МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерних наук

ПОЙДИЧ Владислав Сергійович

Математичне та програмне забезпечення для заповнення сховища завдань у системі тайм-менеджменту/ Mathematical tools and software for filling the bank of tasks in time-management system

спеціальність: 8.05010301 - Програмне забезпечення систем
магістерська програма - Програмне забезпечення систем

Магістерська робота

Виконав студент групи ПЗСм-21
В. С. Пойдич
_____

Науковий керівник:
к.т.н., доцент СТРУБИЦЬКА І.П.
_____

Магістерську роботу допущено
до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ТЕРНОПІЛЬ - 2016

РЕЗЮМЕ

Дипломна робота на тему: «Математичне та програмне забезпечення для процесу заповнення сховища даних для системи тайм менеджменту» складається з трьох розділів та виконана на ‘N’ сторінці.

Об’єктом дослідження є процеси аналізу зовнішніх даних, та генерація на їх основі критерій для подальшого використання системою пошуку.

Предметом дослідження є методи та програмні засоби генерації критерій для вирішення задач ефективного заповнення сховища даних для системи тайм менеджменту.

Методи дослідження ґрунтуються на застосуванні системного аналізу, функціонального моделювання, семантичних мереж, ІОВ дерев, теорії алгоритмів та об’єктно-орієнтованого проектування.

Наукова новизна одержаних результатів. Удосконалено алгоритм пошуку даних в умовах невизначеності на основі згенерованого дерева з ІОБ вказівниками.

Практичне значення одержаних результатів. Запропоновані методи дали можливість ефективніше заповнювати сховище даних задачами, для більш автоматизованого процесу планування часу, що в свою чергу призводить до менших затрат користувачів на процес монотонного набору даних і конфігурацію системи під себе.

Ключові слова: рекомендації, чанки, об’єктно-рольове моделювання, самонавчання, тайм менеджмент, дерева, чінки.

SUMMARY

Thesis: "Mathematical and programming software for process of filling data banks in time management system" consists of three sections and is made on ‘N’ pages.

The object of research is the process of analyzing external data and generation of criterias based on the analysis, with further use of search system.

The subject of research are methods and software for generation of criterias to solve the task of effective data bank filling for time management system.

Methods are based on using system analysis, functional modelling, semantic networks, IOB trees, algorithms theory and object oriented programming.

Scientific novelty of results. Improved algorithm of data search on generated sequence in conditions of uncertainty.

The practical significance of results. Provided software and methods give the possibility to fill data bank more effectively with tasks, for more automated process of time planning, that in turn provides less efforts for users to configure the system initially, and mostly removes the process of entering data from user side.

Keywords: recommendations, rankings, chunks, object-role modeling, learning, time management, trees, chinking.

ВСТУП

Актуальність теми. В нас час технології стрімко розвиваються, що призводить до все більшої кількості інформації яку потрібно обробляти. Багато людей користуються цією інформацією, але враховуючи її кількість потрібно дуже багато часу на те щоб освоїти більшість. Проте звісно не треба забувати про сім'ю та інші цінності. Системи тайм-менеджменту дозволяють максимально ефективно розподіляти свій час, проте потребують потужного налаштування і є мало автономними. Час це гроші, тому наприклад для бізнесменів кожна хвилина важлива, тому люди схильні до використання побічних програмних засобів для того щоб покращити планування свого часу.

Основний напрямок вирішення проблеми затрат часу є повна автоматизація системи. Користувачу не потрібно довго і монотонно заповнювати дані для своєї системи тайм менеджменту, вносити різні корегування, і навіть частково слідкувати за тим чи вдається йому йти згідно того що саме він налаштував. Тому виникає потреба у автоматизації системи, яка б на основні аналізу певних даних змогла автоматично підібрати найкращий графік організації часу для користувача.

Зв'язок роботи з науковими програмами, планами, темами. Напрямок виконаних досліджень безпосередньо пов'язаний з науково-дослідним напрямком кафедри комп'ютерних наук Тернопільського національного економічного університету.

Мета і задачі дослідження. Метою роботи є розробка системи для вирішення задач тайм менеджменту із використання можливості генерації рекомендацій, які б забезпечили підвищення формування планів на основі попередньо згенерованого списку задач та самонавчання системи, на основі вподобань до різних категорій.

Під **об'єктом дослідження** розуміємо процеси аналізу даних, та генерації на їх основі сховища завдань, яке буде в подальшому використане для формування пропозицій.

Предметом дослідження є математичне та програмне забезпечення для задачі тайм менеджменту для покращення ефективності планування часу.

Методи дослідження. Теоретичною основою магістерського дослідження виступають алгоритми структуризації даних, методи вирішення проблем пропущених значень та методи передбачень.

Наукова новизна одержаних результатів. В роботі удосконалено алгоритм пошуку даних в умовах невизначеності з використанням ІОБ вказівників та система автоматичного заповнення пропусків. На основі відгуків про знайдені результати система коригує подальший процес вибірки що сприяє автоматичному навчанню та удосконаленню процесу вибірки.

Практичне значення одержаних результатів полягає у більш ефективному процесі отримання результату по запитах на основі попередньо заповненого сховища даних.

Особистий внесок магістранта. Всі результати отримані автором самостійно.

Апробація результатів. Основні положення магістерського дослідження апробовані на VI Всеукраїнській школі-семінарі молодих вчених і студентів «Сучасні комп'ютерній інформаційні технології», що проходила 20-21 травня 2016 року у м. Тернополі на базі Тернопільського національного економічного університету.

Публікації. Поляруш О.В. Особливості використання Behavioral targeting для задачі тай менеджменту. / Струбицька І. П., Поляруш О. В. // Матеріали VI Всеукраїнської школи-семінару молодих вчених і студентів «Сучасні комп'ютерні інформаційні технології», АСІТ'2016 – Тернопіль: ТНЕУ, 2016. − с. 146-147.

Зміст

РОЗДІЛ 1
ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СИСТЕМИ ВИРІШЕННЯ ЗАДАЧІ ЗАПОВНЕННЯ СХОВИЩА ДАНИХ У ПРОБЛЕМІ TIME MANAGEMENT

1.1 Коротка характеристика об’єкту управління

Тайм менеджмент один з найважливіших критеріїв у бізнесі. Велика кількість інформації одночасно може пагубно вплинути на людину. Тому в період новітніх технологій потрібно максимально автоматизовувати процеси які більшою мірою є марудними, монотонними при ручному виконанні,але які є істотними для успішності в різних сферах діяльності. Розподіл часу використовується всюди, від простого нагадування про те що і коли треба зробити до потужних графіків. І як вже було сказано заповнення сховища даних є довгим процесом, що потребує великого вложення часу, який міг би бути використаний на інші справи.

Система планування на базі вподобань користувача розраховано на велику кількість користувачів, не тільки тому що вона може їх підтримувати, а і тому що чим більше користувачів тим більше даних з якими може працювати система, відповідно тим кращі подальші рекомендації та робота з ними. Даний програмний продукт надає користувачу можливість планування власного часу. Вона дозволяє користувачу розбирати свої задачі по категоріях, робити під-категорії, додавати ключові слова з можливістю пошуку за ними, створювати власний графік задач які повинні бути виконані, можливість перегляду задач друзів. Також система може автоматично пропонувати користувачу різні варіанти задач/графіків на основі його введених даних, та на основі глобального аналізу. Це дозволяє системі покращуватися, а користувачам тратити менше часу на процесу заповнення. Так система зможе проаналізувати різні категорії і на основі попередніх вподобань інших користувачів видати певну рекомендацію. Звісно при першому запуску системи вона починає надавати рекомендації відносно даних введених адміністратором, проте надалі кількість даних буде тільки зростати і сама система зможе ефективніше надавати

рекомендації. Враховуючи те що система дозволяє автоматично планувати свій час, рекомендації можуть надаватися в вдвох режимах, простий та розширений. Простий режим забезпечує просто рекомендацію щодо задачі, наприклад «Сходити на фільм». Розширений режим в той самий час надає список можливих фільмів з кінотеатрами які розміщені близько користувача та години на які ця задача була запланована виділені іншим кольором. Цей режим набагато сильніше виділяється якщо в користувача в системі є друг з схожими вподобаннями, або друг котрий вже навіть підтвердив похід на цей фільм на певну годину. Тоді спеціальне повідомлення буде виведено, щоб знотифікувати користувача про те що його друг/друзі також збираються туди. На основі відгуків користувачів про надані рекомендації система буде автоматично корегувати процес аналізу даних та підбору задач для сховища. Цим займається підсистема аналізу даних та заповнення сховища завдань.

Система аналізу даних та заповнення сховища даних є підсистемою основного додатку, та тісно взаємодіє з системою рекомендацій. Вона складається з декількох підсистем:

- Система глобального аналізу даних та заповнення глобального сховища.
- Система локального аналізу даних.
- Система генерації дерева рішень.
- Система оброки вхідних даних в умовах невизначеності для пошуку задач.

Ця підсистема являє собою складну конструкцію яка виконує роль мозку додатку. Вона аналізує показники задач, вхідні дані які можуть бути задані адміністратором, генерує дерева рішень на основі вхідних даних та заповнює сховище даних отриманими результатами. Адміністратор може сам надати системі джерело, яке буде в подальшому проаналізоване. Але основу в роботі системи складають дані введені користувачами та їх відгуки про отримані рекомендації.

В свою чергу для забезпечення максимального задоволення користувача, системи глобального та локального аналізу даних займаються процесом збору

інформації на різні теми, наприклад фільми, магазини, курорти тощо. В подальшому глобальна система зберігає аналізовані дані, коли локальна на основі глобальних даних пробує знайти схожі варіанти відносно поточного місцезнаходження користувача (день/ніч, країна, місто), що дозволяє ефективно знайти відповідності для рекомендованої задачі при розширеному пошуку.

1.2 Аналіз існуючих методів та алгоритмів для вирішення поставленої задачі.

Задача наповнення сховища даних полягає у аналізі великої кількості вхідних даних та генерації на їх основі критеріїв, генерації дерев рішень для поставленого критерію та виконання пошуку в умовах невизначеності.

Набір даних складається з сутностей і мережі що їх з’єднує. Кожна сутність асоціюється з ключовим словом що вказує її тим, наприклад деякі сутності асоціюються з ключовим словом «кіно», інші сутності з іншими ключовими словами. З розповсюдженням ЕОМ були запропоновані складніші машинні алгоритми, засновані на методі найменших квадратів: регресійний метод, метод головних компонент, покрокова регресія, метод багатовимірної лінійної екстраполяції, метод прогностичних змінних. Враховуючи той факт, що оцінки перших двох моментів повністю визначають оцінки регресії, багато авторів зосередилися на проблемі оцінювання коваріаційної матриці за даними з відсутніми значеннями. Одночасно з’ясувалася і деяка обмеженість методів, заснованих на методі найменших квадратів. Так, Уїлкинсон вказував, що якщо пропуски є тільки у відгуку, то при сумісному оцінюванні пропусків і коефіцієнтів регресії метод найменших квадратів вимагає викреслювати всі рядки з пропусками. Це приводить до неповного використання інформації, що міститься в даних. Основною проблемою в завданні транспонованої регресії є знаходження для кожного об’єкту якнайкращої функції з даного класу функцій і якнайкращої опорної групи об’єктів, по яких будується ця функція.

При проведенні аналізу даних на практиці обмежуються аналізом не всієї генеральної сукупності в цілому, а лише деякою вибірковою кількістю

спостережень. Аналізована вибірка повинна відповідати критеріям якісності і повноти. У реальності доводиться стикатися з ситуацією, коли деякі з властивостей одного або декількох об'єктів відсутні – виникає ситуація даних з пропусками, що значно ускладнює математичну обробку. Оскільки зсув основних характеристик, наприклад, статистичних таких, як математичне очікування або дисперсія, зростає прямо пропорційно кількості пропусків.

До виникнення пропусків в початкових даних може привести багато причин: наприклад, відсутність значень внаслідок якихось дрібних поломок устаткування, не пов'язаних з експериментальним процесом, або небажання респондента при проведенні опитування відповідати на питання про свої доходи.

На сьогодні існує декілька шляхів вирішення проблеми неповних даних:

- виключення некомплектних об'єктів з початкової вибірки. Даний підхід до проблеми можна охарактеризувати як некоректний, оскільки неповні дані несуть в собі нову інформацію, необхідну для дослідження, і тому їх важливо включати в аналіз;
- застосування спеціально розроблених математичних методів аналізу неповних даних, таких як метод зважування або метод максимальної правдоподібності і ЕМ-АЛГОРИТМ (при цьому значно зростає складність аналізу);
- відновлення пропусків (найбільш поширені методи заповнення за середнім і за регресією). В більшості випадків саме цей підхід вважається найбільш ефективним і зручним вирішенням проблеми.

Основним інструментом прикладної обробки даних служать пакети програм, бібліотеки і інші програмні продукти. Можна констатувати, що сучасне програмне забезпечення аналізу даних з пропусками в цілому знаходиться на початковому рівні. Практично всі програмні засоби, в яких передбачена можливість наявності пропусків в даних, містять лише прості методи – такі, як, наприклад, виключення некомплектних спостережень, заповнення пропусків середніми, заповнення за допомогою регресії або обчислення коваріаційної матриці і вектора середніх парними методами і т.д., тобто методи, які були

реалізовані ще в перших версіях пакетів SSP, IMSL, BMDP або Statistica. Проте, ці методи часто дають незадовільні результати.

Методи аналізу неповних даних можна умовно розбити на наступні групи.

Метод виключення некомплектних об'єктів. За відсутності у ряду об'єктів значень яких-небудь змінних некомплектні об'єкти вилучаються з аналізу. Підхід легко реалізується і може бути задовільним при малій кількості пропусків. Проте іноді він приводить до серйозних зсувів і звичайно не дуже ефективний. Головний недолік такого підходу обумовлений втратою інформації при виключенні неповних спостережень.

Методи із заповненням. При даному підході пропущені значення початкової вибірки заповнюються і одержані «повні» дані обробляються звичайними методами. Найчастіше використовуються наступні процедури заповнення пропусків.

Заповнення середніми. Підставляються середні присутніх значень. Метод безумовного середнього – найпростіший вид заповнення. Він полягає в оцінці відсутніх значень y_{ij} середнім $y_j^{(j)}$ за присутніми значеннями змінної Y_j. Середнє спостережуваних і підставлених значень рівне $y_j^{(j)}$ – оцінці методом доступних спостережень. Заповнення з упередженим підбором. Пропуски заповнюються значеннями, одержаними для іншого схожого об'єкту вибірки. Процедуру можна описати як метод, при якому підстановка вибирається для кожного пропущеного значення за оцінкою розподілу на відміну від заповнення пропусків середніми, коли підставляється середнє розподілу. У більшості додатків емпіричний розподіл задається присутніми значеннями, тому при заповненні з підбором підставляються різні значення з даних для схожих об'єктів без пропусків.

Найбільш часто використовувані методи: підстановка з підбором усередині груп і підбір найближчого сусіда. У першому випадку формуються групи, і пропуски в кожній групі заповнюються присутніми значеннями з неї ж. Заповнення з підбором широко поширене. Воно може включати дуже складні схеми відбору об'єктів. Хоча практика підтвердила переваги цього методу, літератури, присвяченої його теоретичним властивостям, явно недостатньо.

Другий підхід базується на введенні метрики d для вимірювання відстані між об'єктами, визначеного в просторі супутніх змінних, і виборі підстановки за об'єктом з присутнім значенням, найближчого до об'єкту з пропуском. Наприклад, позначимо $x_{i1}, \dots, x_{ij}$– значення J змінних, виміряних в нормованих шкалах, у об'єкта i з пропуском y_i. Визначимо відстань $d(i,k) = max\left|x_{ij} - x_{kj}\right|$ між об'єктами i і k. Можемо вибирати підстановку y_i з тих k-х об'єктів, у яких:

а) спостерігаються $y_k, y_{k1}, \dots, y_{kj}$;

б) $d(i,k)$ менша за деякий поріг d_0.

Кількість «кандидатів» – відповідних k-х об'єктів – можна вибирати, змінюючи d_0. Схеми найближчого сусіда вимагають значних обчислювальних витрат. Вони стали застосовуватися порівняно недавно.

Заповнення за допомогою регресії. Коли пропущені значення оцінюються за допомогою регресії на присутні для аналізованого об'єкту змінні. Зокрема, до цієї групи відноситься метод заповнення умовними середніми або так званий метод Бака. Метод є перспективнішим способом заповнення пропусків в порівнянні з попередніми методами. Він полягає в підстановці середніх, умовних за присутніми в спостереженні змінним і відноситься до модельних методів.

Якщо змінні $Y_1, \dots, Y_k$ розподілені за багатовимірному нормальному закону з середнім μ і коваріаційною матрицею $\sum$, то регресія пропущених значень в даному спостереженні лінійна за присутніми значеннями з коефіцієнтами, які є добре відомими функціями від μ і $\sum$. У методі, запропонованому Баком, спочатку оцінюють μ і $\sum$ за повними спостереженнями, а потім використовують ці оцінки для обчислення лінійної регресії пропущених змінних за присутніми для кожного спостереження. Підставляючи значення змінних, присутніх для даного спостереження, в регресійне рівняння, одержують прогноз пропущених змінних для цього спостереження. Обчислення регресійних рівнянь для різної структури пропусків може видатися важким, але на справді він відносно простий, якщо використовувати оператора згортки. Дані, заповнені за методом Бака, забезпечують розумні оцінки середніх, зокрема, якщо прийнятно припущення про нормальність спостережень.

Вибіркова коваріаційна матриця за заповненими даними занижує величину дисперсії і коваріації, хоч і не так сильно, як при підстановці безумовних середніх. Також серед методів із заповненням можна виділити: заповнення без підбору, багатократного заповнення, складені та інші методи.

Методи зважування. Рандомізовані виводи за даними вибіркових обстежень з пропусками побудовані на вагах плану, обернено пропорційних ймовірності вибору. Нехай y_i – значення змінної Y i-го об'єкту популяції. Тоді середнє популяції часто оцінюють величиною:

$$\sum_{i=1}^{N} \pi_i^1 y_i \ / \ \sum_{i=1}^{N} \pi_i^{-1} \tag{1.1}$$

де суми беруться за витягнутими об'єктами, π_i– ймовірність витягання i-го об'єкту, π_i^{-1} – вага плану i-го елементу. Методи зважування змінюють ваги, щоб врахувати відсутні значення. Оцінка (1.1) замінюється оцінкою:

$$\sum_{i=1}^{N} (\pi_i \rho_i)^{-1} y_i \ / \ \sum_{i=1}^{N} (\pi_i \rho_i)^{-1} \tag{1.2}$$

де суми беруться за витягнутими об'єктами, в яких немає пропусків, p_i– оцінка ймовірності присутності значення для i-го об'єкту (звичайно доля об'єктів вибірки з присутніми значеннями). Зважування пов'язане із заповненням середніми. Наприклад, якщо ваги плану постійні в підгрупах вибірки, то заповнення пропусків в кожній підгрупі середніми підгрупи і зважування присутніх значень за допомогою їх долі в кожній підгрупі ведуть до однакових оцінок середнього популяції, хоча оцінки вибіркової дисперсії різні, якщо тільки не використовуються поправки на заповнення середніми.

Методи, що базуються на моделюванні. Широкий клас методів ґрунтується на побудові моделі породження пропусків. Виводи одержують за допомогою

функції правдоподібності, побудованої за умови справедливості цієї моделі, з оцінюванням параметрів методами типу максимальної правдоподібності.

У методах, що використовують функцію правдоподібності, реалізована відносно стара ідея обробки неповних даних:

- заповнення пропусків оцінками пропущених значень;
- оцінювання параметрів;
- повторне оцінювання пропущених значень (оцінки параметрів вважаються точними);
- повторне оцінювання параметрів і так далі до збіжності процесу.

Переваги такого підходу полягають в тому, що він гнучкий; дозволяє відмовитися від методів, розроблених для окремих випадків; дозволяє оцінювати в наближенні великої вибірки дисперсії оцінок за допомогою матриці інших похідних функцій правдоподібності для неповних даних; забезпечує надійну збіжність, тобто в певних нестрогих умовах кожна ітерація збільшує логарифм правдоподібності і послідовність сходиться до деякого стаціонарного значення. Недолік алгоритму полягає в тому, що швидкість збіжності може бути дуже низькою, якщо пропущено багато даних.

Розглянемо завдання з пропусками в залежній змінній. У багатьох завданнях повністю присутні значення $p > 1$ змінних для всіх n об'єктів. Такі дані можна представити як на рисунку 1.3, де X є матрицею , а пропущені значення представляють $(n - m)$ об'єктів Y. При звичайному аналізі неповних даних використовують припущення, що дані відсутні випадково, тобто припускають, що ймовірність пропуску y_i може змінюватися в залежності від змінних плану, але при даному значенні x_i, i-і рядки X, ймовірність відсутності y_i не залежить від y_i. У практичних додатках слід перевіряти допустимість такого припущення. Аналіз будується так, щоб використовувати «майже збалансованість» одержуваної множини даних для спрощення обчислень. При підстановці оцінок пропущених значень замість пропусків слід приділити увагу таким питанням, як вибір значень для підстановки і модифікації методів з метою врахування цих підстановок.

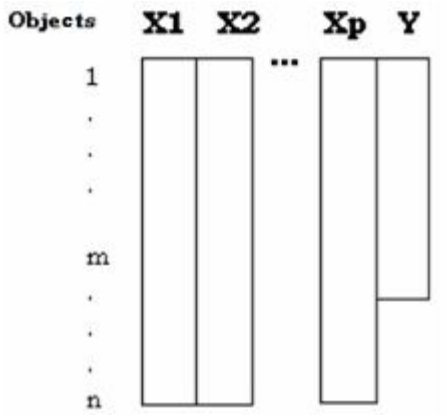

Рисунок 1.3 – Монотонна структура з пропусками в одній змінній

Розглянемо ситуацію, коли X – деякі чинники, що задаються дослідником, а Y – залежна від цих чинників змінна. Оскільки в експерименті значення чинників задаються статистиком, то пропуски, якщо вони є, містяться у вихідній змінній Y набагато частіше, ніж в значеннях чинників X. Тому обмежуються ситуацією, коли пропуски тільки в Y.

Метод, запропонований Бартлеттом для вирішення даної проблеми (1937), полягає в підстановці початкових значень замість пропусків і проведенні коваріаційного аналізу з супутньою змінною пропусків для кожного пропущеного значення.

Допустимо, що кожен пропуск y_i заповнюється початковим значенням, щоб вектор значень Y був повний. Позначимо початкові значення$y_i, i = 1, m_0$. Нехай $Z - n \times x$ матриця m_0 супутніх змінних пропусків. За визначенням i-та супутня змінна пропусків – це індикатор i-го пропущеного значення, тобто завжди 0, за винятком випадку, коли пропущено i-те значення, тоді вона рівна 1. Перший рядок Z, z_1 рівний$(1,0,\dots,0)$, ..., рядок m_0т0 рівний$(0,\dots,0,1)$, а всі z_i, при $i > m_0$ рівні $(0,\dots,0)$, оскільки вони відповідають присутнім y_i При коваріаційному аналізі використовується і X, і Z для прогнозу Y.

Припустимо, що для вихідної змінної $Y = (y_1, \dots, y_n)^T$ вірна лінійна модель:

$$Y = X\beta + Z\gamma + e \qquad (1.3)$$

де γ – вектор-стовпець з m_0 коефіцієнтів регресії для супутніх змінних пропусків,$e = e(e_1, \dots, e_n)^T$, e_i – незалежно і однаково розподілені з нульовим середнім і однаковою дисперсією σ^2, β – оцінюваний параметр – вектор довжини p.

Класична оцінка найменших квадратів β рівна

$$\beta = (X^T X)^{-1} X^T Y \tag{1.4}$$

якщо $(X^T X)$ має повний ранг. Якщо $(X^T X)$ невироджена, то β – незміщена оцінка β з мінімальною дисперсією. Якщо e_i розподілені нормально, то β – оцінка максимальної правдоподібності, розподілена нормально з середнім β β і дисперсією $\sigma^2 (X^T X)^{-1}$.

Розглянемо метод для нашого завдання. Залишкова сума квадратів, що мінімізується за (β, γ), рівна:

$$SS(\beta, \gamma) = \sum_{i=1}^{m_0} (y_i - x_i \beta - z_i \gamma)^2 + \sum_{i=m_0+1}^{n} (y_i - x_i \beta - z_i \gamma)^2 \tag{1.5}$$

Спростимо, використовуючи визначення матриці Z:

$$SS(\beta, \gamma) = \sum_{i=1}^{m_0} (y_i - x_i \beta - y_i)^2 + \sum_{i=m_0+1}^{n} (y_i - x_i \beta)^2 \tag{1.6}$$

Назвемо β_* – правильна оцінка найменших квадратів β, яка одержана за формулою (1.3) на присутніх значеннях, тобто на останніх $m = n - m_0$ рядками (Y, X). Вона мінімізує другу суму у виразі (1.4). Але якщо при $\beta = \beta^T$ покласти $y = (\gamma_1, \dots, \gamma_{m_0})^T$, де

$$\gamma_i \tilde{} y_i - x_i\beta, i = 1, m_0 \quad (1.7)$$

m_0 – кількість відсутніх значень, якими для простоти вважають перші m_0 спостережень, то буде мінімізована і перетвориться на нуль перша сума в (1.4), так що:

$$SS(\beta_*, \gamma) = \sum_{i=m_0+1}^{n} (y_i - x_i\beta_*)^2. \quad (1.8)$$

Означає, (β_*, γ) мінімізує $SS(\beta, \gamma)$ і є оцінкою найменших квадратів (β, γ), що отримують з моделі (1.2). Рівняння (1.4) означає також, що точна оцінка найменших квадратів відсутнього значення y_i, тобто $y_i = x_i\beta_*$, є $y_i - \gamma_i$ або в словесному формулюванні: прогноз i-го пропущеного значення методом найменших квадратів є початкове значення для i-го пропуску мінус коефіцієнт для супутньої змінної i-го пропуску.

У роботі Бартлетта всі y_i прирівнюються за цим методом до нуля, але з обчислювальної точки зору використання в якості y_i загального середнього привабливіше і дає точну суму квадратів відхилень від середнього.

Метод має наступні переваги.

- Він неітеративний, і, отже, знімає питання про збіжність.
- Якщо структура пропусків має виродженість (наприклад, у тому випадку, коли не можна оцінити деякі параметри, як за відсутності всіх значень для якоїсь обробки), цей метод «попереджає» дослідника, тоді як ітеративні методи приводять до відповіді, можливо, неприпустимої.
- Метод дає правильні оцінки і залишкові суми квадратів, а також вірні стандартні помилки, суми квадратів і F-критерії.

Хоча цей метод привабливий в певних відношеннях, його часто не можна реалізувати безпосередньо, тому що спеціалізовані програми дисперсійного аналізу можуть не мати можливості вести обробку при багатьох супутніх змінних.

У пакеті Statistica для заповнення пропусків в даних передбачена можливість заміни за середнім значенням. Це можна зробити в спеціалізованому модулі по роботі з даними «Data Management» за допомогою команди «Replace Missing Data by Means» – підставляються середні присутніх значень. Тому метод середнього включений в дослідження як метод, найбільш часто використовуваний в статистичних пакетах.

Метод безумовного середнього – найпростіший вид заповнення. Він полягає в оцінці відсутніх значень y_{ij} середнім $y_j^{(j)}$ за присутніми значеннями змінної Y_j. Середнє спостережуваних і підставлених значень рівне $y_j^{(j)}$ – оцінці методом доступних спостережень. Для рівноймовірного плану середнє популяції $\bar{Y}$ можна оцінити середнім присутніх і підставлених значень $\overline{y_j^{(j)}}$, а саме $\sum_{j=1}^{N} n_j \bar{y}_j^{(j)} / \sum_{j=1}^{N} n_j$. Дисперсія спостережуваних і підставлених значень рівна $[(n^{(j)} - 1)/(n-1)]S_{jj}^{(j)}$, де $S_{jj}^{(j)}$ – оцінка дисперсії методом доступних спостережень. За умови випадкової відсутності даних $S_{jj}^{(j)}$ – спроможна оцінка дійсної дисперсії, так що вибіркова дисперсія для даних після заповнення – занижена $(n^{(j)} - 1)/(n-1)$ раз оцінка дисперсії. Це заниження – природній наслідок заповнення пропусків середнім (значеннями в центрі розподілу).

У 1977 році американським статистиком Бредлі Ефроном був запропонований метод «bootstrap». Спочатку цей метод виник як засіб подолання зсуву, обумовленого вибіркою, потім, почав широко застосовуватися для роботи з будь-якими статистичними завданнями: перевірка гіпотези про закони розподілу випадкових величин, регресія, дисперсійний аналіз або багатовимірна класифікація. Основною перевагою бутстреп підходу є те, що він не потребує апріорного знання закону розподілу початкових даних, а значить підходить для роботи з будь-якими даними. Відмінність бутстрепа від традиційних методів полягає в тому, що він припускає багатократну обробку різних частин одних і тих же даних, як би поворот їх різними гранями, і зіставлення отриманих таким чином результатів.

Різновидом бутсреп методу є порівняно новий метод обробки статистичних даних, званий resampling. У даній роботі resampling метод застосовується для вирішення завдання заповнення пропусків в неповних даних, коли значення для заповнення пропущених елементів вибираються випадковим чином з початкової множини даних X_i. Значення для заміни пропуску можна вибрати двома способами: з поверненням і без повернення. Використовуватимемо спосіб з поверненнями, коли раніше вибране значення може брати участь в заміні ще раз.

Розглянемо застосування resampling методу за ситуації, коли дані представлені множиною значень НОР СВ X і залежним від них відгуком Y, причому деякі значення відгуку відсутні. Припустимо, що є деяка випадкова вибірка, що складається з незалежних безперервних СВ$X = \{X_1, \dots, X_m\}$ і значення відгуку Y. Причому вибірка така, що деякі значення відгуку пропущені. Розташуємо дані в монотонну структуру. Тоді присутні значення будуть $Y_y, i = \overline{1,k}$, пропуски опиняться в $Y_y, i = \overline{k+1,n}$. Для кожної з незалежних величин $X_1, \dots, X_m$ і відгуку Y можна одержати деяку вибіркову множину повних спостережень $H_i = \{x_{i1}, \dots, x_{in}, Y_i\}$. Припустимо, що розподіл СВ $\{X_i\}$ і Y невідомий. Застосування resampling-методу для завдання заміни пропусків в даному випадку може бути здійснено двома способами.

Resampling метод має 2 способи вирішення.

При першому способі будується матриця повних спостережень $H_{(m+1)\times k} = \{X_1, \dots, X_m\}$, де k – кількість присутніх спостережень.

Для кожного пропуску вибирають з вибірки H випадковим чином спостереження, яким заміщають пропущене значення Y і відповідні йому $X_1, \dots, X_m$. Для цього генерується випадкове число $j = Rnd()$, проводиться заміна спостереження з пропуском $\{X_{i1}, \dots, X_{in}, Y_i\}, i = \overline{(k+1),n}$ на $H_i = \{X_{i1}, \dots, X_{in}, Y_i\}$;

За даними, одержаним при заповненні resampling 1 методом, будується регресійна модель, і знаходяться оцінки β_i коефіцієнтів $i = \overline{1,m}$ і вільного члена β_0.

У другому способі resampling методу за присутніми спостереженнями будується регресійна модель, і знаходяться оцінки β_i коефіцієнтів, $i = \overline{1,k}$.

1. Знаходиться оцінка Y_i за регресійній моделі для $i = \overline{1,k}$.

2. Знаходиться помилка $\varepsilon_i = Y_i - Y_i$, $i = \overline{1,k}$..

3. Для кожного пропуску, підставляючи значення супутніх змінних $X_1, \dots, X_m$ в одержане регресійне рівняння, знаходимо оцінку Y_i, $i = \overline{k+1,n}$.

4. Значення, яким заміщають пропуск, отримують за формулою:$Y_i = Y_i + \varepsilon_j$, $i = \overline{k+1,n}$, де ε_j вибирається випадково з раніше розрахованих помилок (генерується випадкове число $j = Rnd()$.

5. За даними, одержаними після заповнення, будується регресійна модель, і знаходяться оцінки β_i коефіцієнтів $i = \overline{1,m}$ і вільного члена β_0.

Дані алгоритми повторюють r раз і після цього знаходяться середні значення коефіцієнтів регресійної моделі, які розраховуються як:

$$\beta_0 = \frac{\sum_{i=1}^{N} \beta_0}{r},\ \overline{\beta_0} = \frac{\sum_{i=1}^{N} \beta_i}{r},i = \overline{1,m} \tag{1.9}$$

Знайдені значення $\overline{\beta_0}$,$\overline{\beta_\iota}$, $i = \overline{1,m}$ є результатом застосування resampling-методів.

Позитивним чинником на користь resampling-методу є повторне використання початкових даних, адже збільшення кількості підвибірок дозволяє якнайповніші і інформативно використовувати початкову інформацію. З іншого боку, кількість нової інформації зменшується для кожної нової підвибірки, оскільки збільшується ймовірність того, що дані елементи вибірки були вже вибрані раніше – це основний недолік методу.

Як показано у параграфі не існує єдиного підходу до аналізу пропущених даних. Усі методи, що розглянуті, потребують залучення експерта, який би приймав рішення щодо відсутніх даних та методу їх заповнення, тобто вставки значення, яке б досить адекватно відповідало пропущеному значенню. Тому постає проблема дослідити чи можливо автоматизувати ці методики, у випадку

допустимості такого – реалізувати автоматизовану методику у вигляді комп’ютерних програм.

За базовий алгоритм заповнення, на основі проаналізованих методик, доцільно взяти лінійний ітераційний алгоритм відновлення пропущених значень.

1.3 Постановка задачі дослідження

Основною задачею буде розробка алгоритму отримання результатів по запитаних даних в умовах невизначеності. Також задачею буде аналіз зовнішніх даних на основі дій користувача, та відгук про отриманий результат, та оптимального заповнення сховища даних

Для зменшення затрачуваного часу на заповнення сховища даних, необхідно розробити систему яка буде ефективно обробляти вхідні дані та генерувати на їх основі критерії для подальшого використання в критерії пошуку. Для цього система повинна активно вміти розрізняти вхідний набір символів і групувати їх відносно певної категорії. При цьому враховуючи те, що кожна група може мати батьківську групу, що дозволить краще проводити подальший аналіз вхідних параметрів та будувати дерево рішень для процесу пошуку. Прикладом таких процесів є контекстні реклами від Google.

Неповні дані впливають на точність класифікації. Наступні техніки є ефективні для роботи з неповними даними.

1. Модель ISOM-DH працює з неповними даними використовуючи незалежний аналіз компонентів і самоорганізовані карти. Вони використовують існуючі дані для того щоб передбачити дані яких не достає, та візуалізує отримані високорозмірні дані.
2. Інша техніка базується на еволюції стратегій використовуючи параметричний та не параметричний методи поставлення. Загальні алгоритми і багато-шарові персептрони мають бути використані щоб розробити фреймворк для створення поставлений стратегій, які звертаються до декількох неповних атрибутів.

3. Мережевий підходи які базовані на багатозадачному вивченню: вивченню проблеми/сутності в зв’язку з іншими для патерну класифікації, з пропущеними вхідними даними, може бути прирівняний з процедурами представлення та використаний для обробки неповних даних на двох існуючих структурах даних.

1.4 Специфікація вимог до програмного продукту

Зробивши аналіз предметної області, приступаємо до опису варіантів використання. Кожен варіант використання – це окрема функція програмного продукту.

Проаналізувавши всі вимоги до системи, виділено наступні варіанти використання. Вони зображені в вигляді діаграми варіантів використання (рис. 1.1-1.2). З допомогою діаграми варіантів використання, можна з легкістю побачити основні функції системи.

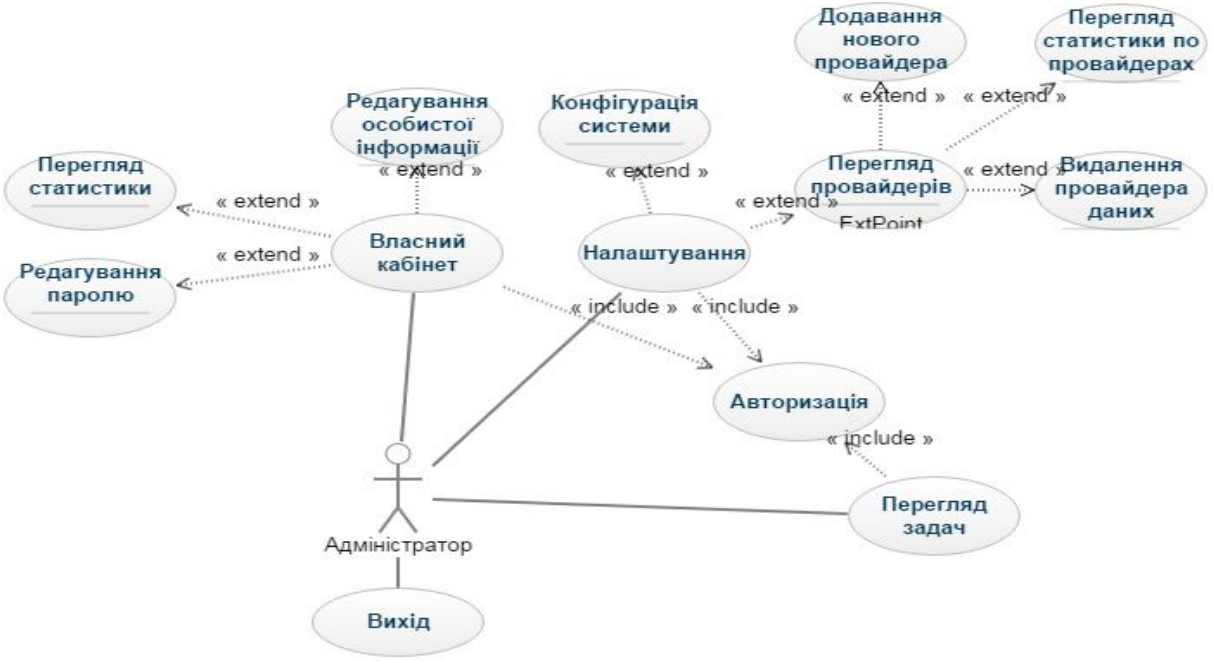

Рис.1.1. Діаграма варіантів використання Адміністратора

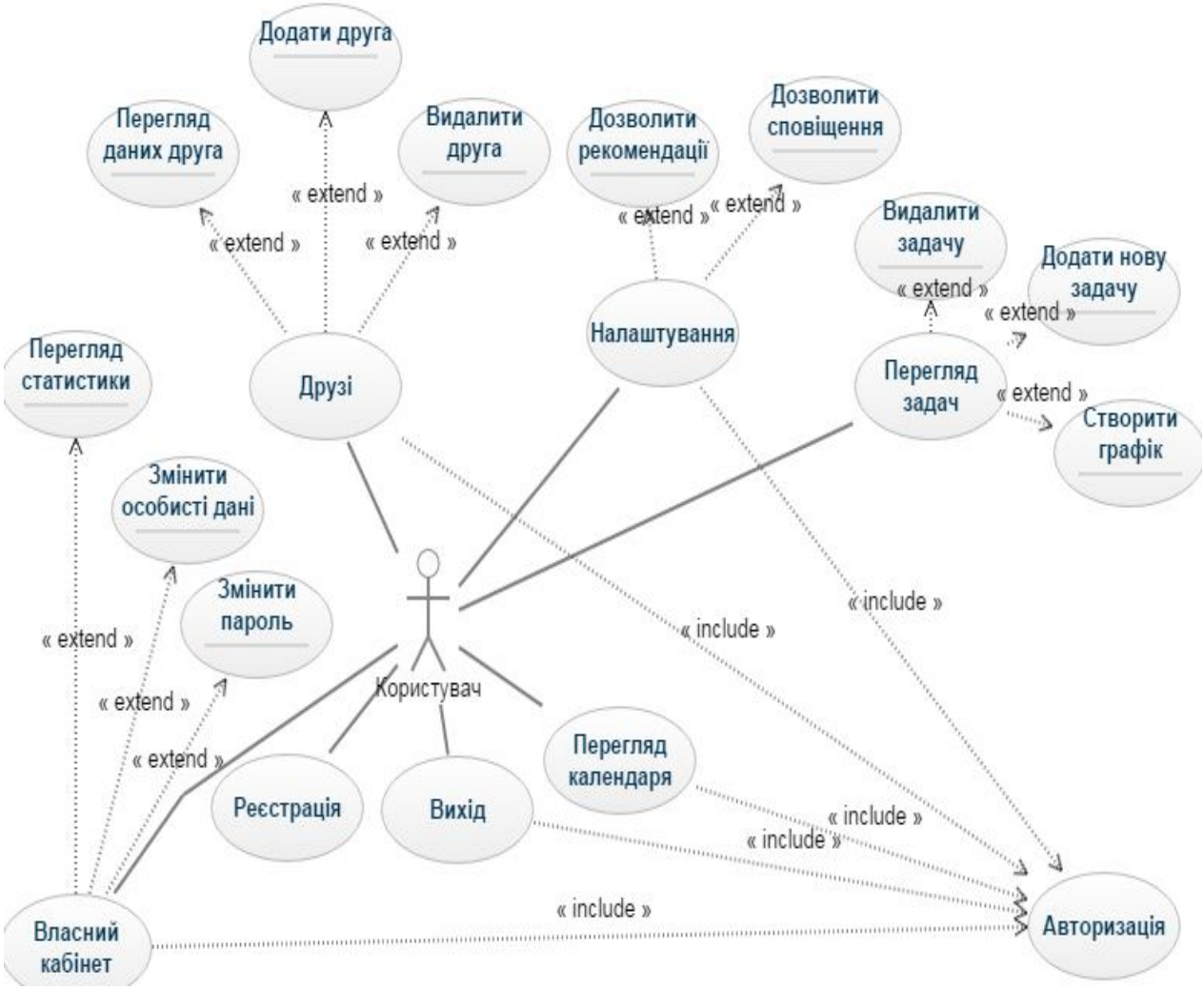

Рис.1.2. Діаграма варіантів використання Користувача

Наступним етапом у специфікації вимог до програмного продукту є розробка діаграми послідовності. Діаграма послідовності показує взаємодію об’єкту в часових послідовностях. Вона зображує об’єкти та класи які використовуються в сценарії, та послідовність повідомлень якими обмінюються об’єкти, які необхідні для виконання функціональної частини сценарію. Діаграми послідовності асоціюються з реалізаціями діаграми варіантів використання в логічному вигляді системи під час розробки. Діаграма послідовності процесу додавання нового провайдера зображена на рисунку 1.3.

Рис.1.3. Діаграма послідовності

Проаналізувавши варіанти використання, було сформовано специфікацію функціональних вимог, яка наведена у таблиці 1.1.

Таблиця 1.1

Специфікація функціональних вимог

Ідент. вимоги	Назва вимоги (варіанту використання)	Атрибути вимоги	
		Пріоритет	Складність
1.	Реєстрація	Обов’язкова	Середня
2.	Вхід	Обов’язкова	Середня
3.	Власний кабінет	Обов’язкова	Висока

Продовження таблиці 1.1

4.	Зміна паролю	Обов’язкова	Середня
5.	Вихід	Обов’язкова	Середня
6.	Зміна особистих даних	Необов’язкова	Низька
7.	Налаштування	Обов’язкова	Висока
8.	Дозволити рекомендації	Обов’язкова	Низька
9.	Дозволити сповіщення	Обов’язкова	Низька
10.	Друзі	Обов’язкова	Висока
11.	Перегляд даних друга	Обов’язкова	Висока
12	Додати друга	Обов’язкова	Висока
13	Видалити друга	Обов’язкова	Висока
14	Перегляд календаря	Обов’язкова	Середня
15	Перегляд задач	Обов’язкова	Висока
16	Видалити задачу	Обов’язкова	Середня
17	Додати задачу	Обов’язкова	Висока
18	Створити графік	Обов’язкова	Висока
19	Конфігурація системи	Обов’язкова	Висока
20	Перегляд стану надходження даних	Обов’язкова	Висока
21	Перегляд статистики	Обов’язкова	Висока
22	Провайдери	Обов’язкова	Висока
23	Додавання провайдера	Обов’язкова	Висока
24	Видалення провайдера	Обов’язкова	Середня
25	Перегляд статистики провайдерів	Обов’язкова	Висока
26	Конфігурація системи	Обов’язкова	Висока

Специфікація нефункціональних вимог наведена у таблиці 1.2.

Таблиця 1.2

Специфікація нефункціональних вимог

Ідент. вимоги	Назва вимоги	Характеристики
1.	Зручність інтерфейсу	Гнучкий інтерфейс який би легко відображався як на телефонах так і на планшетах.
2.	Інтуїтивне розміщення елементів дизайну	Елементи користувацького інтерфейсу повинні бути розміщені на типових для користувачів місцях.
3.	Швидкість відклику.	Не більше двох 2 секунд.
4.	Кольорова гамма	Згідно стандартів
5.	Гнучкість	Можливість додавання нових модулів без перезавантаження старих.
6.	Локалізація	Можливість зміни мови додатку.
7.	Стійкість до збоїв.	Система повинна коректно відновлюватися після неочікуваних збоїв.
8.	Безпека	Система повинна здійснювати взаємодію по токену та https протоколу, щоб забезпечити безпеку даних.
9.	Конфігурабельність	Система повинна бути легко конфігурована і більшість змін повинні вноситися без перезапуску системи.
10.	Тестованість	Система повинна бути покритою тестами.

Висновки до першого розділу

1. Здійснено аналіз об’єкту управління.
2. Проаналізовано поняття «інтелектуальний аналіз даних» та охарактеризовано основні проблеми, які виникають в процесі його реалізації.
3. Визначено та проведено аналіз основних проблем, які виникають в процесі пошуку даних в умовах невизначеності та заповнення сховища даних.
4. Описано та здійснено аналіз існуючих алгоритмів для пошуку даних в умовах невизначеності.

РОЗДІЛ 2
МАТЕМАТИЧНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗАДАЧІ TIME MANAGEMENT

2.1. Механізм отримання даних із вхідного потоку інформації

Інформація буває в багатьох видах. Одна з важливих форм інформації є структуровані дані, де вся інформація побудована логічно і зв’язно. Наприклад ми можемо бути заінтересовані в зв’язкаї між компаніями та їх діяльністю. Беручи до уваги певну компанію, ми хотіли б знайти які ще компанії працюють в тому чи іншому місці, якщо дані структуровані у вигляді нариклад таблиці, то отримання такої інформації легкий процес. Проте набагато тяжче знайти ті самі дані з тексту. Для того щоб отримати результат потрібно спочатку неструктуровані дані конвертувати в структуровані.

На рисунку 2.1 показано архітектуру для простої системи видобутку інформації. Вона починається з обробки документа використовуючи декілька процедур. Спочатку документ розбивається на речення використовуючи сегментатор речень, і потім кожне речення далі розбивається на слова використовуючи токенатор. Далі кожне з речень маркується так званим «тегом», які далі за допомогою механізму розпізнавання іменної сутності будуть знайдені. Останнім кроком є розпізнавання залежностей для знаходження можливих віднршень.

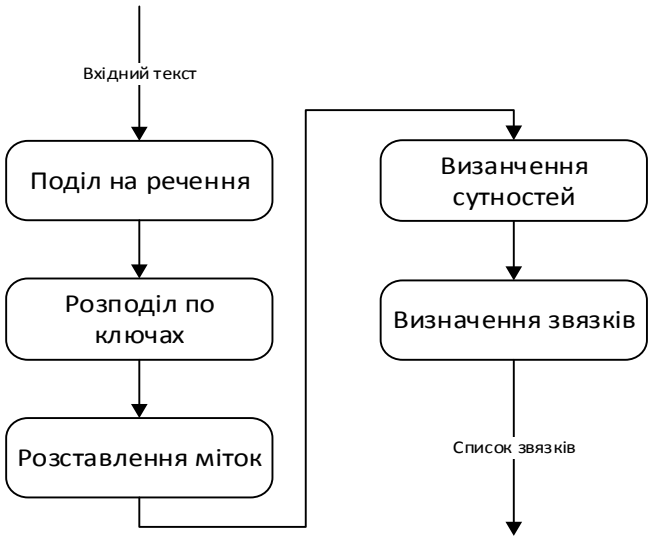

Рис. 2.1. Зображення механізму видобутку даних

Для того щоб знайти частину структури для заданого речення може бути використані регулярні вирази. RegexpParser починається з простої структури в якій немає токенів.Потім за допомогою правил обробки ця структура наповнюється токенами, як тільки всі правила були виконані результуюча структура з токенами повертається. На рис 2.2 показано просту процедуру розбиття з двох правил.

```
grammar = r"""
  NP: {<DT|PP\$>?<JJ>*<NN>}   # chunk determiner/possessive, adjectives
and noun
      {<NNP>+}                # chunk sequences of proper nouns
"""
cp = nltk.RegexpParser(grammar)
sentence = [("Rapunzel", "NNP"), ("let", "VBD"), ("down", "RP"),
                 ("her", "PP$"), ("long", "JJ"), ("golden", "JJ"),
("hair", "NN")]
```

```
>>> print(cp.parse(sentence))
(S
  (NP Rapunzel/NNP)
  let/VBD
  down/RP
  (NP her/PP$ long/JJ golden/JJ hair/NN))
```

Рис. 2.2 процедура розбиття з двох правил.

Перше правило співпадає зопціональним визначником або з присвійним займенником, нулем або більше прикметників, і потім з іменником. Друге правило співпадає з одним або більше необхідних іменників.

Іноді легше виділити те що ми виключаємо з структури. Ми можемо визначити так званий chink, що буде послідовністю токенів які не включиються в структуру. На рис 2.3 показаний приклад chink`a.

```
    [ the/DT little/JJ yellow/JJ dog/NN ] barked/VBD
at/IN [ the/DT cat/NN ]
```

Рис. 2.3. Приклад chink`a

Chinking – це процес видалення послідовності токенів з структури.

- Якщо співпадаюча послідованість токенів займає всю структуру, тоді вся структура видаляється.
- Якщо послідовність токенів знаходиться посередині структури, ці токени видаляються, залишаючи дві структури.
- Якщо послідовність є на периферійності структури, ці токени видаляються і менші структури залишаються.

Ці 3 варіанти показані у таблиці 2.1

Таблиця 2.1.

Три правила chinking`a використані для однакового вводу.

``	**Весь блок**	**Середина блоку**	**Кінець блоку**
Вхідні дані	[a/DT little/JJ dog/NN]	[a/DT little/JJ dog/NN]	[a/DT little/JJ dog/NN]
Операція	Chink "DT JJ NN"	Chink "JJ"	Chink "NN"
Патерн	}DT JJ NN{	}JJ{	}NN{

Як переваги проміжного статусу між тегуванням і парсингом, структурі дані можуть бути представлені у вигляді тегів або дерев. Найбільш поширена репрезецнтація файлів використовує IOB теги. У цій схемі, кожен токен помічений одним із спеціальних тегів I (всередині), O (ззовні), або B (початок). Токен позначається як B якщо він означає початок структури. Всі послідуючі токени позначаються як I. Та всі інші як o. Потім до B та I тегів додається суфікс типу, B-NP, I-NP. Звісно не є необхідністю вказування типу структури для токенів які знаходяться поза межами структури, тому вони просто позначаються як o. IOB теги стали стандартним способом представлення кусків

структур у файлах, і ми також використаємо їх в цьому форматів. Ось приклад інформації який би з’явився у файлі рис. 2.4.

```
We PRP B-NP
saw VBD O
the DT B-NP
yellow JJ I-NP
dog NN I-NP
```

Рис.2.4. Приклад інформації з IOB тегами.

У цьому представленні показаний один токен на лінію, кожен з своєю частиною мовного тегу і тегу куска даних. Цей формат дозволяє нам представляти більше ніж один тип куска даних, до поки вони не перекриваються один одного. Як було видно раніше, струкури кусків даних також можуть бути представлені використовуючи дерева. Вони будуть мати перевагу що кожен блок даних установчий, і може бути маніпульований напряму. Приклад показано на рис. 2.5.

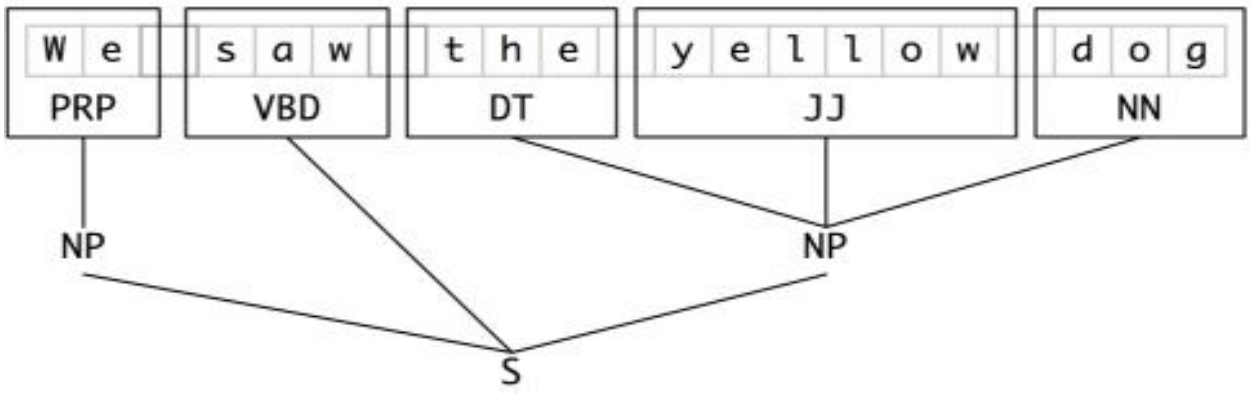

Рис.2.5 – приклад деревовидної репрезентації структур кусків даних.

2.2. Червоно-чорні дерева.

Червоно-чорні дерева це тип самозбалансованого дерева пошуку. Кожен вузол бінарного дерева має один екстра біт, який часто інтерпретується як колір, червоний або чорний. Ці колірні біти використовуються щоб запевнити те що дерева залишаються відносно збалансованими під час вставок та видалень.

Баланс зберігається за рахунок окрашування кожного вузла дерева в один з довх кольорів(типово червоний або чорний), щопо своєму задовільняються певні властивості, які в загальному обмежують те, наскільки не збалансованим може стати дерево в найгіршому випадку. Коли дерево модифікується, нове дерево переставляється і перемальовауються щоб зберегти властивості кольору. Ці властивості розроблені так, щоб перестановка і перемалювання відбувалися ефективно.

Баланс не є ідеальний, але він досить непоганий щоб забезпечити пошук в час O(log n), де nкількість елементів в дереві. Вставка і видалення також відбуваються за O(log n). Відслідковування кольору кожного вузла дерева потребує лиш 1 біт інформації на вузол, оскільк є всього 2 кольори. Дерево не вміщає інших спецю даних, тому її слід у пам’яті майже співпадає з класичним бінарним деревом. У більшості випадків додатковий біт може бути збережений без додаткових затрат пам’яті.

На додаток до вимог, що пред'являються до бінарного дерева пошуку наступне має бути виконано для червоно-чорного дерева:

1. Кожен вузол є червоний або чорний.

2. Корінь чорний. Це правило іноді опускається. Так як корінь завжди може бути змінений від червоного до чорного, але не обов'язково навпаки, це правило мало впливає на аналіз.
3. Все листя (NIL) чорного кольору.
4. Якщо вузол червоний, то обидва його нащадка чорні.
5. Кожен шлях від даного вузла до будь-якого з його нащадків NIL вузлів містить однакове число чорних вузлів. Оскільки такі операції, як вставка, видалення і пошук значень вимагають найгіршого часу, пропорційне висоті дерева, ця теоретична верхня межа на висоті дозволяє червоно-чорним деревам бути ефективними в гіршому випадку, на відміну від звичайних бінарних дерев пошуку.

Щоб зрозуміти, чому це гарантується, досить розглянути вплив властивостей 4 і 5 разом. Для червоно-чорного дерева T, нехай B число чорних вузлів у властивості 5. Нехай найкоротший шлях від кореня T до будь-якого листа складаються з B чорних вузлів. Довші можливі шляхи можуть бути побудовані шляхом вставки червоних вузла. Проте, властивість 4 унеможливлює вставити більш одного послідовного червоного вузла. Тому, ігноруючи будь-які чорні листя NIL, найдовший можливий шлях складається з 2 * вузлів B, чергуючи чорний і червоний (це найгірший випадок). Підрахунок чорні листя NIL, найдовший можливий шлях складається з 2 * B-1 вузлів.

Найкоротший можливий шлях має всі чорні вузли, а найдовший можливий шлях чергується між червоними і чорними вузлами. Так як всі максимальні доріжки мають однакове число чорних вузлів, по властивості 5, це показує, що жоден шлях не більше, ніж в два рази до тих пір, як будь-який інший шлях.
Червоно чорне дерево, яке містить n внутрішніх вузлів має висоту O (log N).

h(v) = висота піддерева з коренем у вузлі V
bh (v) = кількість чорних вузлів від V до будь-якого листа в поддереве, не рахуючи V, якщо вона чорна - називається чорно-висота

Лемма: поддерево з коренем у вузлі V має принаймні $2^{bh(v)} - 1$ внутрішніх вузлів.

Доказ леми (через індукцію висоти):

Основа: h (v) = 0

Якщо V має висоту нуля, то воно повинно бути нульовим, тому bh (v) = 0. Таким чином:

$$2^{bh(v)} - 1 = 2^0 - 1 = 1 - 1 = 0 \qquad (2.1)$$

Індукртивний крок: v так як h(v) = k, має хоча б $2^{bh(v)} - 1$ внутрішніх вулзів говорить про те що v' так як $h(v')$ = k+1 має хоча б $2^{bh(v')} - 1$ внутрішніх вузлів.

Оскільки v' має h(v')>0 вона є внутрішнім вузлом. Виходячи з цього вона має двох дітей, кожен з яких має чорну-висоту з bh(v') або bh(v') -1 (зважаючи на те чи дитина є чорною або червоною). Інудктивною гіпотезою кожна дитина має хоча б $2^{bh(v')-1} - 1$ внутрішніх везлів, тому v' має хоча б

$$2^{bh(v')-1} - 1 + 2^{bh(v')-1} - 1 + 1 = 2^{bh(v')-1} - 1 \qquad (2.2)$$

внутрішніх вузлів.

Використовуючи цю лемму ми можемо покзати що висота дерева є логарифмічною. Оскільки хоча б половина вузлів на любому з шляїів з основи до листка є чорними(властивість 4), чорна-висота кореня є хоча б h(root)/2. Тому за допомогою лемми ми отримуємо:

$$n \geq 2^{\frac{h(root)}{2}} - 1 \leftrightarrow log_2(n+1) \geq \frac{h(root)}{2} \leftrightarrow h(root) \leq 2log_2(n+1) \qquad (2.3)$$

Тому висота кореня є O(log n).

На додачу до вставки одного елемента, видалення і операції пошуку, декілька операцій були визначені на червоно чорних деревах: об’єднання, перетин і різниця сетів. Вони залежать від двох допоміжних операцій, Розбивання та Об’єднання.

Об’єднання : Функція об’єднання на двох деревах t1 та t2 і ключом k поверне дерево що містить елементи t1 t2 та k. Вона потребує щоб k був більший ніж всі ключі в t1 і менший ніж всі ключі в t2. Якщо два дерева мають таку саму висоту, об’єднання створить новий вузол з лівим піддеревом t1, корінь k, та правим піддеревом t2. Якщо обидва t1 та t2 мають чорний корінь k стає червоним. Інакше k стає чорним.

Розбивання : Для того щоб розділити дерево у менші дерева, менші ніж ключ x і більші ніж ключ x, спочатку малюється шлях до від кореня через вставку x в дерево. Після вставки всі зачення більші за x будуть справа. Застосовуючи Об’єднання, всі піддерева на лівій стороні об’єднуються знизу вверх використовуючи ключ на шляху як проміжні вузли, а права частина є асиметричною. Час затрачений на розбирвання є O(log n).

Складність кожного об'єднання перехвату і різниці є :

$$O\left(m\ log\left(\frac{n}{m}+1\right)\right) \tag{2.4}$$

Для двох дерев з розмірами m та $n(\geq m)$. Ця складність є оптимальною по клікості порівнянь. Більш важливим є те, що рекурсивні виклики до об’єднання, перехвати та різниці є незалженими один від одного, тому можуть бути виконані в паралелі з $O(\log m \log n)$ коли m=1.

Паралельні алгоритми для створення червоно-чорних дерев з відсортованих списків може бути виконана за константний час O(log log n), залежачи більше від моделі комп’ютера, і чи кількість процесорів є асимптотично пропораціональна до клікості елементів де $n \to \infty$.

2.3 Розробка та оцінка методу розбиття на блоки

Почнемо з того щоб розібратися як конвертувати IOB формат, і потім як це виконується на більшому масштабі.

Використовуючи модуль corpus ми можемо завантажити текст який був промарканий як і потім погрупований в блоки використовуючи нотацію IOB. Блок категорія який надає corpus є NP, VP та PP. Так як ми бачили, кожне речення показане використовуючи декілька ліній, рис 2.6.

```
he PRP B-NP
accepted VBD B-VP
the DT B-NP
position NN I-NP
...
```

Рис 2.6. Приклад блоку з IOB вказівниками.

Функція конвертації будує деревовидне представлення однієї з цих стрічок. Більше того вона дає можливість вибрати любий підсет з трьох типів блоків на використання. Дерево показане на рис 2.7.

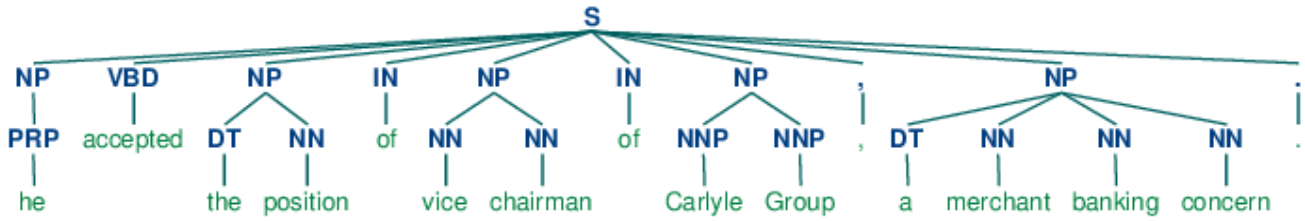

Рис.2.7 побудоване дерево

Враховуючи те що ми маємо доступ до блоків, ми можемо оцінити їх. Рис 2.8.

```
>>> from nltk.corpus import conll2000
>>> cp = nltk.RegexpParser("")
>>> test_sents = conll2000.chunked_sents('test.txt', chunk_types=['NP'])
>>> print(cp.evaluate(test_sents))
ChunkParse score:
    IOB Accuracy:  43.4%
    Precision:      0.0%
    Recall:         0.0%
    F-Measure:      0.0%
```

Рисунок 2.8. Результат прогону парсера блоків

Точність IOB тегу індикує що більше ніж третя частина слів промаркована з o, тобто не знаходяться в блоці. Проте оскільки маркувальник не знайшов ніяких блоків, його точність, вігук і f-вимір всі дорівнюють 0. Тепер добавимо регулярний вираз який шукає на теги що починаються з літер і характерзують іменник(CD, DT, JJ). Рис 2.9

```
>>> grammar = r"NP: {<[CDJNP].*>+}"
>>> cp = nltk.RegexpParser(grammar)
>>> print(cp.evaluate(test_sents))
ChunkParse score:
    IOB Accuracy:  87.7%
    Precision:     70.6%
    Recall:        67.8%
    F-Measure:     69.2%
```

Рис 2.9. застосування регулярного виразу.

Як можна бачити такий підхід приніс кращі результати. Проте ми можемо покращити його використовуючи підхід де ми використовуємо corpus щоб знати блок тегів(I, O, B) що є найбільш вірогідним для кожної частини блоку. Іншими словами, можна побудувати розбивач блоків використовуючи unigram маркувальник.

2.4 проектування алгоритму пошуку даних в умовах невизначеності.

Ухвалення рішень в умовах ризику може ґрунтуватися на одному з таких критеріїв:

- критерії очікуваного значення;
- комбінації очікуваного значення і дисперсії;
- критерії граничного рівня;
- критерії найбільш імовірної події в майбутньому.

Розглянемо детальніше застосування цих критеріїв.

Використання критерія очікуваного значення припускає ухвалення рішення, що обумовлює максимальний прибуток за наявних початкових даних про

ймовірність отриманого результату при тому або іншому рішенні. По суті, КОЗ – вибіркові середні значення випадкової величини.

Природно, що достовірність отриманого рішення при цьому залежатиме від обсягу вибірки. Так, якщо позначити

$$E = \sum_{i=1}^{n} P_i - X_i$$

(2.5)

де $P_i X_i$ – відповідно ймовірність і значення і-го результату; n – кількість можливих результатів.

Таким чином, КОЗ може застосовуватися, коли однотипні рішення в схожих ситуаціях доводиться приймати багато разів. Як вказувалося вище, КОЗ має сферу застосування, обмежену значним числом однотипних рішень, що приймаються в аналогічних ситуаціях. Цей недолік можна усунути, якщо застосовувати комбінацію КОЗ і дисперсії σ^2:

$$\sigma^2 = \sum_{i=1}^{n} (X_i - E)^2 - P_i, \qquad BP = \frac{\sigma}{E}$$

(2.6)

Критерій граничного рівня не має чітко вираженого математичного формулювання і базується значною мірою на інтуїції і досвіді ЛПР. При цьому ЛПР на підставі суб'єктивних міркувань визначає найбільш прийнятний спосіб дій. Критерій граничного рівня зазвичай не використовується, коли немає повного уявлення про безліч можливих альтернатив. Урахування ситуації ризиків при цьому може здійснюватися за рахунок введення законів розподілів випадкових чинників для відомих альтернатив.

Незважаючи на відсутність формалізації критерієм граничного рівня користуються досить часто, задаючи їх значення на основі експертних або експериментальних даних.

Критерій найбільш імовірного результату припускає заміну детермінованої випадкової ситуації шляхом заміни випадкової величини прибутку (або витрат)

єдиним значенням, що має найбільшу ймовірність реалізації. Використання цього критерію, також, як і у попередньому випадку, значною мірою спирається на досвід та інтуїцію. При цьому необхідно враховувати дві обставини, що роблять більш важким застосування цього критерію:

- критерій не можна використовувати, якщо найбільша ймовірність події неприпустимо мала;
- застосування критерію неможливе, якщо декілька значень імовірності можливого результату рівні між собою.

Вибір найкращого рішення в умовах невизначеності істотно залежить від того, яка ступінь цієї невизначеності, тобто від того, якою інформацією розташовує ЛПР.

Припущення суб'єктивні, тому й ступеня невизначеності з боку ЛПР повинні відрізнятися. Практикуються два основні підходи до прийняття рішення в умовах невизначеності. Особа, що приймає рішення, може використовувати наявну в нього інформацію і свої власні особисті судження, а також досвід для ідентифікації та визначення суб'єктивних ймовірностей можливих зовнішніх умов, оцінки можливих наслідків альтернатив в різних умовах зовнішнього середовища. Це, по суті, робить умови невизначеності аналогічними умовами ризику, а процедура прийняття рішення, що обговорювалася раніше для умов ризику, виконується і в цьому випадку.

Якщо ступінь невизначеності занадто висока, то ЛПР воліє не робити припущень щодо ймовірностей різних зовнішніх умов, тобто ця особа може або не враховувати ймовірності, або розглядати їх як рівні, що практично одне і те ж. Якщо застосовується даний підхід, то для оцінки передбачуваних стратегій є чотири критерію рішення:

1) критерій рішення Вальда, званий також Максиміна;

2) альфа-критерій рішення Гурвіца;

3) критерій рішень Севіджа, званий також критерієм відмови від минимакса;

4) критерій рішень Лапласа, званий також критерієм рішення Бейеса.

Мабуть, найважче завдання для ОПР полягає у виборі конкретного критерію, найбільш підходящого для вирішення запропонованого завдання. Вибір критерію повинен бути логічним за даних обставин. Крім того, при виборі критерію повинні враховуватися філософія, темперамент і погляди нинішнього керівництва фірми (оптимістичні або песимістичні, консервативні або прогресивні).

Розглянемо ці твердження на конкретному прикладі. Елементами моделі вибору альтернатив в умовах невизначеності є матриця прийняття рішень | ***A I,*** Sj | і цільова функція ***E {A*** i, w ***(S*** j)}.

A I, - альтернативи дій; Sj - стан зовнішнього середовища; ***W (S*** j) - ймовірності настання стану ***S J,*** причому Σmj = 1 ***W (S*** j) = 1; ***E*** ij - результат, який буде досягнутий, якщо обрана альтернатива ***A*** i і настане стан зовнішнього середовища ***S*** j

В якості прикладу візьмемо матрицю рішень (рис. 6.10), що включає в себе п'ять альтернатив ***(A*** i; ***I*** = 1, ..., 5) і чотири стану зовнішнього середовища ***(S*** j; j = 1,4). Наслідки прийнятих рішень наведені на перетині рядків і стовпців ***(E IJ).***

В умовах визначеності, тобто коли прийняття рішень відбувається після настання подій у зовнішньому середовищі (апостеріорі), має прийматися рішення що максимізує цільову функцію. Так, при настанні події ***S*** 1 необхідно приймати альтернативу A2, при S2 → A4, при S3 → A5, при S4 → A1.

В умовах ризику необхідно приймати рішення (вибирати альтернативу Ai) до настання події Sj в зовнішньому середовищі (апріорі), що вимагає врахування ймовірності ***W*** (Sj) настання цієї події. Це можна зробити шляхом множення

ймовірності настання цієї події ***W (S*** j) на результат ***E IJ,*** одержуваний від прийняття того чи іншого рішення, і вибрати найбільше значення Ai.

У разі якщо ступінь невизначеності занадто висока, то ЛПР може привласнювати значенням ймовірності свої суб'єктивні значення, зводячи задачу до прийняття рішень в умовах ризику, або не робити припущень щодо ймовірностей різних зовнішніх умов, тобто може або не враховувати ймовірності, або розглядати їх як рівні, застосовуючи різні критерії для вибору.

Критерієм Вальда "розраховуй на гірше" (критерій крайнього песимізму, або максимин) називають критерій, який наказував забезпечити значення параметра ефекту, рівного a:

$$a = \max_i \min_j a_{ij} \tag{2.7}$$

Цей критерій орієнтує ЛПР на найгірші умови і рекомендує вибрати ту стратегію, для якої виграш максимальний. В інших, більш сприятливих умовах використання цього критерію призводить до втрати ефективності системи або операції.

Іншим граничним випадком критерію Вальда є критерій "неприборканого оптимізму", або максимакс:

$$a = \max_i \max_j e_{ij}$$

(2.8)

Альфа критерій ріщення Гурвіца рекомендує при виборі рішення в умовах невизначеності не керуватися крайнім песимізмом (завжди "розраховуй на гірше", α = 0) або крайнім оптимізмом ("все буде найкращим чином", а = 1). Рекомендується якесь середнє рішення (0 ≤ α ≤ 1). Цей критерій має наступний вигляд:

$$H = \max_i \left[a \min_j e_{ij} + (1-a) \max_j e_{ij} \right]$$

(2.9)

де α - якийсь коефіцієнт, обираний експериментально з інтервалу між 0 і 1.

Використання цього коефіцієнта вносить додатковий суб'єктивізм в ухвалення рішень з використанням критерію Гурвіца.

Відповідно критерія рішення Севіджа, якщо потрібно в будь-яких умовах уникнути великого ризику, то оптимальним буде те рішення, для якого ризик, максимальний при різних варіантах умов, виявиться мінімальним.

При використанні критерію Севіджа забезпечується найменше значення максимальної величини ризику:

$$S = \min_i \max_j r_{ij}$$

(2.10)

де ризик ***R*** ij визначається виразом ***R*** ij = β - ***E IJ,*** β - максимально можливий виграш.

Критерій Севіджа, як і критерій Вальда, - це критерій крайнього песимізму, але тільки песимізм тут проявляється в тому, що мінімізується максимальна втрата у виграші в порівнянні з тим, чого можна було б досягти в даних умовах.

Критерій лапласа**,** або байес критерій, свідчить, що якщо ймовірності стану середовища невідомі, то вони повинні прийматися як рівні. У цьому випадку вибирається стратегія, що характеризується самої передбачуваною вартістю за умови рівних ймовірностей. Критерій Лапласа дозволяє зводити умова невизначеності до умов ризику. Критерій Лапласа називають критерієм раціональності, і він підходить для стратегічних довгострокових рішень, як і всі названі вище критерії.

Висновки до розділу 2

За результатами другого розділу зроблено наступні висновки:

1. Визначено алгоритм для оптимізації і автоматизації задачі заповнення сховища даних в умоваз невизначеності.
2. Проаналізовано роботу алгоритму на тестових даних, що дозволяє приступити до реалізації системи.

РОЗДІЛ 3
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Розробка архітектури програмної системи

Потрібно розробити систему для формування критерій та збереження даних на основі набору символів для системи тайм менеджменту. Система повинна зберігати дані про критерії(назва, дата створення, батьківська критерія, додаткова інформація), ключові слова критерії(критерія, ключове слово), елементи(назва, дата створення, дата модифікації, чи є валідний,

ідентифікатор), критерії елементів (елемент, критерія). Постачальники (назва, час створення, час оновлення, хто створив, хто оновив, чи використовується), елементи постачальників (елемент, постачальник), користувач(ім’я, прізвище, рік народження, логін, пароль, електронна адреса), пристрій(ім’я, ОС), версія ОС (ім’я, версія), пристрій користувача(користувач, пристрій). Категорії(ім’я, картинка, базова категорія). Критерії категорій(критерія, категорія)

Архітектура системи зображена на рисунку 3.1.

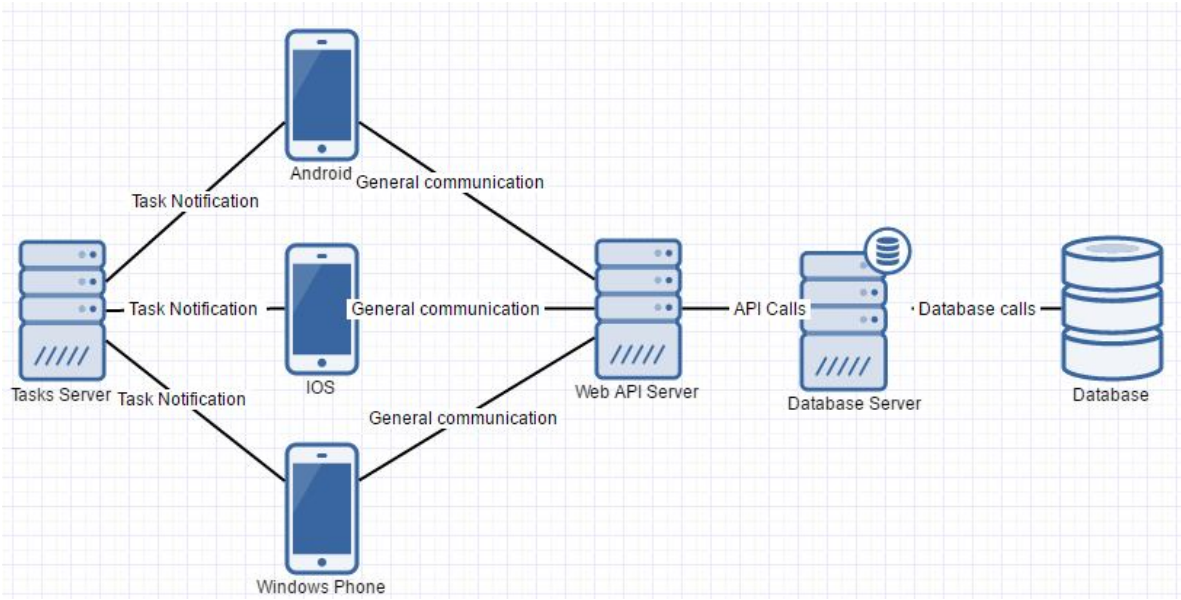

Рис.3.1. Архітектура системи

Враховуючи те що клієнтом виступає мобільний додаток, потрібно врахувати процес масштабованості серверної частини. Також потрібно приділити увагу кількості даних які будуть відправлятися з сервера, оскільки клієнтський інтернет не є ще дуже розвинутим. Рішенням масштабованості постає принцип розробки який має назву «тонкий клієнт». У цьому випадку уся бізнес логіка переноситься на серверну частину, коли клієнт тільки реалізовує мінімальну бізнес логіку свого додатку та логіку представлення даних. Це дозволяє максимально ефективно масштабувати проект без необхідності вносити суттєві зміни в код клієнта. В свою чергу деякі зміни можуть бути зроблені тільки на сервері, що дозволить клієнтам навіть не оновлюючись до

нової версії мати новіший функціонал та кращу швидкодію. При розробці було вирішено використовувати паттерн репозиторій та UnitOfWork.

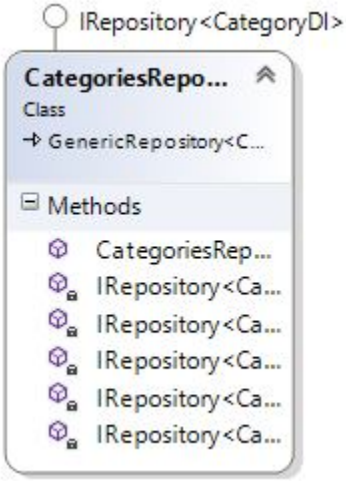

Рис.3.2 Клас CategoriesRepository

Клас CategoriesRepository (рис. 3.2) – цей клас представляє прошарок DLL, та реалізовує доступ до даних сутності категорія.

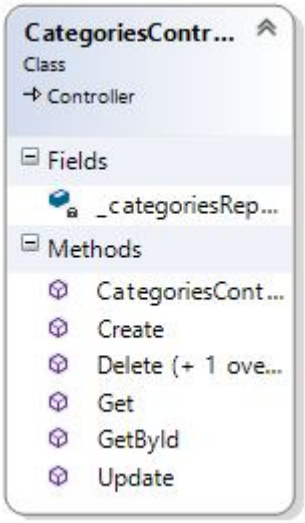

Рис. 3.3 Клас CategoriesController

Клас TaskController (рис. 3.3) цей клас відповідає за зв’язок клієнтів з сервером бд. Реалізовує методи взаємодії з DLL, тонко зв’язується з імплементацією репозиторія через DI що надає системі гнучкість в розширюванні.

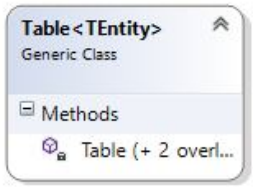

Рис. 3.4 Клас Table

Клас Table (рис. 3.4) відповідає сутності таблиці в системі. Цей клас розроблений з метою імплементації кастомної ОРМ системи. Включає в себе логіку роботи з таблицею в БД, а саме її створення та налаштування зв’язків між різними елементами таблиці.

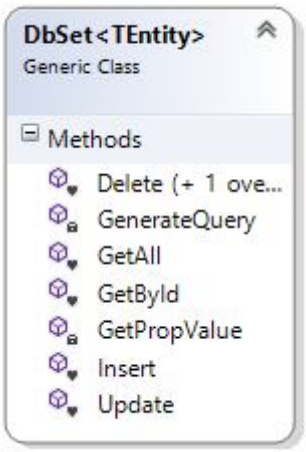

Рис.3.5 Клас DbSet

Клас DbSet (рис. 3.5) відіграє роль зв’язку з АПІ бази даних. Надає інтерфейс доступу до загальних CRUD операцій у вигляді загального класу, може бути застосований до різного роду моделей.

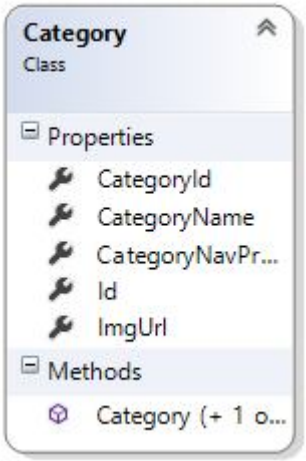

Рис.3.6 Category

Клас Category (рис 3.6) відіграє роль моделі у системі. Містить в собі властивість навігації, технологія яка була позичена в відомого EF, що дозволяє завантажувати батьківський об’єкт на етапі отримання даних.

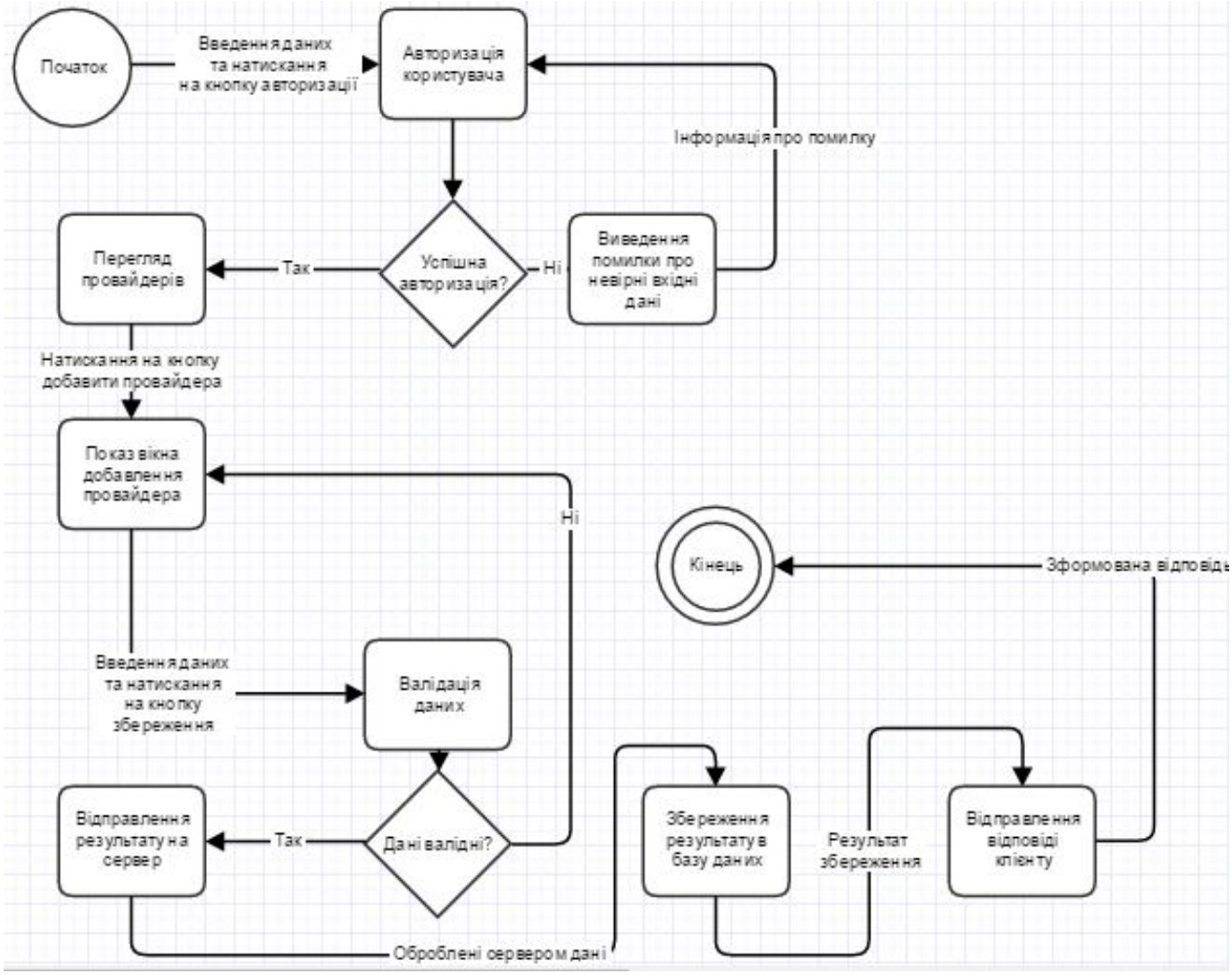

Рис. 3.7. Діаграма діяльності варіанту використання «Додавання провайдера»

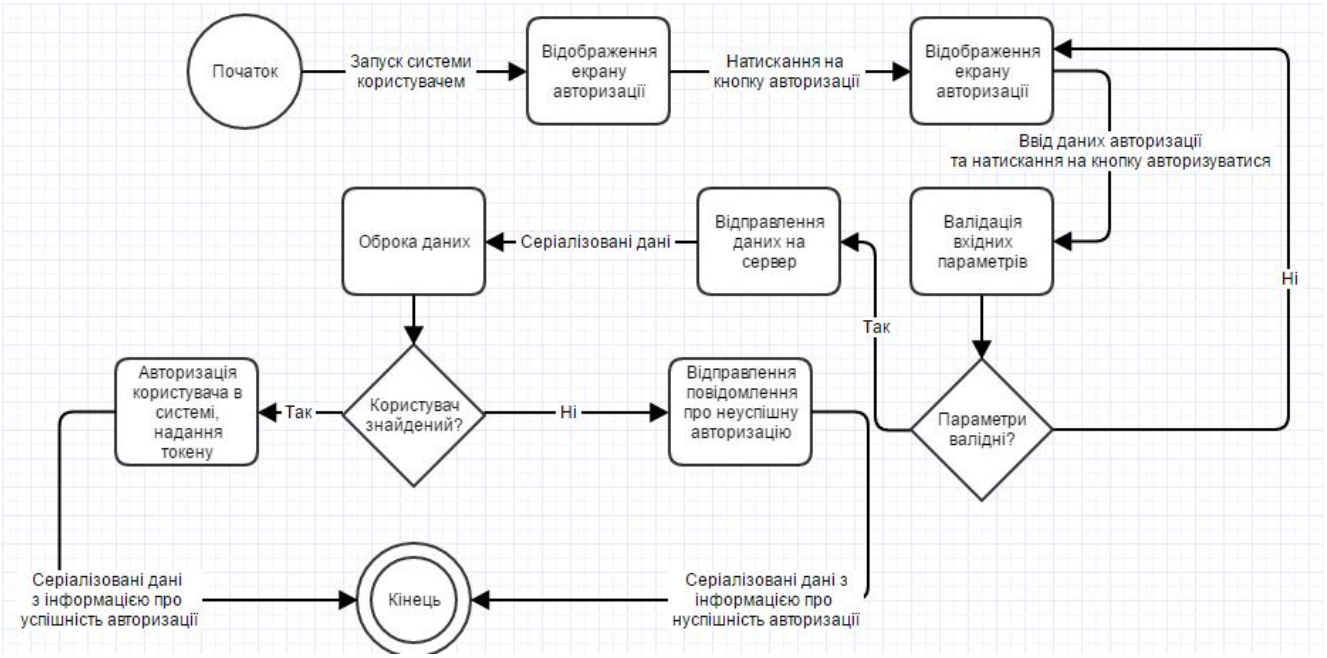

Рис. 3.8 Діаграма діяльності варіанту використання «Авторизації користувача»

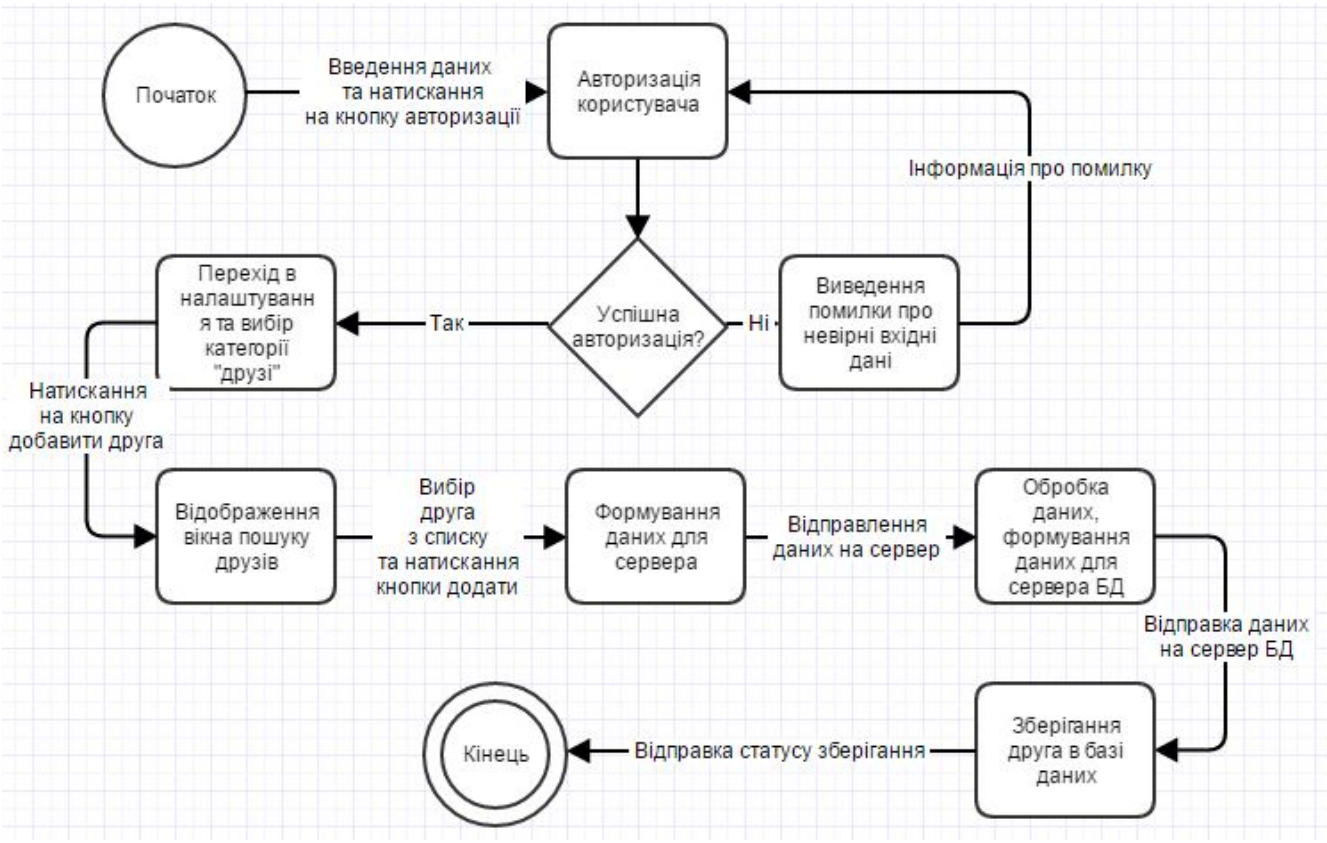

Рис. 3.9 Діаграма діяльності варіанту використання «Додавання друга користувача»

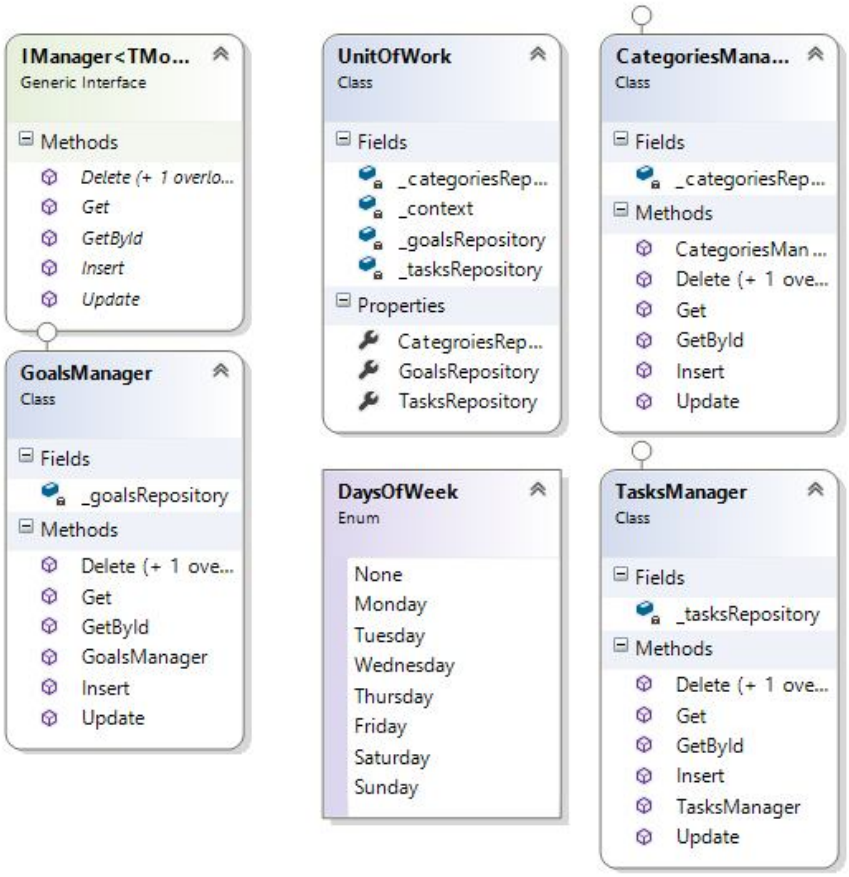

Рис. 3.10. Класи менеджерів та UnitOfWork.

Класи менеджерів (рис 3.10) відіграють роль бізнес логіки у системі. Вони слабко зв’язані з системою через принцип інверсії залежностей та абстракцію. Вони реалізовують процес роботи з даними на рівні бізнес логіки інкапсулюючи в собі клас репозиторію який використовується як інтерфейс доступу до даних. Також тут є енум тип який відображає дні тижня, при імплементації якого було використано принцип побітових значень, які в комбінації дають можливість зберігати складний стан при малих затратах пам’яті.

Інші класи у системі відіграють роль комунікаторів, або елементів доступу до БД. Також присутні елементи абстракції у вигляді абстрактних класів або інтерфейсів. Інтерфейси слугують для взаємодії із контейнером залежності.

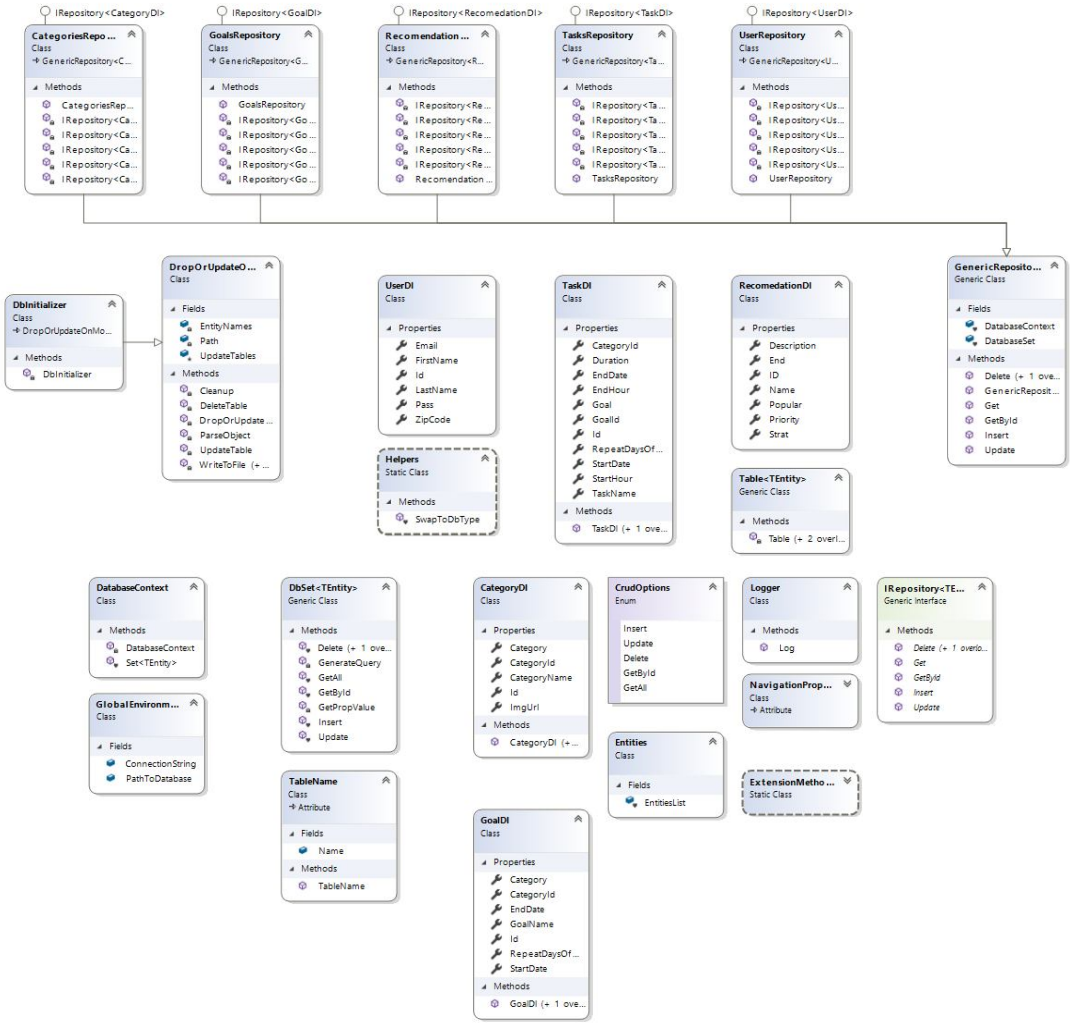

Рис.3.11. Діаграма класів рівня доступу до бази даних.

Для розробки програмного додатку було вирішено використати технлогію Xamarin Forms. Архітектурним патерном клієнтів виступає MVVM який надається бібліотекою Prism, що дозволяє використовувати звичний процес розробки як на WPF. У якості сервісу було вирішено використати ASP.NET MVC та .NET Framework 4.5. Рівень доступу до даних на рівні серверу було розроблено з використанням технології Entity Framework яка була розроблена компанією Microsoft. Основною мовою програмування вибрано C# версії 6.0, та IDE для розробки Visual Studio 2015 з емуляторами під андроїд. Для розробки під IOS було використано віртуальну машину ElCapitan. Для розгортання та тестування веб додатку використано IIS8.

C# потужний інструмент розробника при створенні застосунків. Він використовується для рішень різного виду, декстоп, веб, мобайл, сервіс аплікації. Те що це сімейство .NET робить C# дуже масштабованим, та дає можливість використовувати найкращі інструменти розробки.

Технологія ASP.NET (Active Server Pages) розроблена компанією Microsoft надає потужний інструмент в розробки веб сервісів. Влаштована інтеграція з EF зменшує час необхідний для написання коду, а Razor дозволяє швидко і ефективно вставляти C# код в сторінки, що дозволяє потужну розширюваність та зменшує необхідність написання коду на іншій мові.

Для написання клієнтів було вирішено використати експериментальну технологію Xamarin Forms. Її перевага в тому, що вона дозволяє писати звичний нам C# код, який потім транслюється в код під кожну з платформ. Це дозволяє писати так звані Shared бібліотеки, які можуть бути використані у проектах IOS Andoird Windows Phone, що дозволяє виділити спільну логіку в одне місце, і писати тільки код відносно кожної платформи, наприклад при конструюванні користувацького інтерфейсу. Для гнучкості було вирішено використати Prism MVVM, бібліотека що довзоляє використовувати патерн MVVM в технології Xamarin.

Рис 3.12 Архітектура системи на базі технології Xamarin.

Model-View-ViewModel (MVVM) - це шаблон проектування(рисунок 3.13), що застосовується під час проектування архітектури застосування (додатків). Цей шаблон забезпечує максимальне розділення логіки між в'юшкою та бізнес логікою. Зв'язок налаштовується через ViewModels, що слугують мостом між моделлю та в'ю. Через внутрішній механізм прив'язки

MVVM забезпечує повну автоматизацію оновлення в'ю при зміні моделі, що довзоляє чітко виокремити бізнес логіку від логіки користувацького інтерфейсу. Це дозволяє розробникам писати повністю розділений код.

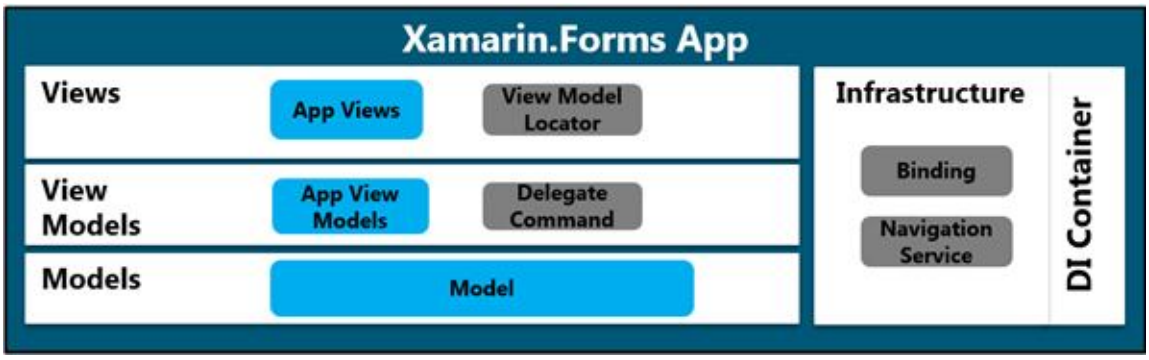

Рис.3.13. Архітектура MVVM

Розглянемо приклад реалізації контролера:

Клас CategoriesController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using WebApplication1.Repositories;

namespace WebApplication1.Controllers
{
   [Authorize]
   public class CategoriesController : Controller
   {
      private UnitOfWork _unitOfWork;

      [HttpDelete]
      public ActionResult Delete([FromBody]CategoryModel category)
      {
         _unitOfWork.CategoriesRepository.Delete(category);
         return Json("{message:success}");
      }

      [HttpDelete]
      [Route("{id}")]
      public ActionResult Delete(Int32 id)
      {
         _unitOfWork.CategoriesRepository.Delete(id);
         return Json("{message:success}");
      }

      [HttpPost]
      public ActionResult Create([FromBody]CategoryModel category)
```

```
        {
            CategoryModel createdCategory = _unitOfWork.CategoriesRepository.Create(category);
            return Json(createdCategory);
        }

        [HttpPut]
        [Route("{id}")]
        public ActionResult Update([FromBody]CategoryModel category)
        {
            CategoryModel updatedCategory = _unitOfWork.CategoriesRepository.Update(category);
            return Json(updatedCategory);
        }

        [HttpGet]
        public ActionResult Get()
        {
            List<CategoryModel> categoriesList = _unitOfWork.CategoriesRepository.Get().ToList();
            return Json(categoriesList);
        }

        [HttpGet]
        [Route("{id}")]
        public ActionResult GetById(Int32 id)
        {
            CategoryModel category = _unitOfWork.CategoriesRepository.GetById(id);
            return Json(category);
        }
    }
}
```

Цей клас відповідає за роботу з сутністю категорій. Використовуючи UnitOfWork він налаштовує доступ до даних через внутрішньо інкапсульовані репозиторії, та представляє REST доступ до даних. Контролер має атрибут Authorize що дозволяє передбачити неавторизоване втручання і повернення статусу 401(unauthorized) якщо користувач попередньо не авторизувався в системі.

Рівень доступу до даних був розроблений на основі патеру Repository та UnitOfWork. Патер репозиторій забезпечує ефективний абстрактний проміжок, що забезпечує абстрагування від внутрішньої реалізації і працює на рівні інтерфейсу. Це дозволяє. При потребі швидко замінити один провайдер на інший без великих затрат.

Клас UnitOfWork

```
using TimeZilla.DataLayer.Database;
using TimeZilla.DataLayer.Interfaces;
using TimeZilla.DataLayer.Models;
using TimeZilla.DataLayer.Repositories;
```

```
namespace TimeZilla.BL
{
    public class UnitOfWork
    {
        public IRepository<CategoryDl> CategroiesRepository
        {
            get
            {

                if (this._categoriesRepository == null)
                {
                    this._categoriesRepository = new CategoriesRepository(_context);
                }
                return _categoriesRepository;
            }
        }

        public IRepository<GoalDl> GoalsRepository
        {
            get
            {

                if (this._goalsRepository == null)
                {
                    this._goalsRepository = new GoalsRepository(_context);
                }
                return _goalsRepository;
            }
        }

        public IRepository<TaskDl> TasksRepository
        {
            get
            {

                if (this._tasksRepository == null)
                {
                    this._tasksRepository = new TasksRepository(_context);
                }
                return _tasksRepository;
            }
        }

        public IRepository<RecomendationDl> RecomendationsRepository
        {
            get
            {

                if (this._recomendationsRepository == null)
                {
                    this._recomendationsRepository = new RecomendationsRepository(_context);
                }
                return _recomendationsRepository;
```

```
        }
    }

    private readonly DatabaseContext _context = new DatabaseContext();
    private IRepository<CategoryDl> _categoriesRepository;
    private IRepository<GoalDl> _goalsRepository;
    private IRepository<TaskDl> _tasksRepository;
    private IRepository<RecomendationDl> _recomendationsRepository;
  }
}
```

Лістинг коду класу UnitOfWork. Це клас забезпечує що на час життя контролера буде створений тільки один репозиторій для кожної сутності, що забезпечує відправку одного реквесту до БД,що забезпечує менше навантаження та більшу швидкодію.

Розглянемо архітектуру доступу до БД в деталях. На рисунку 3.14 наведено UML діаграму класів на рівні доступу до даних.

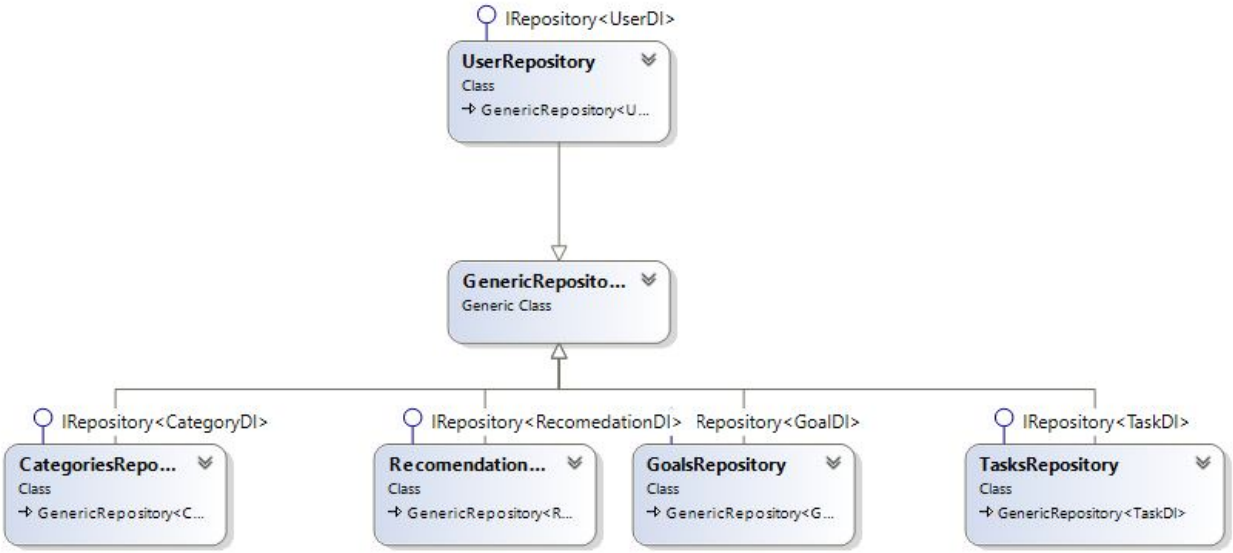

Рис.3.14 UML діаграма класів рівня доступу до даних

Як можна побачити з рисунку 3.14, існує основний клас із реалізацією методів взаємодії із БД, а в дочірніх класах є реалізація методів для взаємодій із рівнем бізнес логіки. Програмний код основних класів кожного із рівнів представлений у додатку А.

3.2 Проектування структури бази даних

Проаналізувавши вербальний опис можна збудувати діаграму елементів та зав’язків (див. рис. 3.16).

На базі отриманої діаграми можна приступати до проектування елементів у БД та побудови ER-діаграми. Взявши усі сутності та звязки ми переносимо їх у діаграму сутностей та звязків, де інформаційними сутностями виступають елементи із діаграми елементів та звязків, котрі містять у собі інші сутності. Отже, можна виділити такі інформаційні сутності як Criteria, CriteriasKeywords, Elements, Providers, User, Device, OS, Categories.

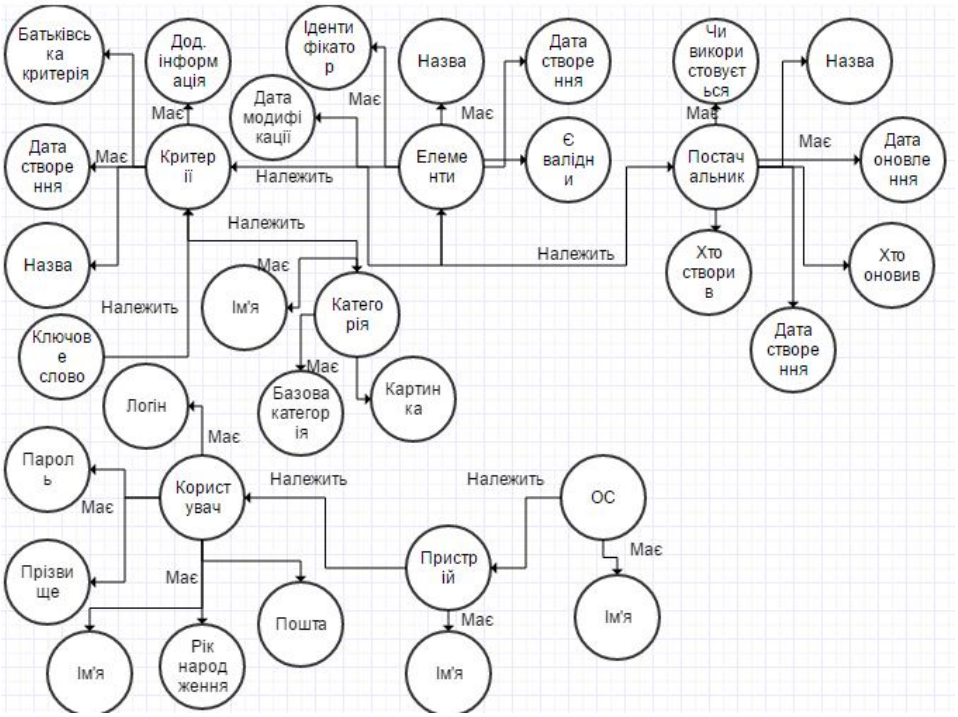

Рис. 3.16 Діаграма елементів і зв’язків

Опис ідентифікаторів та планування індексів представлено у таблицях 3.1 і 3.2 відповідно.

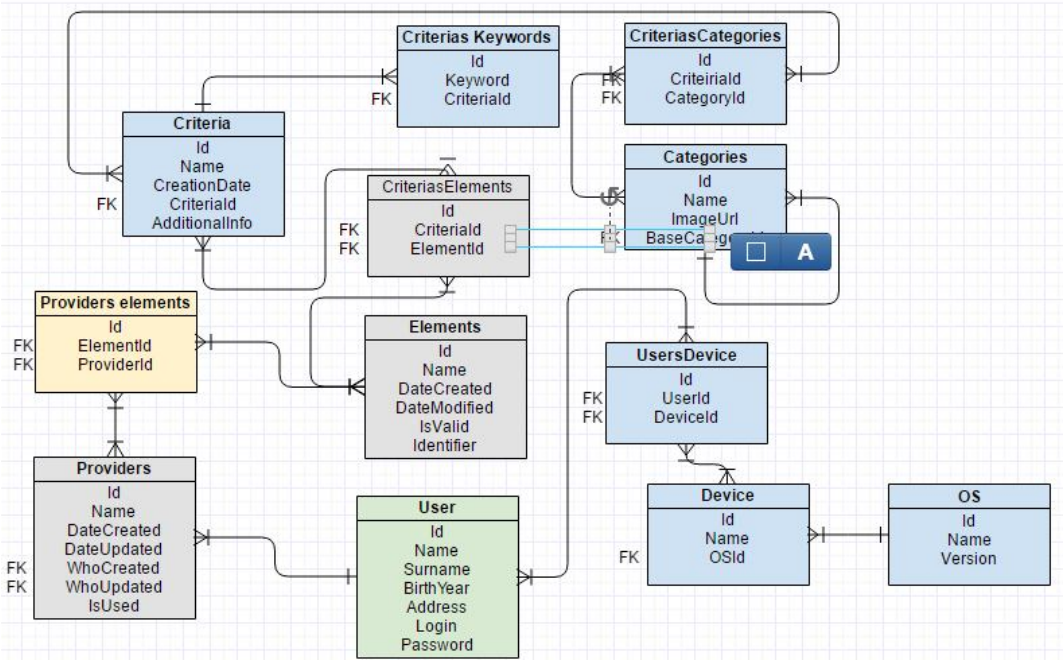

Рис. 3.17 ER-діаграма

Таблиця 3.1

Таблиця ідентифікаторів

Об’єкт	Властивість	Тип	Розмірність	Ідентифікатор
Criteria	Name	varchar	45	Name
	CreationDate	date		CreationDate
	CriteriaId	int		CriteriaId
	AdditionalInfo	varchar	300	Info
Provider	Name	varchar	45	Name
	DateCreated	date		CreationDate
	DateUpdated	date		DateUpdated
	WhoCreated	int		Creator
	WhoUpdated	int		WhoUpdated
	IsUsed	bit		IsUsed
Elements	Name	varchar	45	Name
	DateCreated	date		CreationDate
	DateModified	date		DateUpdated
	IsValid	bit		IsValid
	Identifier	varchar	300	Identifier
Categories	Name	varchar	45	Name
	ImageURL	varchar	300	ImgUrl
	BaseCategory	int		BaseCategory
Device	Name	varchar	45	Name
	OsId	int		OsiD
OS	Name	varchar	45	Name
	Version	varchar	45	Version

Таблиця 3.2

Таблиця планування індексів

Назва таблиці	Атрибут	Опис
Criteria	CriteriaId	Foreignkey
CriteriasElements	CriteriaId	Foreignkey
CriteriasElements	ElementId	Foreignkey
ProvidersElements	ElementId	Foreignkey
ProvidersElements	ProviderId	Foreignkey
Providers	UserId	Foreignkey
Providers	UserId	Foreignkey
CriteriasKeywords	CriteriaId	Foreignkey
CriteriasCategories	CriteriaId	Foreignkey
CriteriasCategories	CategoryId	Foreignkey
UsersDevice	UserId	Foreignkey
UsersDevice	DeviceId	Foreignkey
Categories	CategoryId	Foreignkey
Device	OSId	Foreignkey

Доступ до БД на рівні сервера буде здійснений за допомогою EF ORM технології, що дозволить сильно пришвидшити процес написання коду, оскільки непотрібно буде писати складні SQL запити. На клієнті було вирішено написати кастомну ORM систему на базі EF. Вона працює по схожих принципах але огортає в собі API для роботи з ADO.NET провайдером під SQLite бази даних.

Кожна сутність що на сервері що на клієнті представляється моделлю, тому для кожної таблиці буде створена своя модель, яка буде представляти таблицю на рівні бізнес логіки. На рис 3.18 приклад декількох вже створених моделей.

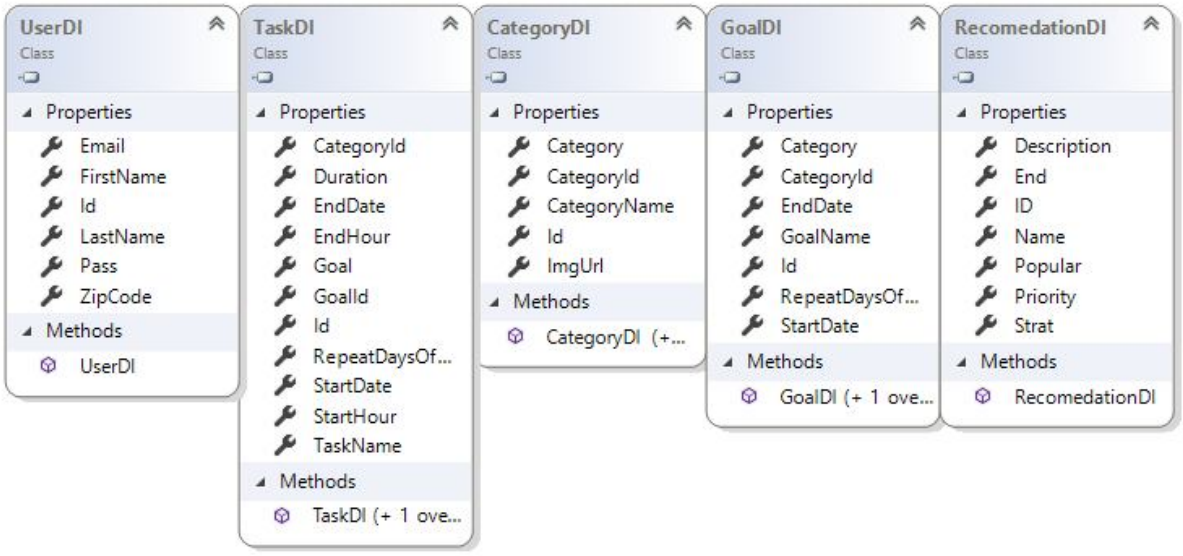

Рис. 3.18. Діаграма класів «Проксі класів»

3.3 Програмування бази даних

Для реалізації доступу до даних було вирішено використати підхід EntityFrameowork Code First. Цей підхід дозволяє швидко і ефективно конвертувати моделі написані на c# коді в таблиці баз даних, а потужний інтерфейс LINQ2SQL надає провайдери для роботи з базою, що забезпечує гнучку генерацію запитів.

Сервером БД було вирішено використати Microsoft SQL Server 2012, тоді як на клієнтах використовується локальні бази на основі SQLite. Сервер 2012 дозволяє створювати визначені користувачем серверні ролі, що полегшує процес масштабування додатку. Також з 2012 сервером прийшла підтримка BigData.

Нижче наведений приклад як було використано моделі при створенні таблиць у БД.

Клас DatabaseInitializer

```
using System;
using System.IO;
using Mono.Data.Sqlite;
using TimeZilla.DataLayer.Models;

namespace TimeZilla.DataLayer.Database
{
```

```
    internal class DbInitializer : DropOrUpdateOnModelChanged
    {
        static DbInitializer()
        {
            String pathToDatabase = GlobalEnvironment.PathToDatabase;

            bool exists = File.Exists(pathToDatabase);

            if (!exists)
            {
                SqliteConnection.CreateFile(pathToDatabase);
            }

            UpdateTables = true;

            #region Tables

            Activator.CreateInstance<GoalDl>();

            Activator.CreateInstance<CategoryDl>();

            Activator.CreateInstance<TaskDl>();

            #endregion

        }
    }
}
```

У даному прикладі показано як відбувається ініціалізація базового рівня додатку. При наслідуванні від DropOrUpdateOnModelChanged ми вказуємо що при зміні моделей наші таблиці у БД будуть відповідно змінені, і якщо врахувати UpdateTables поставлений у true, це означає що таблиці не будуть втрачати попередні дані, а буде виконана міграція.

Також через Activator.CreateInstance запускається процес ініціалізації конструктора вказаних типів, які в свою чергу генерують таблиці.

3.4 Тестування та дослідна експлуатація

Етап тестування є одним із найважливіших етапів у розробці програмного забезпечення, адже помилки в роботі програми можуть призвести до великих матеріальних витрат. За способом виконання тестів розрізняють автоматизоване та ручне тестування. Основна відмінність між цим підходами

полягає у самому способі тестування програмного продукту: при автоматизованому підході використовуються спеціальні програмні продукти типу Selenium та Ranorex. Написання автоматизованих тестів є клопіткою справою та потребує багато часу та грошей, адже потрібно найти спеціалістів із правильними навичками, або самому потратити час на дослідження даної області.

Процес тестування – один із найважливіших етапів життєвого циклу проекту, адже він направлений на виявлення дефектів програмної системи, з метою їх подальшого усунення. Результати тестування можуть оцінюватись на основі критеріїв:

- відповідність вимогам, якими керувалися проектувальники та розробники
- правильна відповідь для усіх можливих вхідних даних
- виконання функцій за прийнятний час
- практичність
- сумісність з програмним забезпеченням та операційними системами
- відповідність задачам замовника.

Є дуже багато видів та методів тестування, наприклад, тестування білої скриньки, тестування чорної скриньки, димове тестування, модульне тестування, тестування навантаження, тестування безпеки.

Метою тестування веб-орієнтованого додатку є знайти його слабкі місця і визначити наскільки ефективно відбувається обмін даними між клієнтом та сервером, визначити можливі помилки у інтерфейсі та те, наскільки коректно відображається сторінка на різних браузерах. Було проведено функціональне тестування та тестування інтерфейсу. Було створено специфікацію тестів для функціонального тестування та тестування інтерфейсу користувача.

Таблиця 3.3

Дефекти, які виявлені під час функціонального тестування

Дефект	Опис дефекту	Вирішення дефекту
При введенні поля електронної адреси, поле допускає введення любого набору даних	Дефект полягає у тому що система повинна валідувати введені дані про електронну адресу за допомогою регулярного виразу	Додавання перевірки даних через регулярний вираз на етапі набору.
При виборі показати статистику інформація про статистику відображається в неправильних часових рамках	Дефект полягає у тому, що типу дати неправильно конвертується при обробці її на веб АПІ.	Додавання кастомної опції до процесу ініціалізації БД, що дозволяє вручну задати тип даних для дати, а не автоматично. Це дозволить поставити коректний тип даних.
Можна зареєструвати користувача із вже існуючим логіном	Дефект полягає у тому, що немає перевірки на існування вже існуючого користувача	Потрібно добавити перевірку при додаванні на існування користувача із такою інформацією

Одним із найважливіших аспектів тестування веб сервісу це тестування безпеки. Оскільки через реінженерію можна дізнатися роутинг додатку, додаток повинен бути захищений від неавторизованого доступу.

Для організації безпечного доступу у додатку була використана технологія JWT(Json Web Token) та технологія Asp.Net Identity. В комбінації ці дві технології дозволили створити гнучкий продукт який захищений від несанкціонованого доступу і потребує попереднього процесу авторизації для ідентифікації користувача і надання йому токена, за допомогою якого в

подальшому користувач може бути ідентифікований, і в залежності від його прав він буде мати доступ до певних роутів.

Для тестування безпеки ресурсу було використано програмний продукт Postman. Основний інтерфейс даної програми представлений на рисунку 3.19.

Для тестування безпеки ми спробуємо неавторизованим користувачем отримати доступ до даних веб АПІ. (рис. 3.20).

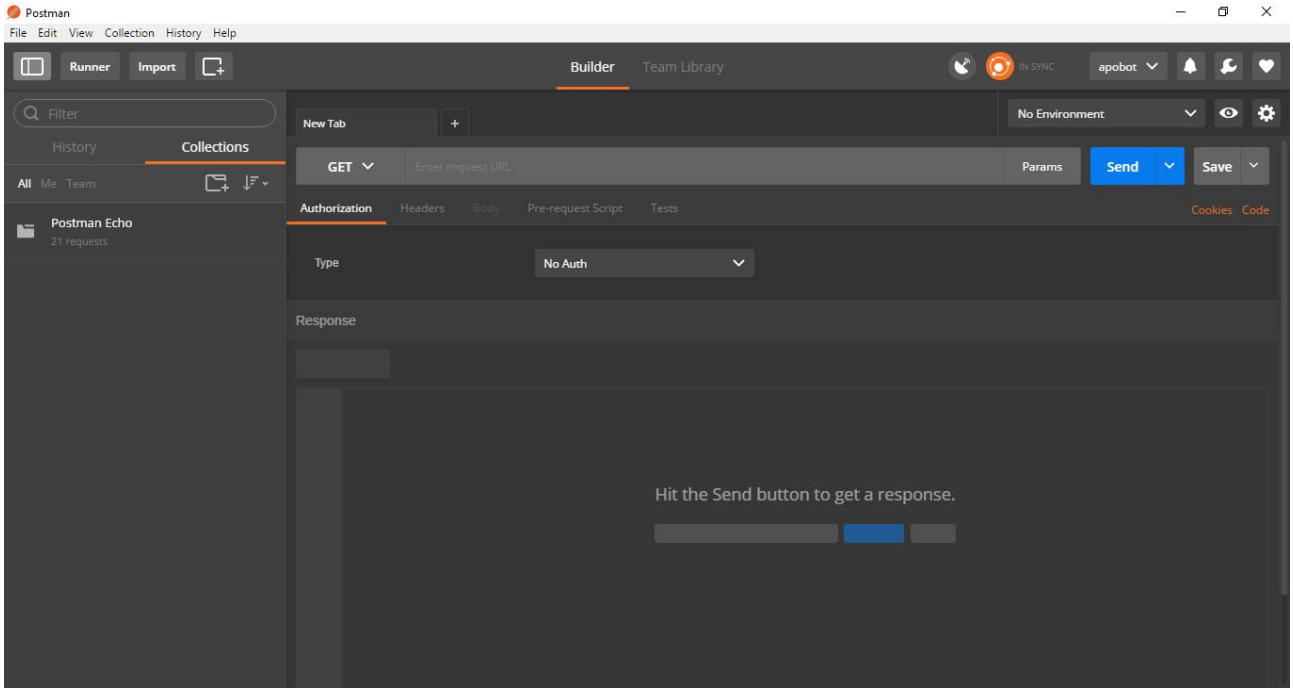

Рис. 3.19 Основне вікно роботи із PostMan

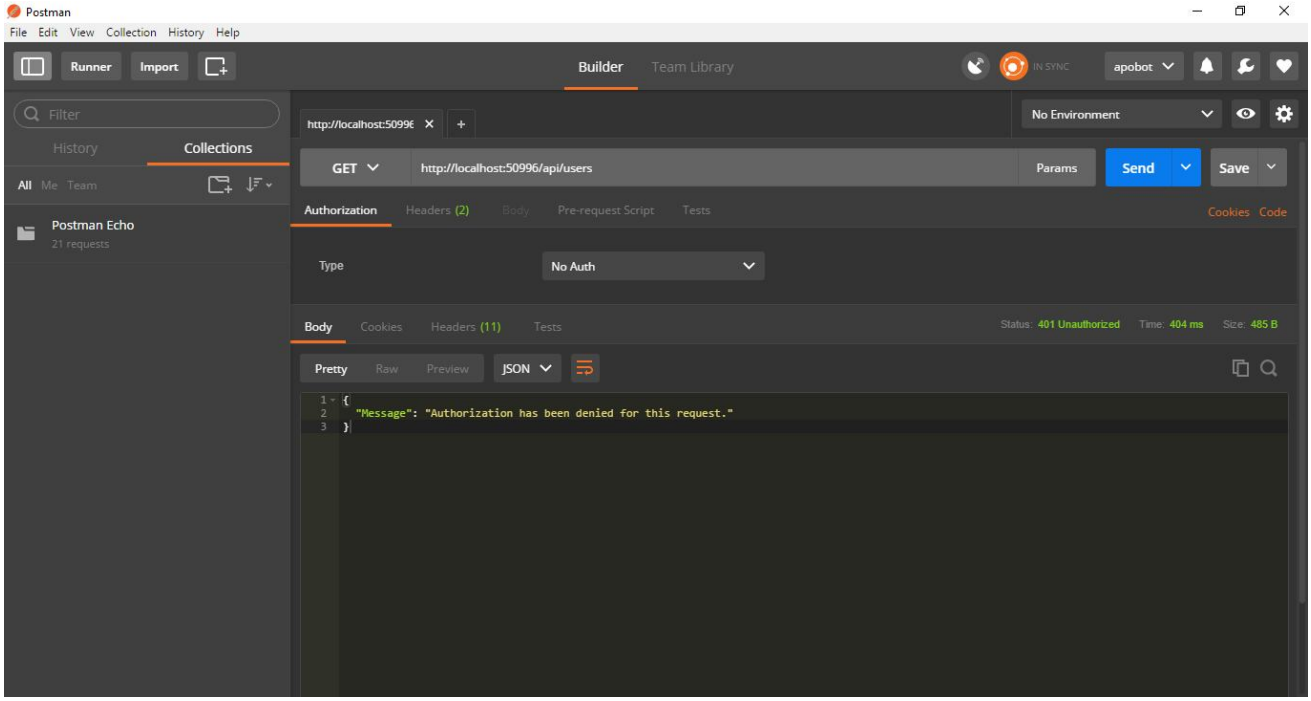

Рис. 3.20 Вікно неавторизованого запиту.

Результат тестування веб-орієнтованої програмної системи є позитивним.

Наступним етапом є тестування інтерфейсу користувача. Враховуючи що цей додаток орієнтований на мобільні платформи, тестування повинне бути проведене з урахуванням різних розширень екранів, а також частина тестування зроблена на планшетах. Складність тестування досить велика, оскільки необхідно перевірити функціональність на device-native controls.

Під час тестування відображення було виявлено ряд дефектів представлених у таблиці 3.5.

Таблиця 3.5

Дефекти, які виявлені під час тестування інтерфейсу

Дефект	Опис дефекту	Вирішення дефекту
При відображенні контролів введення даних, на IOS вони зміщені на 20 пікселів вверх	Дефект полягає у специфічному відображенні користувацького інтерфейсу на IOS.	Потрібно в рендеринг контролів додати OnDevice метод та визначити розміри в залежності від пристрою.
При відображенні статистичних даних, графіки некоректно відображаються.	Дефект полягає у тому що неправильно прораховані розміри базового контейнера для графіків.	Потрібно змінити розміри з статичних на динамічні з урахуванням пристроїв.

Після проведення тестування інтерфейсу було виправлено ряд суттєвих дефектів, які впливали на відображення системи.

3.5 Розгортання програмного продукту

Для розгортання програмної системи необхідно задовольнити вимоги, що наведені нижче:

- Для веб сервіса:

- операційна система Microsoft Windows 8.1/10;
- SQL Server 2012;
- web-сервер IIS 7/8;
- Microsoft .NET Framework4.5

- Для мобільного додатку:
  - операційна система Android 6.0 Marshmallow
  - операційна система IOS9,IOS10.

Для формування апаратних вимого потрібно провести аналіз кількості користувачів, котрі будуть використовувати розроблену систему. Для невеликої кількості (до 1000 одночасно) користувачів достатньо наступної конфігурації для апаратних ресурсів:

- Для веб сервіса:

- процесор з тактовою частотою не менше 2 Ghz;
- оперативна пам’ять в межах 16Gb;
- система потребує великої кількості місця на жорсткому диску для зберігання бази даних та зображень. Тому об’єм жорсткого диску має бути не менше 1Tb;
- рекомендована швидкість з’єднання становить від 100 Mb/s.

- Для мобільного додатку:

- процесор з тактовою частотою не менше 2 Ghz;
- оперативна пам’ять в межах 1Gb;
- необхідно пам’яті для розгортання 256мб

Для розгортання веб системи на сервері потрібно спочатку збілдити систему у реліз версії. Далі потрібно налаштувати деплоймент опції на веб сервері, для потрібно провести потужну конфігурацію IIS сервера, налаштувати порти, відкрити доступа. Також на сервері системи потрібно налаштувати релізні системні змінні, для того щоб веб-сервер працював максимально ефективно. Далі за допомогою застосунків VS в залежності від того по якому протоколу буде відбуватися розгортання, вибрати його на слідувати майстру розгортання.

Для розгортання мобільного додатку, необхідно підключити пристрій до

комп’ютера та перемістити файл для інсталяції на мобільний пристрій. Після переносу додатку слід відкрити інсталяційний файл з мобільного пристрою. В результаті буде проведена збереження. У випадку з IOS потрібно зайти в епл-стор і скачати додаток через звичний для вас процес.

Для роботи програмно продукту потрібно налаштувати середовище згідно попередніх вимог. Також середовище роботи повинно включати в собі бібліотеки наведені в таблиці 3.5.

Таблиця 3.5

Набір бібліотек для коректної роботи програми

№	Назва бібліотеки	Призначення
1.	System.Web.Mvc.dll	головна бібліотека фреймворку ASP.NET MVC
2.	System.Web.Razor.dll	бібліотека для роботи з формуванням інтерактивного та гнучкого користувацького інтерфейсу
	Microsoft.AspNet.Identity.Core	Бібліотека для забезпечення безпечного доступу до веб сервісу та ідентифікації користувача.
3.	TimeZilla.Domain.dll	бібліотека для визначення сутностей предметної області
4.	TimeZilla.DAL.dll	бібліотека для реалізації доступу до даних
5.	TimeZilla.BLL.dll	бібліотека для реалізації всієї бізнес-логіки
	System.Attributes	бібліотека для використання атрибутики.
6.	Microsoft.AspNet.Identity.EntityFramework	Допоміжна бібліотека для роботи з Asp.Net identity через EF.
	EntityFramwork	Бібліотека потужної ОРМ що була використана для роботи з БД.
	Newtonsoft.Json	Бібліотека для роботи з Json.
6.	Web.config	файл налаштувань програмної системи

Для роботи системи та можливості доступу до бази даних необхідно вказати стрічку з’єднання із сервером бази даних в файлі Web.config. Приклад з’єднання стрічки наведений нижче:

<connectionStrings>

<add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet-WebApplication2-20170212022549.mdf;Initial Catalog=aspnet-WebApplication2-20170212022549;Integrated Security=True" providerName="System.Data.SqlClient" /> </connectionStrings>

Програмний код основних класів кожного із рівнів представлений у додатку А.

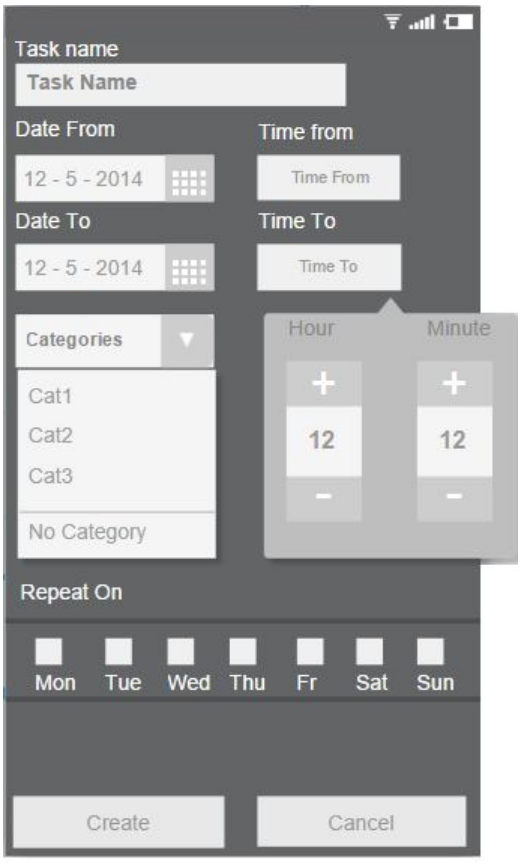

Рис. 3.21 Вікно створення нової задачі.

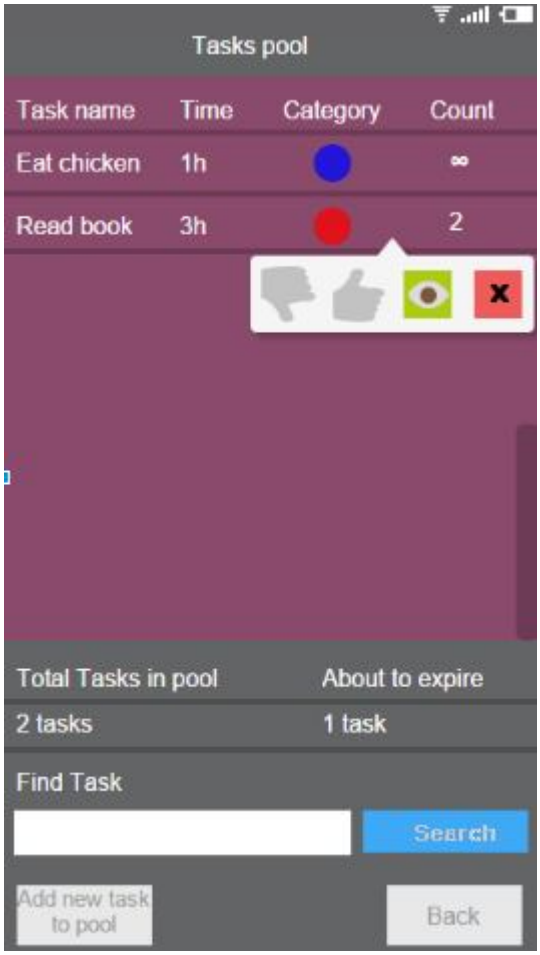

Рис. 3.22 Вікно перегляду набору задач.

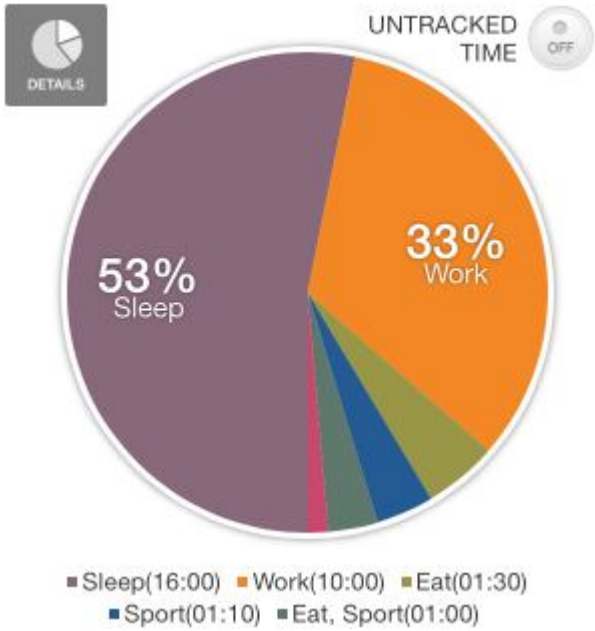

Рис. 3.23 Аналітика за категорією

Висновки до 3 розділу

1. Спроектовано та розроблено програмну систему «TimeZilla».

2. Проведено модульне, інтеграційне, функціональне тестування графічного інтерфейсу та тестування безпеки.

3. Розроблено технологічну інструкцію для користувача та системного адміністратора.

ВИСНОВКИ

В результаті виконання магістерської роботи був розроблений алгоритм пошуку даних в умовах невизначеності, погрупований з використанням RB дерев та системи передбачень. Розроблений модуль надає можливість швидко і ефективно надавати базу для формування рекомендацій щодо заданих параметрів користувачів, що в свою чергу сприяє більш ефективному розділенню часу.

Розроблений продукт «TimeZilla» дозволить користувачам більш ефективніше слідкувати за своїм часом. Гнучка система рекомендацій дозволяє швидко і ефективно отримувати найновіші рекомендації відносно вподобавнь користувача або його друзів, а вебсервіс з синхронізацією даних забезпечую збереження даних та їх безпеку.

Для досягнення мети в магістерській роботі вирішено наступні задачі:

1) Проведено аналіз та порівняння існуючи алгоритмів для роботи з невизначеними даними. Було визначено що такі алгоритми є досить ефективними, але не є гнучкими і потребують налаштування, тому було запропоновано модифікувати процес отримання даних від просто дерева до RB дерев, що забезпечує приріст в швидкодії у порівнянні з простими дерева, оскільки процес аналізу даних, надання їм тегів та зберігання їх в структуру є складним і постійно потребує різного виду вставок/видалень елементів. Також застосовано алгоритми передбачень до процесу пошуку даних у випадку якщо дані не повні або схожість не була знайдена .

2) Проведено теоретичні дослідження основних підходів, які відносяться до формалізації предметної області. Запропонований механізм вирішення задач наповнення даних у системах тайм менеджменту, а саме процедура аналізу зовнішнього тексту, генерація на його основі ІОБ РБ дерев, та аналіз вхідних даних в умовах невизначеності.

3) На основі результатів аналізу функціональних, нефункціональних та архітектурних вимог здійснено проектування та програмну реалізацію розроблених методів та алгоритмів, та системи в цілому, проведене відповідне

тестування (модульне, інтеграційне, функціональне, тестування графічного інтерфейсу, тестування безпеки).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

4. Шилдт Г. C# 4.0 Полное руководство – Вильямс, 2013. - 1056 с.

5. Мартин Р., Чистий код: создание, анализ и рефрактинг. Библиотека програмиста. – СПб.: Питер, 2010 р. – 466 с.

6. Lerman J. - Programming Entity Framework. Code First – 2011/ Lerman J., Miller R.

7. Bloesch, A. & Halpin, T. 1997, ‘Conceptual queries using ConQuer-II’, Proceedings of the 16th International Conference on Conceptual Modeling ER'97 (Los Angeles), Springer LNCS 1331 (Nov.) 113- 126.

8. Data mining [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Data_mining

9. Halpin, T.A. & Bloesch, A.C. 1999, ‘Data modeling in UML and ORM: a comparison’, Journal of Database Management, vol. 13, no. 2, Idea Group Publishing, Hershey PA, pp. 20-30.

10. Halpin, T.A. & Ritson, P.R. 1992, ‘Fact-oriented modelling and null values’, Research and Practical Issues in Databases, eds B. Srinivasan & J. Zeleznikov, World Scientific, Singapore.

11. Halpin, T.A. 1998, ‘ORM/NIAM Object-Role Modeling’, Handbook on Information Systems Architectures, eds P.Bernus, K. Mertins & G. Schmidt, Springer-Verlag, Berlin, pp. 81-101.

12. IEEE Std. 610.12:1990. IEEE Standard Glossary of Software Engineering Terminology.

13. База даних [Електронний ресурс]. – Режим доступу: http://www.unicyb.kiev.ua/~boiko/pr2k/dbintro.htm

14. IOB trees [Електронний ресурс] – Режим доступу - http://surnames.meaning-of-names.com/family-trees/iob/.

15. Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних і знань. Книга 1. Організація баз даних і знань : навч. посіб. Львів : Магнолія, 2011. – 456 с.

16. Extracting information from text [Електронний ресурс] – Режим доступу. http://www.nltk.org/book/ch07.html

17. Бізнес-процес [Електронний ресурс]. – Режим доступу: http://www.rusnauka.com/6_PNI_2012/Economics/6_102471.doc.htm

18. Білас О. Є. Якість програмного забезпечення та тестування : навч. посібн. - Львів : Видавництво Львівської політехніки, 2013. - 216 с.

19. Вищий навчальний заклад [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Вищий_навчальний_заклад

20. Вікіпедія [Електронний ресурс]. – Режим доступу: http://uk.wikipedia.org/wiki/

21. Дивак М. П., Шпінталь М.Я., Шевчук Р.П., Козак О.Л., Пукас А.В., Спільчук В.М., Гончар Л.І. Методичні рекомендації до виконання та захисту дипломної роботи на здобуття освітньо-кваліфікаційного рівня магістр за спеціальностями: 8.05010301 “Програмне забезпечення систем” та 8.05010302 “Інженерія програмного забезпечення” // Тернопіль : Економічна думка, 2011. – 31 с.

22. Дин Леффингуэлл, Дон Уидриг. Принципы работы с требованиями к программному обеспечению. Унифицированный подход. – М.: Вильямс, 2012. – 448 с.

23. Joachims, T. (2003) Optimizing Search Engines using Clickthrough Data // Proceedings of the ACM Conference on Knowledge Discovery and Data Mining

24. Коротун Т.М. Совершенствование процесса тестирования ПО // Проблемы программирования. – 1998. - № 3 – С.59-64.

25. Майерс Г. Искусство тестирования программ. – М: Финансы и статистика. - 1982. – 176 с.

26. Павловская Т. С#. Програмирование на языке высокого уровня: Учебник для вузов. - СПб. : Питер, 2010. – 432 с.

27. Тарасов О. В., Федько В. В, Лосєв М. Ю. Організація баз даних та знань. Проектування баз даних : навч.-практ. Посіб. - Х. : ХНЕУ, 2011. – 200 с.

28. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ. - 2-е изд. - М.: «Вильямс», 2006. - 1296 с.