модель тестування Тестові приклади План тестування Оцінюв-ання тестів

Терміни розробки залежать від замовника, від наданої необхідності інформації (зразків довідників, документів), рівня та порядку оплати та інших умов від замовлення, і можуть становити від 2 тижнів до 6 місяців і більше (за даними IT компаній)

Вартість складання технічного завдання зазвичай складає до 10% від планувальної вартості розробки і становить від 200 до 5000 гривень (за даними IT компаній). Роботу зі складанням технічного завдання веде керівник проекту разом із програмістами та проконсультувавшись із замовником.

Для визначення загальної тривалості проведення робіт дані витрат часу для окремих операцій техпроцесу було зведено в таблицю (таблиця 4.3).

Таблиця 4.3 — Тривалість та трудомісткість розробки та реалізації програмного проекту

Найменування роботи	Виконавець, посада, спеціальність	К-сть виконав-ців	Тривалість виконання роботи, дні
Дослідження предметної області, вибір середовища для реалізації проекту, вивчення та аналіз аналогічних розробок	Керівник проекту Програміст	2	1
Економічне обгрунтування доцільності виконання проекту, складання ТЗ, узгодження ТЗ із зацікавленими сторонами, складання плану та розрахунок розробки	Керівник проекту Програміст	2	1
Доведення техніко-економічного обґрунтування, аналіз технічного завдання та визначення пріоритетних аспектів розробки, доведення та уточнення загального обсягу робіт, строків виконання та витрат	Керівник проекту Програміст	2	1
Теоретична розробка процедурний підхід	Програміст	2	2
Теоретична розробка об'єктно-орієнтований підхід	Системний аналітик, специфікатор ВВ, архітектор, інженер з ВВ	3	3
Практична реалізація процедурного підходу	Програміст	2	4
Практична реалізація об'єктно-орієнтованого підходу	Системний інтегратор, інженер-програміст	3	5
Тестування програмного забезпечення	Інженер з тестування	1	4
Доробка всього комплексу програмного забезпечення	Програміст	1	3
Заключна стадія	Керівник проекту Програміст	2	1

Як для процедурного, так і для об’єктно-орієнтованого підходу, підготовча і заключна стадії та технічна пропозиція будуть виконуватися однаково, теоретична розробка і практична реалізація для об’єктно-орієнтованого підходу вимагатимуть

залучення більшої кількості ІТ-спеціалістів, що може збільшити тривалість виконання роботи.

4.2 Розрахунок витрат на реалізацію проекту та оцінка економічної ефективності

Оцінка вартості комп'ютерних програм базується на витратному підході: використанні первісної вартості об'єктів, виходячи з фактичних витрат на розробку та доведення до комерційного використання з урахуванням амортизації. Оподаткування заробітної плати програмістів підприємства-розробника, яким встановлено певний посадовий оклад, відбувається аналогічно оподаткуванню зарплати працівників інших галузей господарства.

Враховуючи залежність якості кінцевого програмного продукту від кваліфікації програмістів, розробники часто йдуть на додаткові заходи їх стимулювання,які найчастіше втілюються у:

- в певній системі преміювання (за виконання графіку розробки чи його дострокове виконання, оптимізація етапів розробки тощо). Премії оподатковуються аналогічно витратам на оплату праці;

- авторській винагороді за кожну продану одиницю програми. Сума не обкладається внесками до пенсійного фонду та фондів соціального страхування, а використання авторських прав оформлюється відповідними договорами (ліцензійний авторський договір).

Разом з тим до створення ПЗ можуть залучатися також позаштатні програмісти як зареєстровані, так і не зареєстровані підприємцями. В обох випадках співпраця з ними здійснюється на підставі цивільно-правового договору, найчастіше – договорами підряду. Щодо оподаткування виплат за договором підряду, то все залежить від того, чи зареєстрований виконавець підприємцем:

- якщо виконавець за договором підряду не зареєстрований фізичною особою-підприємцем, то виплати за таким договором оподатковуються ПДФО за ставкою 15 %, також нараховуються (у розмірі 33,2 %) та утримуються (у розмірі 2 %) внески до

Пенсійного фонду (п. 1 та п. 4 ст. 4 Закону № 400). Оподаткування провадиться у межах максимальної величини, що дорівнює 15 розмірам прожиткового мінімуму, встановленого для працездатних осіб;

- якщо ж виконавець за договором підряду є фізичною особою-підприємцем, яка сплачує єдиний податок чи знаходиться на загальній системі оподаткування, виплати, що здійснюються на його користь в рамках договору підряду, не оподатковуватимуться ПДФО.

Вважатимемо, що усі програмісти працюють у штаті підприємства-розробника і їм встановлено певний посадовий оклад. Місячний оклад, денна заробітна плата, трудомісткість і основна заробітна плата кожного учасника технічного процесу представлено у таблиця 4.4.

Таблиця 4.4 — Сума часткових заробітних плат

	Місячний оклад, грн.	Денна зар. плата, грн	Трудомісткість, людино-дні		Основна заробітна плата, грн.	
			Процедурний підхід	Об'єктно-орієнтований підхід	Процедурний підхід	Об'єктно-орієнтований підхід
Керівник проекту	17800	712	4	4	2848	2848
Інженер-програміст	14600	584	17	33	9928	19272
Інженер-тестувальник	9800	392	4	4	1568	1568
Дизайнер	7500	300	3	3	900	900
Всього			28 днів	44 днів	15244 грн.	24588 грн.

Визначимо витрати на основну заробітну плату. Вони складають суму заробітних плат за кожну із робіт. Витрати на основну зарплату для процедурного та об’єкт-орієнтованого підходів:

Додаткова заробітна плата обчислюється за формулою $ЗП_{дод}=ЗП_{осн}\cdot 0.2$.

Обчислимо витрати на додаткову зарплату для двох підходів:

$ЗП_{осн(п)}= 2848 + 9928 + 1568 + 900 = 15244$ (грн.);

$Зп_{осн(о)}= 2848 + 19272 + 1568 + 900 = 24588$ (грн.);

Це сума часткових заробітних плат за кожну роботу, і вона приведена в таблиці 4.4.

Додаткова заробітна плата обчислюється за формулою:

$ЗП_{дод}= 0,2 \cdot ЗП_{осн}$.

$Зп_{дод(п)}= 15244 \cdot 0,2 = 3050$ (грн.);

$Зп_{дод(о)}= 24588 \cdot 0,2 = 4918$ (грн.);

Таким чином загальний фонд заробітної плати, що обчислюється за формулою:

$ФЗП = Зп_{осн}+ЗП_{дод}$.

$ФЗП_{(п)} = 15244 + 3050 = 18294$ (грн.);

$ФЗП_{(о)} = 24588 + 4918 = 29506$ (грн.);

З цієї суми утримуються обов’язкові відрахування на заробітну плату: єдиний соціальний внесок, який складає 3,6% від суми нарахованої заробітної плати та податок на доходи фізичних осіб, який складає 15% від суми нарахованої заробітної плати, зменшеної на суму єдиного внеску на загальнообов’язкове соціальне страхування та податкової соціальної пільги.

Таким чином обов'язкові відрахування на заробітну плату для працівників складають:

Утримання єдиного соціального внеску:

$Відр_{ЄСВ1} = 0{,}036 \cdot ФЗП = 0{,}036 \cdot 18294 = 659$ (грн.);

$Відр_{ЄСВ2} = 0{,}036 \cdot ФЗП = 0{,}036 \cdot 29506 = 1062$ (грн.);

Податок на доходи фізичних осіб:

$Відр_{ПДФО1} = 0{,}15 \cdot (ФЗП - Відр_{ЄСВ1}) = 0{,}15 \cdot (18294\text{-}659) = 2645{,}25$ (грн.);

$Відр_{ПДФО2} = 0{,}15 \cdot (ФЗП - Відр_{ЄСВ2}) = 0{,}15 \cdot (29506\text{-}1062) = 4266{,}6$ (грн.);

$Відр_1 = Відр_{ЄСВ1} + Відр_{ПДФО1} = 659 + 2645{,}25 = 3304{,}25$ (грн.);

$Відр_2 = Відр_{ЄСВ2} + Відр_{ПДФО2} = 1062 + 3304{,}25 = 4366{,}25$ (грн.);

Законом № 1621 підрозділ 10 розділу XX Перехідних положень Податковий кодекс України від 02 грудня 2010 року № 2755-VI зі змінами та доповненнями (далі – ПКУ) доповнено пунктом 16, яким тимчасово, до 1 січня 2015 року, встановлено військовий збір.

Водночас Законом України від 28 грудня 2014 року № 71-VIII «Про внесення змін до Податкового кодексу України та деяких законодавчих актів України щодо податкової реформи» (далі – Закон № 71), який набрав чинності з 01.01.2015, оподаткування військовим збором подовжено до набрання чинності рішенням Верховної Ради України про завершення реформи Збройних Сил України (пункт 16 підрозділ 10 розділу XX Перехідних положень ПКУ) [35].Водночас Законом України від 28 грудня 2014 року № 71-VIII «Про внесення змін до Податкового кодексу України та деяких законодавчих актів України щодо податкової реформи» (далі – Закон № 71), який набрав чинності з 01.01.2015, оподаткування військовим збором подовжено до набрання чинності рішенням Верховної Ради України про завершення реформи Збройних Сил України (пункт 16 підрозділ 10 розділу XX Перехідних положень ПКУ) [9].

Платниками збору в ромірі 1,5% від заробітної плати є особи, визначені пунктом 162.1 статті 162 цього ПКУ (підпункт 1.1 пункт 16 підрозділу 10 ПКУ) [10].

Військовий збір з фізичних осіб:

$ВЗФО = 0{,}015 \cdot ФЗП$;

$ВЗФО_1 = 0{,}015 \cdot 18294 = 274{,}41$ (грн.);

$ВЗФО_2 = 0{,}015 \cdot 29506 = 442{,}59$ (грн.);

Нарахування на фонд оплати праці, які включають відрахування до Пенсійного фонду, фонду з тимчасової втрати працездатності, фонду з безробіття і фонду страхування від нещасних випадків на виробництві; для бюджетної організації тариф на фонд оплати праці встановлено на рівні 36,3%, для підприємців розмір єдиного внеску залежно від класу професійного ризику виробництва становить від 36,76 % до 49,70 %. Зокрема, видання програмного забезпечення – 36,77%) [11].

Нарахування на фонд оплати праці (ФОП):

$ФОП_{ЄСВ} = 0{,}3677 \cdot ФЗП$

$ФОП_{ЄСВ1} = 0{,}3677 \cdot 18294 = 6726{,}70$ (грн.)

$ФОП_{ЄСВ2} = 0{,}3677 \cdot 29506 = 10849{,}35$ (грн.)

Всього витрат:

$В_{ЗП1} = ЗП_1 + ФОП_{ЄСВ1} = 18294 + 6726{,}70 = 25020{,}7$ (грн.);

$В_{ЗП2} = ЗП_2 + ФОП_{ЄСВ2} = 29506 + 10849{,}35 = 40355{,}35$ (грн.);

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни;

$$M_{Bi} = q_i \cdot p_i$$

де q_i — кількість витраченого матеріалу i-го класу; p_i — ціна матеріалу i-го виду.

Звідси, загальні матеріальні витрати можна визначити:

$$З_{м.в.} = \sum M_{B\,i}$$

Проведені розрахунки занесено в таблицю 4.5:

Таблиця 4.5 — Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Фактично витрачено матеріалів	Ціна 1-ці, грн.	Загальна сума витрат, грн.
1	SCRUM-дошка	шт	1	500,00	500,00
2	Папір для друку	листів	120	0,25	30,00
3	Чорнила для принтера	шт	1	65,00	65,00
4	Маркери	шт	4	14,00	56,00
Всього					651,00

Згідно постанов Національної комісії, що здійснює державне регулювання у сфері енергетики (згідно Постанов Національної комісії, що здійснює державне регулювання у сфері енергетики (НКРЕ) від 22.01.2001р.№47 зі змінами і доповненнями, від 06.12.2002р. № 1358, від 22.10.2004р. № 1030, від 28.04.2016р. № 755 на підставі Закону України від 03.12.1999р № 1274-XIV “Про внесення змін до Закону України “Податкового кодексу України”, згідно Закону України “Про міський електричний транспорт” від 29 червня 2004 року № 1914 – IY; також затвердженого постановою Кабінету Міністрів України від 01.06.2011р. № 869, Порядку розрахунку роздрібних тарифів на електричну енергію, тарифів на розподіл електричної енергії (передачу електричної енергії місцевими (локальними) електромережами), тарифів на постачання електричної енергії за регульованим тарифом, затвердженого постановою Національної комісії, що

здійснює державне регулювання в сфері енергетики та комунальних послуг (НКРЕКП) "Про ринкове формування роздрібних тарифів на електричну енергію, що відпускається для кожного класу споживачів, крім населення, на території України", ця сума становить 2,50 грн/кВт.г. [12].

Витрати на електроенергію одиниці обладнання визначаються за формулою:

$$З_е = W \cdot T \cdot S$$

де W – необхідна потужність, кВт; T – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії. S = 2,50грн/кВт.г.

$З_{е1} = 1{,}5 \cdot 224 \cdot 2{,}5 = 840$ (грн.);

$З_{е2} = 1{,}5 \cdot 352 \cdot 2{,}5 = 1320$ (грн.);

Розрахунок суми амортизаційних відрахувань.

Комп’ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 % (квартальна – 15 %).

Амортизація за час (період) використання обладнання (комп’ютера) складає долю затрат, які припадають на дослідницьку роботу (написання програми), і визначається:

$$A = \frac{С_Б * N_A * T_{ФАК}}{Т_{год}}$$

де $С_Б$ — балансовна вартість обладнання, грн; N_A – норма амортизаційних відрахувань в рік, $Т_{год}$ — річний робочий фонд часу, год.; $Т_{ФАК}$ — фактичний час роботи обладнання по написанню програми, год.

Таблиця 4.6 — Перелік обладнання необхідного для розробки

Найменування	Кількість, шт.	Ціна за одиницю продукту, грн.	Сума, грн.
Комп'ютер	4	15775	63100
Принтер	1	2000	2000
Хостинг	1	995	995
Доменне ім'я	1	366	366
Всього			66461

$$A_1 = \frac{66461 * 0{,}6 * 224}{2016} = 4430{,}75(\text{грн.}) A_2 = \frac{66461 * 0{,}6 * 352}{2016} = 6962{,}60(\text{грн.})$$

Накладні витрати пов'язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

Залежно від організаційно-правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників.

$НВ = 0{,}5 \cdot ЗП_{осн}$.

$НВ_п = 0{,}5 \cdot 18294 = 9147$ грн.;

$НВ_о = 0{,}5 \cdot 29506 = 14753$ грн.

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у собі собівартість і планований прибуток.

Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво

$Св = В_{зп} + З_{мв} + З_е + А + НВ;$

$Св_1 = 25020{,}7 + 651 + 840 + 4430{,}75 + 9147 = 40089{,}45$ (грн.);

$Св_2 = 40355{,}35 + 651 + 1320 + 6962{,}60 + 14753 = 64041{,}95$ (грн.).

Приймемо прибуток на рівні 27%. Для нових інноваційних продуктів, що користуються високим попитом на ринку.

Отже, вартість розробленого програмного забезпечення:

$В_1 = Св_1 + 0,27 \cdot Св_1 = 40089,45 + 0,27 \cdot 40089,45 = 50913,60$ (грн.);

$В_2 = Св_2 + 0,27 \cdot Св_2 = 64041,95 + 0,27 \cdot 64041,95 = 81333,27$ (грн.)

Економічна ефективність (*Е*) полягає у відношенні результату виробництва до затрачених ресурсів:

$$Е = \frac{П}{Св}$$

де *П* — прибуток, (*П* = *В* — *Св*); Св — собівартість.

Якщо ринкова вартість програмного продукту рівна прийнятій, то економічна ефективність визначається встановленим рівнем прибутку.

$Е_п = (50913,60 - 40089,45) / 40089,45 = 0,27;$

$Е_о = (81333,27 - 64041,95) / 64041,95 = 0,27.$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($Т_{ок}$):

$$Т_{ок} = \frac{1}{Е}$$

У нашому випадку $Т_{ок1} = Т_{ок2} = 1 / 0,27 = 3,7$ років, що є нормальним, оскільки допустимим вважається термін окупності до 5 років.

Загальна вартість пропонованих робіт по розробці програмного продукту становить 18294 грн. для першого варіанту та 29506 грн. для другого. Оскільки ефективність для обидвох проектів відповідно до встановленого рівня прибутку становить 0,27, що є високим показником, то проводити дані роботи варто і вкладені кошти окупляться за 3,7 року.

4.3 Визначення витрат на супровід та модернізацію програмного продукту та уточнений аналіз ефективності вкладених інвестицій

Виходячи із експертних оцінок і складності програми, приймемо величину витрат на супровід і модернізацію програмного забезпечення, створеного за процедурним методом 55% від початкових витрат, а за об'єктно-орієнтованим – 25%.

Собівартість модернізації:

$СвМ_{п} = 0,55 \cdot Св_{п}$=0,55 · 40089,45 = 22049,20 грн.,

$СвМ_{о} = 0,25 \cdot Св_{о}$=0,25 · 64041,95 = 16010,48 грн.

Вартість модернізації для споживача:

$М_{п}$= 0,5 · $В_{п}$ = 0,55 · 50913,60 = 28002,50 грн.,

$М_{о}$= 0,25 · $В_{о}$ = 0,25 · 81333,27 = 20333,32 грн.

Таким чином, уже після першої модернізації, загальні витрати на створення і супровід ПЗ для виробника за об'єктно-орієнтованим методом менші, ніж за процедурним, навіть якщо його собівартість є дещо дорожчою.

$ЗВ_{п\ (вир)}$ = 40089,45 + 22049,20 = 62138,65 грн.,

$ЗВ_{о\ (вир)}$ = 64041,95 + 16010,48 = 80052,43 грн.

Як і для споживача:

$ЗВ_{п}$ = 50913,60 + 28002,50 = 78916,1 грн.,

$ЗВ_{о}$ = 81333,27 + 20333,32 = 101666,60 грн.

Річна економія витрат за всіма можливими напрямами і додатковими витратами, пов'язаними з супроводом і тільки одноразовою модернізацією (у розрахунку на одиницю продукції) при об’єктно-орієнтованому методі порівняно із процедурним:

$\Delta C_{(вир)} = ЗВ_{1(вир)} - ЗВ_{2(вир)} = 68091,95 - 80052,43 = -11960,48$ грн,

$\Delta C = ЗВ_1 - ЗВ_2 = 78916,1 - 101666,60 = -22750,5$ грн,

Визначення ЧПД відбувається за формулою:

$$ЧПД = \sum_{i=1}^{i} ГП_i a_{ТВi} - \sum_{i=1}^{i} ІК_i a_{ТВi};$$

де $Гп_i$ — грошовий потік і-го розрахункового року;

$ІК_i$ – сума інвестицій і-го розрахункового року;

$\alpha_{ТВi}$ – коефіцієнт дисконтування (коефіцієнт приведення інвестицій і грошового потоку до теперішньої вартості).

Для розрахунку коефіцієнта дисконтування (коефіцієнта приведення) грошових потоків за роками періоду економічного життя інвестицій використовується формула:

$$a = \frac{1}{(1+i)n}$$

де i — ставка дисконтування або норма дисконту, $i = 0,27$;

n — час або кількість періодів (років), протягом якого планується отримувати дохід.

$$a_0 = 1, a_1 = \frac{1}{1+0,27} = 0,79$$

Вважатимемо, що обидва програмних продукти однаково забезпечують потреби і вимоги споживача, і тому придбання першої чи другої програми однаково вплинуть на розмір його додаткових доходів на вкладений капітал, Тому приймемо цю величину за постійну, а порівняння дохідності двох проектів проведемо тільки за витратами.

Усі результати розрахунків, приведених в розділі зведені у таблицю 4.7

Таблиця 4.7 — Результати розрахунків

№ п.п.	Назва	Процедурний підхід	Об’єкто-орієнтований підхід
1	Зарплата основна, грн	18294	29506
2	Зарплата додаткова, грн	3050	4918
3	Фонд заробітньої плати (1+2)	21344	34424
4	Обов’язкові відрахування на заробітну плату (4.1+4.2), грн	3304,25	4366,25
4.1	Утримання єдиного соціального внеску (3,6%), грн	659	1062
4.2	Податок на доходи фізичних осіб (15%), грн	2645,25	3304,25
5	Військовий збір (1.5%), грн	264,5	3304,25
6	Відрахування на ФОП (36,77%), грн	6726,70	10849,35
7	Разом на оплату праці (3+6), грн	28070,7	45273,35
8	Матеріальні витрати, грн	651	651
9	Електроенергія, грн	840	1009.2
10	Амортизація, грн	4430,75	6962,60
11	Накладні витрати, грн	9147	14753
12	Разом на ін. витрати (8+9+10+11)	15177,3	23375,8
13	Собівартість, грн	40089,45	64041,95
14	Прибуток, грн	10824,15	17291,33
15	Вартість розробленого ПЗ	50913,60	81333,27
16	Економічна ефективність	0,27	0,27
17	Термін окупності, років	3,7	3,7
18	Собівартість модернізації, грн	22049,20	16010,48
19	Супровід і модернізація, грн	28002,50	20333,32

Загальна вартість пропонованих робіт по розробці програмного продукту становить 78916,10 грн. для першого варіанту та 101666,60 грн. для другого. Оскільки ефективність для обидвох проектів відповідно до встановленого рівня прибутку становить 0,27, що є високим показником, то проводити дані роботи варто і вкладені кошти окупляться за 3,7 року.

При використанні процедурного підходу зменшується кількість працівників, які залучаються у проект, та зменшуються витрати на реалізацію проекту, але для підтримки проекту і його подальшої модернізації все ж в майбутньому потрібні набагато більші витрати. Для об'єктно-орієнтованого підходу потрібна більша кількість працівників, часу і коштів, але на модернізацію і підтримку в загальному потрібно менше ресурсів, і у висновку програма виконана по функціональній парадигмі стає дешевшою для споживача, і приносить економію ресурсів для розробників.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

У логістичних компаніях для робітників забезпечені певні інстркуції з охорони праці. Інструкція з охорони праці для робітника є обов'язковим для виконання нормативним документом для робітників, що працюють у диспетчерській службі підприємства. Інструкція розроблена на основі ДНАОП 0.00-8.03-93 "Порядок опрацювання та затвердження власником нормативних актів про охорону праці, що діють на підприємстві", ДНАОП 0.00-4.15-98 "Положення про розробку інструкцій з охорони праці", НПАОП 0.00-4.12-05 "Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці". Строк перегляду інструкції 1 раз на 3 роки. За даною інструкцією робітник диспетчерської служби інструктується перед початком роботи (первинний інструктаж), а потім через кожні 6 місяців (повторний інструктаж). Результати інструктажу заносяться до «Журналу реєстрації інструктажів з питань охорони праці».

Основні обов'язки робітника компанії:

- забезпечує ефективну роботу підприємства в частині інформаційного обміну між клієнтами підприємства та відділом логістики;

- приймає замовлення на товар (назва, кількість та ін.) від клієнтів безпосередньо або через агентів торговельних по телефону або електронною поштою;

- повідомляє одержані дані про замовлення товару телефоном або електронною поштою менеджеру з транспортно-експедиторської діяльності.

Робоче місце робітника служби створене на підприємстві для працевлаштування інвалідів, в тому числі з фізичними вадами. Робоче місце знаходиться в адміністративному приміщенні, для операторів інвалідів - за місцем проживання (вдома). Графік роботи встановлено згідно Правил внутрішнього трудового розпорядку та трудової угоди. До самостійного виконання робіт за професією допускаються особи, які:

- досягли 18 років;

- пройшли медичний огляд відповідно до Закону України «Про охорону праці» та Порядку проведення медичних оглядів працівників певних категорій, затвердженого наказом Міністерства охорони здоров'я України від 21.05.2007 № 246 і не мають медичних протипоказань;

- пройшли вступний інструктаж з охорони праці, інструктаж на робочому місці, інструктаж з пожежної безпеки;

- пройшли інструктаж з електробезпеки в обсязі І-ої кваліфікаційної групи.

Правила охорони праці під час експлуатації електронно-обчислювальних машин поширюються на всі підприємства, установи, організації, юридичні особи, незалежно від форми власності, відомчої належності, видів діяльності та на фізичних осіб, які здійснюють розробку, виробництво та застосування електронно-обчислювальних машин і персональних комп'ютерів (ЕОМ), у тому числі й на тих, які мають робочі місця, обладнані ЕОМ, або виконують обслуговування, ремонт та налагодження ЕОМ. Облаштування робочих місць, обладнаних відеотерміналами, враховує:

- належні умови освітлення приміщення і робочого місця, відсутність відблисків;
- оптимальні параметри мікроклімату;
- м'яке рентгенівське випромінювання;
- наявність шуму та вібрації;
- електромагнітне випромінювання;
- ультрафіолетове та інфрачервоне випромінювання;
- електростатичне поле між екраном і оператором.

Для всіх споруд і приміщень, в яких експлуатуються відеотермінали та ЕОМ, визначається категорія з вибухопожежної і пожежної безпеки відповідно до ДСТУ Б В.1.1-36:2016 "Визначення категорій приміщень, будинків, установок за вибухопожежною та пожежною небезпекою".

Виробничі приміщення, в яких розташовані ЕОМ, не повинні межувати з приміщеннями, де рівні шуму та вібрації перевищують норму. Робочі місця з відеотерміналами або персональними ЕОМ у приміщеннях з джерелами шкідливих

виробничих факторів розміщуються в ізольованих кабінах з обладнаним повітрообміном. Площу приміщень, в яких розташовують відеотермінали, визначають згідно з чинними нормативними документами з розрахунку на одне робоче місце, обладнане відеотерміналом: площа – не менше 6,0 м^2, обсяг – не менше 20,0 м^3, з урахуванням максимальної кількості осіб, які одночасно працюють у зміні.

Розрахунок освітленості для приміщеня, де здійснювалась розробка логістичної системи виконаємо методом коефіцієнта використання світлового потоку. Цей метод використовується для розрахунку загального рівномірного висвітлення горизонтальних поверхонь виробничих приміщень при відсутності затемнень. Розрахунок висвітлення методом коефіцієнта використання виконується за формулою:

$$\Phi = \frac{E \times S \times k \times z}{N \times \eta}$$

де Φ – необхідний світловий потік ламп у кожнім світильнику, лм; E – нормативна мінімальна освітленість, лк, з довідкової літератури; k – коефіцієнт запасу, вибирається з довідкової літератури; S – освітлювана площа, кв м; z – коефіцієнт мінімальної освітленості, величина якого знаходиться в межах від 1,1 до 1,5 (при оптимальних відносинах відстані між світильниками до розрахункової висоти для ламп розжарювання і ДРЛ z=1,15 і для люмінесцентних ламп z = 1,1); N – число світильників у приміщенні; η – коефіцієнт використання світлового потоку. Приймаємо: E=400 лк; k =1,5; z=1,1.

Освітлювана площа приміщення визначається за формулою:

$$S = A \times B$$

де S – освітлювана площа, кв м; A – довжина приміщення, м; B – ширина приміщення, м. Отримуємо $S = 5 \times 3 = 15$ м^2.

Розміщення світильників у приміщенні при системі загального висвітлення залежить від розрахованої висоти їхнього підвісу h, що звичайно задається розмірами приміщень. Найбільш вигідне співвідношення відстані між світильниками до розрахункової висоти підвісу приймається за таблицею в довідковій літературі у залежності від типової кривої сили світла світильника. Для люмінесцентних ламп при косинусоїдальній типової кривої вибираємо a = 1,4.

Знаходимо розрахункову висоту підвісу за формулою:

$$h = H - h_з - h_p,$$

де H – висота приміщення, м; $h_з$ – висота звису світильника (від перекриття), м; h_p – висота робочої поверхні над підлогою, м. Приймаємо: Н=3,9 м, $h_з$=0,7 м, h_p =0,8 м. Тоді h =3,9– 0,7 – 0,8=1,4 м.

Відстань між світильниками визначаємо за формулою:

$$L = \lambda \times h, \quad L = 1{,}4 \times 0{,}4 = 2{,}5м$$

Визначаємо кількість світильників для установки в приміщенні:

$$N = \frac{S}{L^2}$$

Отже, $N = \frac{15}{0{,}96^2} = 1{,}6,$ приймаємо N = 2 шт.

Для визначення коефіцієнта використання η знаходимо індекс приміщення *i* за формулою:

$$i = \frac{A \times B}{h \times (A + B)},$$

де А і В – відповідно довжина і ширина приміщення, м; h – розрахункова висота підвісу, м. Отримуємо i = 0,78. Оцінюємо коефіцієнти відображення поверхонь приміщення: стелі – ρ_i, стін – ρ_n, робочої поверхні – ρ_p. Приймаємо згідно довідкової літератури: ρ_i = 70%, ρ_n = 50%, ρ_p = 30%. За отриманим значенням *i* і ρ визначаємо величину коефіцієнта використання світлового потоку для обраного світильника.

Вибираємо світильник типу ПВЛМ–Д, для якого η=73%. За формулою визначаємо необхідний світловий потік ламп у кожнім світильнику:

$$\Phi = \frac{400 \times 15 \times 1{,}5 \times 1{,}1}{2 \times 0{,}73} = 6781\ лм.$$

Опираючись на вищезроблені розрахунки можна зробити висновок, що освітення робочого місця є достатнім для роботи з логістичною системою. Виконання усіх перекислених вимог зробить роботу безпечною.

4.2 Безпека в надзвичайних ситуаціях

На основі вивчення факторів, які впливають на стійкість роботи об'єктів, і оцінки стійкості елементів і галузей виробництва проти уражаючих факторів ядерної, хімічної і біологічної зброї, стихійних лих і виробничих аварій, необхідно завчасно організувати і провести організаційні, інженерно-технічні й технологічні заходи для підвищення стійкості роботи.Здійснення організаційних заходів передбачає завчасну підготовку всіх структур цивільного захисту, служб і формувань до надзвичайних ситуацій.

Вжиттям технологічних заходів підвищується стійкість роботи об'єктів шляхом змінювання технологічних процесів, режимів, можливих в умовах надзвичайних ситуацій. Інженерно-технічні заходи мають забезпечити підвищену стійкість виробничих споруд, технологічних ліній, устаткування, комунікацій об'єкта до впливу уражаючих факторів під час надзвичайних ситуацій. При проведенні цих заходів необхідно враховувати конкретні умови об'єкта. Проте є загальні організаційні інженерно-технічні заходи, які мають проводитись на всіх об'єктах.

Забезпечення захисту людей та їх життєдіяльності. Створення на об'єкті надійної системи оповіщення про загрозу нападу противника, радіоактивне забруднення, хімічне і біологічне зараження, загрозу стихійного лиха і виробничої аварії. Створення фонду захисних споруд ЦО, запасів засобів індивідуального захисту і забезпечення своєчасної видачі їх населенню. Завчасна підготовка до масової санітарної обробки населення і знезаражування одягу, організація взаємодії з установами охорони здоров'я для медичного обслуговування населення у надзвичайних ситуаціях.

Захистити цінне і унікальне устаткування можна завдяки проведенню інженерно-технічних заходів, щоб зменшити небезпеку пошкодження і руйнування цінного й унікального устаткування, станків з програмним керуванням, шліфувальних, токарних, розточних, зубофрезер-них, пресових станків,

автоматичних конвеєрних ліній та іншого устаткування. Варіантами такого захисту є розміщення зазначеного устаткування в заглиблених приміщеннях а також використання спеціальних захисних пристосувань, закріплення станків на фундаментах, застосування контрфорсів для підвищення стійкості проти перекидання обладнання.

Стійкість роботи автотранспортної та іншої техніки, технологічного обладнання і механізмів. Організація своєчасного оповіщення гаража, технологічного парку, їх керівників, водіїв, механізаторів про загрозу надзвичайної ситуації.Підготовка автотранспортної техніки до проведення робіт в умовах радіоактивного забруднення, хімічного біологічного зараження і світломаскування.Пристосування і використання всіх видів транспортних засобів для евакуації населення і перевезення потерпілих. Розробка заходів з метою пристосування автотранспортної, іншої техніки для виконання завдань ЦЗ. Розробка пристосувань і технологічних процесів для відбору потужностей тракторів і автомобілів з метою приведення в дію електрогенераторів і технологічного обладнання, насосів для подачі води до місця споживання зі свердловин, відкритих водойм і шахтних колодязів. Підготовка всієї техніки для проведення рятувальних та інших невідкладних робіт у надзвичайних умовах мирного і воєнного часу.

Забезпечення стійкого постачання об'єкта. Для забезпечення виробництва продукції необхідні електроенергія, паливо, мастила, засоби захисту рослин, міндобрива, профілактичні й лікувальні препарати ветеринарної медицини, запасні частини, сировина та інші матеріально-технічні засоби. Забезпечення об'єктів цими ресурсами дасть можливість випускати необхідну продукцію в надзвичайних умовах мирного і воєнного часу. Тому повинні проводитись такі заходи, які б забезпечили стійкість постачання і сприяли підвищенню захисту мережі електро-, водо-, газопостачання, транспортних комунікацій і джерел постачання всім необхідним для забезпечення функціонування галузей сільського господарства в надзвичайних умовах. З метою попередження аварій на електричних мережах необхідно

встановити автоматичну систему відключення перенапруги. Повітряні лінії електропостачання слід замінити на підземно-кабельні. Газ використовується як паливо і на хімічних підприємствах у технологічному процесі. Для безперебійного забезпечення газом, газові мережі необхідно підводити до об'єкта з двох напрямків, які мають бути з'єднані в єдине кільце з обладнанням для можливого дистанційного автоматичного управління й у разі необхідності відключення пошкоджених ділянок. На великих підприємствах необхідно мати підземні ємності із закачаним резервним газом. На підприємствах, де використовується пара, необхідно захистити джерела його постачання, заглибити в ґрунт комунікації паропо-стачання і встановити запірні пристосування. Запас резервних матеріалів необхідно розраховувати на такі строки роботи підприємства, за які можливе відновлення регулярного постачання. Передбачити, на випадок перебоїв в постачанні підприємствами-суміжниками, створення місцевих матеріалів, сировини для виготовлення комплектуючих виробів і інструментів силами свого підприємства.

Забезпечення надійності системи управління і зв'язку. Організація захищеного пункту управління, оснащення його засобами зв'язку, які б дали можливість швидко доводити сигнали ЦЗ до всіх виробничих підрозділів і населення у місцях проживання. Розробка документів, які регламентують чіткі дії персоналу для забезпечення сталої роботи об'єкта в надзвичайних умовах. Підготовка необхідного резерву кадрів спеціалістів, механізаторів і керівних працівників для зміни тим, які будуть мобілізовані.

Планування збору даних про обстановку, передачу команд і розпоряджень в умовах впливу на об'єкт уражаючих факторів. Організація використання радіозасобів, телефонного зв'язку, посильних для зв'язку з віддаленими населеними пунктами, виробничими підрозділами, забезпечення дублювання ліній і каналів зв'язку.

Враховуючи все вищесказане можна зробити висновок, що для підвищення стійкості роботи автотранспортного підприємства у воєнний час застосовуються

окремі і загальні правила при надзвичайних ситуаціях. Також варто зазначити, що ключовим моментом є вчасне оповіщення персоналу про можливі загрози та своєчасна реакція на них.

Згідно чинного законодавства держава може вилучати транспортні засоби в осіб на безоплатній основі. Під час мобілізації задоволення потреб військових формувань здійснюється шляхом безоплатного залучення транспортних засобів і техніки підприємств, установ та організацій незалежно від форми власності на умовах повернення їх після оголошення демобілізації. Військові комісаріати здійснюють залучення транспортних засобів і техніки за лімітами вилучення транспортних засобів і техніки та відсотковими нормами вилучення, затвердженими Кабінетом Міністрів України, як окремо, так і у складі спеціальних формувань, які передаються до складу військових формувань під час мобілізації. Перелік транспортних засобів і техніки, які можуть призначатися для задоволення потреб військових формувань на особливий період, визначається Міноборони. У воєнний час вилучення транспортних засобів здійснюється шляхом передачі підприємствами, установами та організаціями, а також громадянами транспортних засобів і техніки для задоволення потреб військових формувань відповідно до визначених завдань.

ВИСНОВКИ

Дана робота присвячена створенню інформаційної системи для логістичної компанії. Дана система є базою даних, що допомагає диспетчерам та працівникам логістичних фірм оптимізувати та прискорити їх робочий процес. В ході розробки було використано модель бази даних, яка дозволила максимізувати свої переваги, завдяки чому було досягнуто хороших результатів. Система в цілому получилась добре, але є речі які потребують доробки, а також можливо реалізувати декілька нових функцій для розширення функціоналу системи. Результатами розробки я залишився задоволений.

ПЕРЕЛІК ПОСИЛАНЬ

1. Rumbaugh K. The unified modeling language reference manual [Text] / J. Rumbaugh, I. Jacobson, G. Booch – Boston. : Addison Wesley Longman Inc., 1999 – p 26. - ISBN 0-201-30998-X.
2. [Електронний ресурс] – Режим доступу:http://moodle.ipo.kpi.ua/moodle/mod/resource/view.php?inpopup=true&id=44416
3. Stellman, Andrew; Greene, Jennifer (2005). Applied Software Project Management. O'Reilly Media. ISBN 978-0-596-00948-9
4. [Електронний ресурс] – Режим доступу:https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/
5. Ethan Brown Web Development with Node and Express: Leveraging the JavaScript Stack (2019) O'Reilly Media. ISBN 1492053481, 9781492053484 http://www.vanmeegern.de/fileadmin/user_upload/PDF/Web_Development_with_Node_Express.pdf
6. [Електронний ресурс] – Режим доступу: https://xbsoftware.ru/blog/zhiznennyj-tsykl-razrabotki-spiral
7. [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm
8. [Електронний ресурс] – Режим доступу: http://org2.knuba.edu.ua/pluginfile.php/8977/mod_resource/content/2/cli-se.pdf
9. Eurostat - Tables, Graphs and Maps Interface (TGM) table [Електронний ресурс] – 2018 – Режим доступу: https://ec.europa.eu/eurostat/tgm/table.do?tab=table&init=1&plugin=1&language=en&pcode=tec00120
10. [Електронний ресурс] – Режим доступу: http://kharchuk.ru/JavaScript.pdf
11. [Електронний ресурс] – Режим доступу: https://pidruchniki.com/11221213/bzhd/osnovni_napryamki_pidvischennya_stiykosti_roboti_obyekta_nadzvichaynih_situatsiyah

12. Stellman, Andrew; Greene, Jennifer (2005). Applied Software Project Management. O'Reilly Media. ISBN 978-0-596-00948-9
13. Eurostat - Tables, Graphs and Maps Interface (TGM) table [Електронний ресурс] – 2018 – Режим доступу: https://ec.europa.eu/eurostat/tgm/table.do?tab=table&init=1&plugin=1&language=en&pcode=tec00120
14. ECMAScript Language Overview (PDF). 2007-10-23. с. 4. Процитовано 2009-05-03.19. About npm [Електронний ресурс]. – Режим доступу: https://docs.npmjs.com/about-npm/
15. [Електронний ресурс] – Режим доступу: https://pidruchniki.com/81326/tehnika/sistemi_keruvannya_bazami_danih
16. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/MongoDB
17. [Електронний ресурс] – Режим доступу: https://metanit.com/nosql/mongodb/1.1.php
18. [Електронний ресурс] – Режим доступу: https://github.com/rinde/RinSim/blob/master/core/src/main/java/com/github/rinde/rinsim/core/Simulator.java
19. Моделі і методи проектування інформаційних систем Проектування інформаційних систем Тема 1 - Проектування ІС [Електронний ресурс]. – Режим доступу: https://elearning.sumdu.edu.ua/free_content/lectured:de1c9452f2a16143939 1120eef364dd8ce4d8e5e/20151203140326/165292/index.html
20. James Rumbaugh, Ivar Jacobson, Grady Booch (1999). The unified modeling language reference manual. Addison Wesley Longman Inc. ISBN 0-201- 30998-X.
21. ISO/IEC 2382:2015, Information technology — Vocabulary — Part 1: Terms and definitions: «database: collection of data organized according to a conceptual structure describing the characteristics of these data and the relationships among their corresponding entities, supporting one or more application areas»
22. Закон України «Про внесення змін до Податкового кодексу України та деяких законодавчих актів України щодо податкової реформи» від 28 грудня 2014

року № 71-VIII. – [Електронний ресурс] – Режим доступу: http://portal.rada.gov.ua/

23. Гандзюк М. П. Основи охорони праці: підручник. - 4-те вид. / М. П. Гандзюк, Є. П. Желібо, М. О. Халімовський ; за ред. М. П. Гандзюка. - К. : Каравела, 2008.
24. Жидецький В.Ц. Основи охорони праці. Підручник. – Львів: Афіша, 2004.
25. Методичні вказівки для виконання розділу дипломної роботи щодо техніко-економічного обґрунтування вибору проектного рішення розробки та оцінки якості програмного забезпечення / Упор. Петрик М.Р., Кінах Я.І., Головатий А.І., Рогатинська Л.Р. – Тернопіль: Вид-во ТНТУ ім. І.Пулюя –2013.-34с.
26. Методичні вказівки до виконання дипломної роботи освітнього рівня “магістр” студентами усіх форм навчання для напряму підготовки 121 – “Інженерія програмного забезпечення” / Укладачі: Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ
Кафедра “Програмної інженерії”

ЗАТВЕРДЖУЮ
Завідувач кафедрою
програмної інженерії
Петрик Михайло Романович
“___ “_____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання проекту

“ Розробка інформаційної системи для організації логістичних перевезень з використанням клієнт-серверної архітектури **”**

Розробники
Керівник проекту
д.ф.-м.н. проф. Петрик М.Р.
19 “червня” 2019 р.

Виконавець:
студент Корнієнко В.О.
19 “червня” 2019 р.

Тернопіль 2019

ЗМІСТ

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану на 2019 р. з курсу «Методологія та технологія створення складних програмних систем».

Тема проекту: «Розробка інформаційної системи для організації логістичних перевезень з використанням клієнт-серверної архітектури».

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Програмний продукт призначений для полегшення роботи диспетчерів та працівників логістичних фірм. Дане програмне забезпечення буде розроблене для ПК.

Мета розробки полягає в створені програмного забезпечення, що дозволить користувачам, оптимізувати їх роботу, шляхом полегшення пошуку та сортування замовлень.

Даний продукт дозволить користувачам значно економити час при пошуку необхідного програмного продукту.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Функціональні характеристики

Програмний продукт має забезпечити виконання наступних дій:

- Відображати повний список замовлень;
- Створення замовлень та їх редагування у будь-який час;
- Сортування замовлень по категоріях, для спрощення пошуку;
- Видалення непотрібних чи неактуальних замовлень;
- Можливість імпорту чи експорту замовлень з бази даних.

3.2 Склад та параметри технічних засобів

Функціонування програми забезпечується: ПК, процесор з розрядністю x64/86, з 1 Гб оперативної пам‘яті, встановленою системою Windows XP/7/10, та не менш 350 Мб вільного місця на жорсткому диску.

3.3 Інформаційна та програмна сполучність

Програмний продукт повинен коректно функціонувати під керуванням ОС Microsoft Windows XP/7/10. Програмний продукт має назву “Logic soft ” і повинен бути написаний мовою Javascript.

4. СТАДІЇ РОЗРОБКИ

- - аналіз предметної галузі;
- - ознайомлення з програмною документацією;
- - розробка програмної моделі та бази даних;
- - тестування програмного продукту;
- - оформлення супровідної документації;
- - здача продукту.

5. ПРОГРАМНА ДОКУМЕНТАЦІЯ

Для програмного продукту повинні бути розроблені наступні документи:

- Технічне завдання;

- Пояснювальна записка;

6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Розроблений програмний продукт має виконувати всі вимоги, що складаються з перерахованих у п.3.1 характеристик.

Приймання проводиться спеціально створеною комісією в термін до

“__” _____ 2019 р.

УДК 004.422.83

В.О. Корнієнко – магістрант

Тернопільський національний технічний університет імені Івана Пулюя, Україна

М.Р. Петрик –доктор фізико-математичних наук, професор

Тернопільський національний технічний університет імені Івана Пулюя, Україна

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ОРГАНІЗАЦІЇ ЛОГІСТИЧНИХ ПЕРЕВЕЗЕНЬ З ВИКОРИСТАННЯМ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

Бурхливий розвиток галузі вантажних перевезень є головною причиною формування запиту у сфері технічних засобів та комунікацій для створення відповідних програмних рішень і формуванні у суспільстві чіткого розуміння всієї важливості та актуальності побудови схем маршрутного планування і діяльності забезпечення розвитку руху комунікацій між постачальниками логістичних послуг та вантажними перевізниками, які працюють в умовах високої навантаженості і загальної напруги. На водія вантажних перевезень лягає непроста ноша здійснення комунікацій та власне самого перевезення і для більш злагодженої координації у сфері використовуються мобільні групи операторів, які забезпечують надійний тил для подальшої роботи групи.

Останніми роками, попри всі труднощі, використання елементів логістики транспорту постійно розширюється завдяки, наприклад, введенню в дію локальної мережі ВМ, інформації про переміщення вантажів у транспортних потоках процесів, введенню нових методів бухгалтерського обліку матеріальних коштів, які проходять разом з вантажопотоками через транспортні підприємства [1]. Цілком природно, що в таких умовах виникає потреба в надійних і високопродуктивних інструментах з допомогою яких оператор вантажних перевезень зможе швидко та якісно виконувати свої завдання, які допоможуть координувати подальші дії водія вантажних автомобілів.

Протягом останніх років бурхливо розвиваються засновані на інформатиці нові логістичні технології. Інформаційні системи займають у цих технологіях центральне положення. Підприємство є відкритою системою, що матеріальним і інформаційним потоками зв'язана з постачальниками, споживачами, експедиторами і транспортними організаціями. [2]. Питання якісної побудови маршрутних карт та фіксації часових швидкостей є ключовим не тільки при розумінні важливості розсередження та розподлі навантаженостей між різними учасниками процесу формування і налагодження орієнтації в полі часового планування та оптимізації графічних частин устаткування на картах і в графіках водіїв.

Підсумовуючи вище сказане можна зробити висновок, що розробка інформаційної системи для логістичних компаній є хорошим рішення. Це дозволить зменшити навантаження на операторів та структуризувати і оптимізувати процес роботи.

Література

1. Кальченко А. Г. Логістика : підручник / А. Г. Кальченко. – К. : КНЕУ, 2004. – 284 с. Режим доступу: http://studentbooks.com.ua/content/view/126/76/1/45/#532474

2. Інформаційна система в логістиці: створення логістичних інформаційних систем [Електронний ресурс] – Режим доступу: http://ru.osvita.ua/vnz/reports/logika/25336/

```
/*
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package com.github.rinde.rinsim.core;

import static com.google.common.base.Preconditions.checkArgument;

import java.util.LinkedHashSet;
import java.util.Set;

import javax.annotation.Nonnull;
import javax.measure.quantity.Duration;
import javax.measure.unit.SI;
import javax.measure.unit.Unit;

import org.apache.commons.math3.random.MersenneTwister;
import org.apache.commons.math3.random.RandomGenerator;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.github.rinde.rinsim.core.model.DependencyProvider;
import com.github.rinde.rinsim.core.model.Model;
import com.github.rinde.rinsim.core.model.Model.AbstractModel;
import com.github.rinde.rinsim.core.model.ModelBuilder;
import com.github.rinde.rinsim.core.model.ModelBuilder.AbstractModelBuilder;
import com.github.rinde.rinsim.core.model.ModelManager;
import com.github.rinde.rinsim.core.model.ModelProvider;
import com.github.rinde.rinsim.core.model.rand.RandomModel;
import com.github.rinde.rinsim.core.model.rand.RandomProvider;
import com.github.rinde.rinsim.core.model.time.ClockController;
import com.github.rinde.rinsim.core.model.time.TickListener;
import com.github.rinde.rinsim.core.model.time.TimeLapse;
import com.github.rinde.rinsim.core.model.time.TimeModel;
import com.github.rinde.rinsim.util.StochasticSuppliers;
import com.google.auto.value.AutoValue;
import com.google.common.collect.ImmutableSet;
```

```
/**
* Simulator is the core class of a simulation. It is responsible for managing
* time which it does by periodically providing {@link TimeLapse} instances to
* registered {@link TickListener}s. Further it provides methods to start and
* stop simulations. The simulator also acts as a facade through which
* {@link Model}s and objects can be added to the simulator.
*
* The configuration phase of the simulator looks as follows:
* <ol>
* <li>Use {@link Simulator#builder()} to register models and define other
* global settings.</li>
* <li>Obtain a {@link Simulator} instance via {@link Simulator.Builder#build()}
* .
* <li>register objects using {@link #register(Object)}</li>
* <li>start simulation by calling {@link #start()}</li>
* </ol>
*
* @author Rinde van Lon
* @author Bartosz Michalik
*/
public final class Simulator implements SimulatorAPI {
private static final Logger LOGGER = LoggerFactory.getLogger(Simulator.class);

private final ModelManager modelManager;
private final ClockController clock;
private final RandomModel rand;
private final Set<Object> toUnregister;

Simulator(Builder b) {
modelManager = b.mmBuilder
.add(SimulatorModelBuilder.create(this))
.addDefaultProvider(
RandomModel.builder().withRandomGenerator(
StochasticSuppliers.constant(b.rng)))
.addDefaultProvider(
TimeModel.builder()
.withTickLength(b.tickLength)
.withTimeUnit(b.timeUnit))
.build();
toUnregister = new LinkedHashSet<>();
clock = modelManager.getModel(TimeModel.class);
rand = modelManager.getModel(RandomModel.class);
}

/**
* @param m The model.
* @return False.
```

```
* @deprecated To add a {@link Model} use {@link #builder()} instead.
* @throws UnsupportedOperationException is always thrown.
*/
@SuppressWarnings("static-method")
@Deprecated
public boolean register(Model<?> m) {
throw new UnsupportedOperationException(
"Models can be added via Simulator.builder().");
}

@Override
public void register(Object obj) {
LOGGER.info("{} - register({})", clock.getCurrentTime(), obj);
modelManager.register(obj);
}

/**
* {@inheritDoc} Unregistration from the models is delayed until all ticks are
* processed.
*/
@Override
public void unregister(Object o) {
if (o instanceof Model<?>) {
throw new IllegalArgumentException("can not unregister a model");
}
toUnregister.add(o);
}

void checkUnregister() {
if (!toUnregister.isEmpty()) {
for (final Object o : toUnregister) {
modelManager.unregister(o);
}
toUnregister.clear();
}
}

/**
* Returns all models registered in the simulator.
* @return immutable list of models.
*/
public ImmutableSet<Model<?>> getModels() {
return modelManager.getModels();
}

/**
* Returns the {@link ModelProvider} that has all registered models.
* @return The model provider
*/
```

```
public ModelProvider getModelProvider() {
return modelManager;
}

/**
* @return The current simulation time.
*/
public long getCurrentTime() {
return clock.getCurrentTime();
}

/**
* @return The time step (in simulation time) which is added to current time
* at every tick.
*/
public long getTimeStep() {
return clock.getTickLength();
}

/**
* Adds a tick listener to the simulator.
* @param listener The listener to add.
*/
public void addTickListener(TickListener listener) {
register(listener);
}

/**
* Removes the listener specified.
* @param listener The listener to remove
*/
public void removeTickListener(TickListener listener) {
unregister(listener);
}

/**
* Start the simulation.
*/
public void start() {
if (modelManager.getUserInterface().isPresent()) {
modelManager.getUserInterface().get().show();
} else {
clock.start();
}
}

/**
* Either starts or stops the simulation depending on the current state.
*/
```

```
public void togglePlayPause() {
if (!clock.isTicking()) {
start();
} else {
stop();
}
}

/**
* Stops the simulation.
*/
public void stop() {
clock.stop();
}

/**
* Advances the time a single tick.
*/
public void tick() {
clock.tick();
}

/**
* @return true if simulator is playing, false otherwise.
*/
public boolean isPlaying() {
return clock.isTicking();
}

/**
* Get access to the main random generator used in the simulator.
* @return the random generator of the simulator
*/
@Override
public RandomGenerator getRandomGenerator() {
return rand.get(RandomProvider.class).masterInstance();
}

/**
* @return The unit of time that is used for generating ticks.
*/
public Unit<Duration> getTimeUnit() {
return clock.getTimeUnit();
}

/**
* @return A new {@link Builder} for creating a {@link Simulator} instance.
*/
public static Builder builder() {
```

```
return new Builder();
}

/**
* A builder for {@link Simulator}.
* @author Rinde van Lon
*/
public static class Builder {
static final long DEFAULT SEED = 123L;
static final long DEFAULT TICK LENGTH = 1000L;

RandomGenerator rng;
Unit<Duration> timeUnit;
long tickLength;
ModelManager.Builder mmBuilder;

Builder() {
rng = new MersenneTwister(DEFAULT SEED);
timeUnit = SI.MILLI(SI.SECOND);
tickLength = DEFAULT TICK LENGTH;
mmBuilder = ModelManager.builder();
}

/**
* Sets the random seed used in the {@link RandomGenerator} used in the
* simulator. The default {@link RandomGenerator} is a
* {@link MersenneTwister} with seed <code>123</code>.
* @param seed The seed to use.
* @return This, as per the builder pattern.
*/
public Builder setRandomSeed(long seed) {
rng.setSeed(seed);
return this;
}

/**
* Sets the {@link RandomGenerator} to use in the simulator. This overwrites
* any previous calls made to {@link #setRandomSeed(long)}. The default
* {@link RandomGenerator} is a {@link MersenneTwister} with seed
* <code>123</code>.
* @param randomGenerator The generator to set.
* @return This, as per the builder pattern.
*/
public Builder setRandomGenerator(RandomGenerator randomGenerator) {
rng = randomGenerator;
return this;
}

/**
```

```
* Sets the time unit to use in the simulator. The default time unit is
* milliseconds.
* @param unit The time unit to use.
* @return This, as per the builder pattern.
*/
public Builder setTimeUnit(Unit<Duration> unit) {
timeUnit = unit;
return this;
}

/**
* Sets the length of a single tick in the simulator. The default tick
* length is <code>1000</code>.
* @param length The tick length to set.
* @return This, as per the builder pattern.
*/
public Builder setTickLength(long length) {
checkArgument(length > 0,
"Tick length must be strictly positive but is %s.", length);
tickLength = length;
return this;
}

/**
* Adds the specified {@link ModelBuilder} to the simulator. The
* {@link ModelBuilder} will be used to obtain a {@link Model} instance that
* will be added to the simulator.
* @param builder The builder to add.
* @return This, as per the builder pattern.
*/
public Builder addModel(ModelBuilder<?, ?> builder) {
mmBuilder.add(builder);
return this;
}

/**
* Adds the specified {@link ModelBuilder}s to the simulator. Each
* {@link ModelBuilder} will be used to obtain a {@link Model} instance that
* will be added to the simulator.
* @param builders The builders to add.
* @return This, as per the builder pattern.
*/
public Builder addModels(Iterable<? extends ModelBuilder<?, ?>> builders) {
for (final ModelBuilder<?, ?> b : builders) {
addModel(b);
}
return this;
}
```

```
/**
* Builds the simulator, at least one {@link Model} must have been added.
* @return A new {@link Simulator} instance.
*/
public Simulator build() {
return new Simulator(this);
}
}

@AutoValue
abstract static class SimulatorModelBuilder extends
AbstractModelBuilder<SimulatorModel, SimulatorUser> {

private static final long serialVersionUID = -2639242259656135434L;

SimulatorModelBuilder() {
setProvidingTypes(SimulatorAPI.class);
}

abstract Simulator getSimulator();

@Override
public SimulatorModel build(DependencyProvider dependencyProvider) {
return new SimulatorModel(getSimulator());
}

static SimulatorModelBuilder create(Simulator sim) {
return new AutoValue Simulator SimulatorModelBuilder(sim);
}
}

static class SimulatorModel extends AbstractModel<SimulatorUser>
implements TickListener {
final Simulator simulator;

SimulatorModel(Simulator sim) {
simulator = sim;
}

@Override
public boolean register(SimulatorUser element) {
element.setSimulator(simulator);
return true;
}

@Override
public boolean unregister(SimulatorUser element) {
return true;
}

@Override
```

```
@Nonnull
public <U> U get(Class<U> clazz) {
checkArgument(clazz == SimulatorAPI.class);
return clazz.cast(simulator);
}

@Override
public void tick(TimeLapse timeLapse) {}

@Override
public void afterTick(TimeLapse timeLapse) {
simulator.checkUnregister();
}
}
```

}