Зміст

[Анотація 4](#_Toc530646353)

[Abstract 5](#_Toc530646354)

[Вступ 6](#_Toc530646355)

[Розділ 1. Огляд електронних словників та інструментів для їх побудови 7](#_Toc530646356)

[1.1. Поняття електронного словника 7](#_Toc530646357)

[1.1.1. Сучасна лексикографія: можливості електронних словників 7](#_Toc530646358)

[1.1.2. Особливості представлення лексикографічного матеріалу в цифровому середовищі 11](#_Toc530646359)

[1.2 Алгоритми опрацювання тексту 12](#_Toc530646360)

[1.2.1. Алгоритми морфологічного аналізу 14](#_Toc530646361)

[1.2.1. Алгоритми лексичного аналізу 19](#_Toc530646362)

[1.2.2. Алгоритми синтаксичного і дискурсивного аналізу 24](#_Toc530646363)

[Розділ 2. Постановка задачі розробки програмного продукту для формування лексико-семантичного словника української мови 33](#_Toc530646364)

[2.1. Загальна постановка задачі 33](#_Toc530646365)

[2.2. Вибір інструментальних засобів 33](#_Toc530646366)

[2.3. Специфікація вимог до програмного продукту 34](#_Toc530646367)

[2.3.1. Вступ 34](#_Toc530646368)

[2.3.1.1. Призначення, мета 34](#_Toc530646369)

[2.3.1.2. Продукти аналоги 34](#_Toc530646370)

[2.3.2. Загальний опис 34](#_Toc530646371)

[2.3.2.1. Характеристики продукту 34](#_Toc530646372)

[2.3.2.2. Середовище функціонування 35](#_Toc530646373)

[2.3.2.3. Характеристики системи 35](#_Toc530646374)

[2.3.2.4.1. Добавлення нового слова до словника 35](#_Toc530646375)

[2.3.2.4.2. Редагування та видалення слів 36](#_Toc530646376)

[2.3.2.4.3. Добавлення до поточного слова певних ознак 36](#_Toc530646377)

[2.3.2.4.4. Редагування ознак слова(видалення та зміна) 37](#_Toc530646378)

[2.3.2.4.5. Редагування дерева ознак 37](#_Toc530646379)

[2.3.2.4.7. Пошук слів у словнику 38](#_Toc530646380)

[2.3.2.4.8. Швидкий доступ до останніх доданих слів 39](#_Toc530646381)

[2.3.2.4.9. Додавання нових слів із файлу 39](#_Toc530646382)

[2.3.2.4.10. Автоматичне наповнення словника 40](#_Toc530646383)

[2.3.3. Вимоги зовнішніх інтерфейсів 40](#_Toc530646384)

[2.3.3.1. Користувацькі інтерфейси 40](#_Toc530646385)

[2.3.3.2. Програмні інтерфейси 41](#_Toc530646386)

[2.3.3.3. Комунікаційні інтерфейси 41](#_Toc530646387)

[2.3.4. Нефункціональні вимоги 41](#_Toc530646388)

[2.3.4.1. Вимоги продуктивності 41](#_Toc530646389)

[2.3.4.2. Вимоги безпеки 41](#_Toc530646390)

[2.3.4.3. Вимоги надійності 41](#_Toc530646391)

[Розділ 3. Проектування та архітектура підсистеми для формування та друку документації та звітності у ІС «Деканат» 42](#_Toc530646392)

[3.1. Архітектура програмного продукту 42](#_Toc530646393)

[3.2. Проектування бази даних 42](#_Toc530646394)

[3.2.1. Концептуальна модель бази даних 43](#_Toc530646395)

[3.2.2. Логічна модель бази даних 44](#_Toc530646396)

[3.2.3. Фізична модель бази даних 44](#_Toc530646397)

[3.3. Проектування об’єктної моделі 49](#_Toc530646398)

[3.4. Проектування графічного інтерфейсу користувача 49](#_Toc530646399)

[Розділ 4. Реалізація та тестування підсистеми формування та друку документації та звітності у ІС «Деканат» 51](#_Toc530646400)

[4.1. Реалізація підсистеми 51](#_Toc530646401)

[4.1.1. Проектування звітів 51](#_Toc530646402)

[4.1.2. Проектування інтерфейсу користувача 56](#_Toc530646403)

[4.1.3. Реалізація функціоналу системи 58](#_Toc530646404)

[4.2. Тестування системи. 59](#_Toc530646405)

[Розділ 5. Економічна частина 60](#_Toc530646406)

[5.1. Економічна характеристика проектного рішення (програмного продукту) 60](#_Toc530646407)

[5.2. Інформаційне забезпечення та формування гіпотези щодо потреби розроблення товару. 60](#_Toc530646408)

[5.3. Оцінювання та аналізування факторів зовнішнього та внутрішнього середовищ. 61](#_Toc530646409)

[5.4. Формування стратегічних альтернатив 63](#_Toc530646410)

[5.5. Бюджетування 66](#_Toc530646411)

[5.6. Остаточний вибір стратегії 71](#_Toc530646412)

[Висновки 73](#_Toc530646413)

[Список використаної літератури 74](#_Toc530646414)

[Додаток А 75](#_Toc530646415)

[Додаток Б 83](#_Toc530646416)

[Додаток В 91](#_Toc530646417)

# Анотація

# Abstract

# Вступ

Значення словників у житті кожної людини важко переоцінити. Словник — джерело, яке містить слова розташовані в певному порядку з відповідним роз’ясненням до них. Вони є одним із засобів накопичення результатів пізнавальної діяльності людства, показником культури народу. Одні з найбільших колекцій словників знаходяться в бібліотеках різних країн світу. Вони використовуються при таких задачах:

* наукова обробка документів: при систематизації та пересистематизації документів, для якісного лінгвістичного забезпечення електронних каталогів (формулювання предметних рубрик, ключових слів тощо);
* інформаційно-бібліографічна робота: відповіді на численні бібліографічні довідки тематичного і фактографічного характеру, відповіді на запити віртуальних довідкових служб;
* при обслуговуванні студентів-іноземців, яких у вишах стає все більше.

Користувачі словникової літератури використовують її в друкованому вигляді, але останнім часом все частіше і частіше звертаються до електронних словників. Багато паперових словників були переведені в електронний вигляд. Завдяки "перекладу" багатотомних важких паперових словників в електронну форму з'явилася можливість обладнати такі довідники потужними і зручними механізмами пошуку, а також системами гіпертекстових посилань. Електронний словник — комп'ютерна база даних у комп'ютері або іншому пристрої. Ці словники дозволяють швидко знайти потрібне слово, часто з урахуванням морфології і можливістю пошуку словосполучень (прикладів вживання), а так само з можливістю перекладу на інші мови.

В результаті цього виникає потреба написання програмного продукту, який дозволить структурувати слова за певними ознаками.

# Розділ 1. Огляд електронних словників та інструментів для їх побудови

## 1.1. Поняття електронного словника

Електронний словник — словник в комп'ютері або іншому електронному пристрої. Дані словники дозволяють швидко знайти потрібне слово, часто з урахуванням морфології і можливістю пошуку словосполучень (прикладів вживання), а так само з можливістю перекладу його на інші мови.

Деякі з таких словників забезпечують звуковий супровід перекладених слів. Серед найпопулярніших програм даного класу слід відзначити Context 3.51, ABBYY Lingvo, Pragma, Magic Translator та інші.

В електронному вигляді є і тлумачні словники української мови:

* E-slovnik;
* Словопедія;
* УКРЛІТ.ORG.

### 1.1.1. Сучасна лексикографія: можливості електронних словників

В сучасному значенні лексикографія — це теорія і практика складання словників, головним чином мовних, лінгвістичних, на відміну від енциклопедичних. Вона носить комплексний характер, але визначальною рисою лексикографії є ​​її прикладна спрямованість. Вчені стверджують, що теоретична лексикографія вивчає питання розробки макроструктури і мікроструктури словника, розробку типології словників, історія розвитку лексикографії. Макроструктуру словника визначає набір лексики, обсяг і характер словника, принципи розташування матеріалу.

Практичну лексикографію можна визначити як процес складання словників різних типів на базі теоретичних розробок. На думку більшості фахівців, із пришвидшенням темпів технічного розвитку суспільства змінюється характер сучасної лексикографії в цілому. На сьогоднішній день застосування комп'ютера для лексикографічних робіт набуває дедалі більшого поширення як в наукових дослідженнях, так і в практичній побудові словників. Обсяг лексикографічної праці, виконуваної за допомогою комп'ютера, сьогодні настільки великий, що багато фахівців говорять про появу відповідного розділу дисципліни — комп'ютерної лексикографії. Вона представляє:

* галузь комп'ютерної індустрії зі швидким ростом;
* прикладну наукову дисципліну в мовознавстві, що вивчає методи, технології і окремі прийоми використання комп'ютерної техніки в теорії і практиці складання словників.

Інші фахівці визначають лексикографію як сукупність словників, доступних в електронній формі, незалежно від способу їх створення. Електронна версія словника, покликана дати можливість швидко отримувати інформацію. Вона може міститься десь у глибинах словника і має безпосередньо відповідати тому запиту, який сформульований користувачем в зручній для нього формі.

Також лексикографія є особливим напрямком у практичній лексикографії зі своїми власними підходами не тільки до відображення, але і до вмісту словника. А. В. Шляхова в роботі «Електронний словник» надає характеристику цього типу видання. Основними є такі виділені нею ознаки:

* оборотність (зміна вхідної і вихідної мови);
* гнучкість (комплекс прийомів, що спрощує форму запиту для пошуку; зручність і легкість, з якою користувач може звертатися до комп'ютера, входити в словник і за короткий час отримати необхідну інформацію);
* динамічність (поповнення, корекція, вилучення застарілих даних);
* множинність (багатомовність);
* об'ємність (можливість включення будь-якої кількості словникових одиниць).

До додаткових можливостей можна віднести використання мультимедійних засобів для подання різного роду інформації.

Підсумовуючи вищесказане, можна стверджувати, що однією з головних переваг електронних версій словників є те, що вони дозволяють вносити нові слова в уже доступний користувачам словник, в той час як в друкований аналог добавити додаткову інформацію без перевидання неможливо. Але є й інша причина, по якій створення електронних словників є необхідним: автоматизація пошуку інформації. У сучасному світі, де швидкість знаходження інформації, швидкість життя і необхідність в швидкому отримання результатів будь-який пошук принципово важливий, це особливо актуально. Проте використання електронних словників не є надзвичайно поширеною практикою по ряду причин. З метою визначення цих причин проведено функціональне порівняння електронних та традиційних словників, з’ясовані переваги і недоліки видів словників. Більшість електронних словників виконує п'ять функцій:

* паралельний запит під час роботи;
* зміна і внесення виправлень (копіювати - вставити);
* замітки і примітки;
* експорт;
* друк.

Електронна форма надає виданню величезні можливості ще і по впливу на читача.

Ці можливості набагато ширші, ніж у друкованого аналога, і вони обумовлені двома факторами:

* у електронне видання може входити не тільки текст і ілюстрації, але також анімація, відео та звук;
* електронне видання володіє інтерактивністю.

Проаналізувавши характеристики електронних і традиційних видань, було виявлено наступні відмінності їх форм:

* для роботи з електронною версією потрібно комп'ютер і оперативна пам'ять пристрою, а для роботи з паперовою — книга і певний навик використання;
* у створенні традиційної версії словника потрібно витрачатись на поліграфічні матеріали та послуги друкарні, а для електронного словника — навички програмування, комп'ютер, що зчитує дані. Таким чином, можна говорити про екологічність виробництва електронних словників;
* електронний словник може містити максимально повну інформацію і довідкові матеріали, традиційний словник може мати обмежений обсяг, тому при його створенні необхідно обмежувати кількість даних;
* робота з паперовими словниками відбувається лінійним чином, в той час як з електронними словниками можна працювати як фрагментарним (пошук слова тільки за назвою) так і лінійним чином;
* електронні словники надають ширші в порівнянні з традиційними словниками можливості по оформленню видання: використання кольорів, розбиття на сторінки, візуалізація різних типів інформації; з іншого боку, в більшості паперових словників стаття не має кольорового оформлення, характеристики мінімальні і представлення інформації фіксоване;
* електронні словники надають користувачеві можливість легкого і швидкого пошуку; дані в них структуровані таким чином, щоб користувач міг здійснювати різні типи пошуку;
* електронний словник може мати постійну систему оновлення інформації, традиційні словники такою можливістю не володіють;
* до складу електронного словника можна включити замітки та коментарі. Крім того, його можна забезпечити перехресними посиланнями на інші словники або довідкові видання;
* електронні словники займають менше місця (на столі і в бібліотеці) і більш зручні в роботі, ніж паперові;
* електронний словник не схильний до старіння, його сторінки не мнуться, не рвуться і т. п.

З усього вищесказаного випливає, що два типи словника відрізняються базою подання інформації, що обумовлює переваги і недоліки тих і інших. Незважаючи на деякі мінуси, властиві електронним словникам, особливо в технічному плані (нестабільність в роботі жорстких дисків, порушення роботи програмного забезпечення, що може призвести в тому числі і до поломки персонального комп’ютера), вони мають великі переваги для користувача.

Створення електронних словників — це перспективний напрямок сучасної лексикографії. При цьому не можна казати про існування чітких правил або методів створення таких словників, єдиного стандарту побудови, єдиних критеріїв, по котрим визначається якість видань цього виду.

### 1.1.2. Особливості представлення лексикографічного матеріалу в цифровому середовищі

Дана АСУ підходить для вищих навчальних закладів різного рівня акредитації. Використовується більше ніж 70-тьма ВНЗ, відповідає усім вимогам ISO [3].

Однією з переваг системи є повна взаємодія з програмами ЄДЕБО, УЦОЯО, ІВС «Освіта».

Складається з таких основних автоматизованих систем:

* Приймальна комісія
* Деканат
* Студмістечко

Також доступна підсистема для тестування студентів. АС студмістечко має вбудовану систему електронних платежів, для оплати навчання, проживання тощо.

Кожен навчальний заклад, який використовує АСУ "ВНЗ" може отримати Веб-розклад, а його студенти та викладачі завантажити безкоштовний мобільний додаток "АСУ ВНЗ. Розклад".

АС «Деканат» – це програмно-технологічний комплекс управління навчальним процесом закладу освіти, призначений для організації роботи методистів та зменшення кількості документації на паперових носіях.

В процесі роботи АС «Деканат», до персональних даних студента додаються інші поля, що відображають поточний стан навчання студента згідно навчального плану. Тому доцільно використовувати дану програму у комплексі з АС «Приймальна комісія».

Дані зарахованих студентів пакетами переносяться з АС «Приймальна комісія». Це позбавляє працівників деканатів від необхідності вводити руками дані студентів. Таким чином, персональна картка для кожного студента вводиться лише один раз – під час подання документів до приймальної комісії, далі картка лише набуває додаткових даних.

Складається АС «Деканат» з таких модулів:

* Студенти
* Навчальні плани
* Сесії
* Журнал
* Контракти
* Накази
* Розклад
* Результати сесії
* Відділ кадрів

Загалом система є платною, та за певну ціну університет отримує увесь набір необхідних підсистем.

## 1.2 Алгоритми опрацювання тексту

За способом комунікації лінгвістичні алгоритми можна розділити на два види: алгоритми аналізу письмового тексту та усного мовлення. Алгоритми аналізу тексту розробляються з 50-х рр. XX ст. і лежать в основі функціонування інформаційно пошукових систем, систем автоматичного реферування [Яцко, 2007]. Алгоритми аналізу усного мовлення стали розроблятися і широко впроваджуватися в 90-х рр. XX ст. і зараз широко застосовуються в автовідповідачах [Experiences with commercial, 2004]; системах розпізнавання таких індивідуальних характеристик особистості, як вік, стать [Age and gender recognition, 2010] і навіть рівень алкогольного сп'яніння [Use of prosodic speech, 2001]; в системах голосового управління технічними об'єктами [Потапова, 1997], в тому числі і наносистемами [Потапова, 2007].

За формою мови можна виділити алгоритми, призначені для обробки монологічного та діалогічного мовлення. Довгий час об'єктом автоматичного аналізу тексту були монологічні тексти, в основному тексти наукових робіт. Розвиток інтернету зумовило появу жанрів діалогічного мовлення: чатів, блогів, форумів. Обробка таких текстів має свою специфіку і вимагає застосування спеціальних алгоритмів, що враховують їх паралінгвістичні особливості [Чухарев, 2008, с. 76-78].

За ступенем інтелектуальності можна виділити в окрему групу алгоритми, за допомогою яких користувачеві видається інформація, що міститься в тексті імпліцитно, або нова інформація, якої немає в оброблюваному тексті, наприклад, коефіцієнти, що відображають інтенсивність якої-небудь події. Такі алгоритми розробляються в процесі інтелектуального аналізу тексту (text mining) і істотно відрізняються від традиційних алгоритмів інформаційного пошуку і реферування, в результаті застосування яких виявляється найбільш значима інформація, що міститься в тексті.

Алгоритми автоматичної обробки тексту можуть застосовуватися на різних рівнях мовної системи, починаючи від окремого символу, який виступає об'єктом аналізу в оптичних системах розпізнавання тексту (optical character recognition - OCR) [OCR Systems, 1994], закінчуючи дискурсивні рівнем, на якому відбувається моделювання структури зв'язного тексту [Marcu, 1999].

### 1.2.1. Алгоритми морфологічного аналізу

За допомогою алгоритмів морфологічного аналізу розпізнаються елементи морфологічної структури слова - коріння, основа, афікси, закінчення. До алгоритмів, широко застосовуваним на морфологічному рівні, відносяться стемінг і лематизації.

Мета стемінгу - ототожнити основи семантично схожих словоформ, що необхідно для адекватного зважування термінів в процесі інформаційного пошуку. На вході стеммера - текст, на виході - список основ слів вхідного тексту. Стеммери, що розробляються з кінця 50-х рр. XX ст., класифікуються на алгоритмічні та словникові [Hull, 1996]. Алгоритмічні стеммери функціонують на основі файлів даних, що містять списки дериваційних суфіксів і флексій. У процесі морфологічного аналізу програма виконує зіставлення суфіксів і закінчень слів у вхідному тексті і в відповідному списку, причому аналіз починається з останнього символу слова. Словникові стеммери функціонують на основі словників основ слів. У процесі морфологічного аналізу такої стеммер виконує зіставлення основ слів у вхідному тексті і в відповідному словнику, а аналіз починається з першого символу слова.

Словникові стеммери забезпечують більшу точність пошуку, в той час як алгоритмічні - більшу повноту, допускаючи більше помилок, які проявляються в недостатньому або надмірному стеммуванні. Надмірне стеммування (overstemming) має місце в тому випадку, якщо по одній основі ототожнюються слова з різною семантикою; при недостатньому стеммуванні (understemming) по одній основі не зіставляються слова з однаковою семантикою. Наприклад, ланкастерский стеммер виділяє bet як основу better, a childr - як основу children. У першому випадку має місце надлишкове стемування, оскільки по основі bet прикметник better ототожнюється з дієсловом bet і його похідними {bets, betting), значення яких не має нічого спільного зі значенням прикметника. У другому випадку має місце недостатнє стеммування, так як по основі childr не можна ототожнити форми множинного {children) і однини {child) однієї лексеми.

Незважаючи на зазначені недоліки, алгоритмічні стеммери набагато більш поширені, ніж словникові. Це пояснюється тим, що кількість суфіксів та флексій в кожній конкретній мові досить обмежена, отже, зміни на рівні морфологічної структури відбуваються набагато повільніше, ніж на лексичному рівні. Стрімкий соціальний і технологічний розвиток обумовлює вихід з обігу одних слів і поява інших. В першу чергу це відноситься до іменників, які створюються для позначення нових об'єктів. У Британському національному корпусі, наприклад, немає таких термінів, як ipod або iphone, оскільки він охоплює тексти, вироблені з 1980-х рр. по 1993 р Іншою проблемою при використанні словникових стеммерів є великий розмір словника, який негативно впливає на швидкодію системи.

В даний час найбільш відомими стеммерами для англійської мови є алгоритмічні стеммери: стеммер Портера і ланкастерский стеммер, який, за прізвищами розробників (CD Paice і G. Husk), також називають Paice / Husk stemmer [Paice, 1990]. М. Портер розробив мову програмування Snowball спеціально призначений для створення Стеммер [Porter, 2001]. В Росії був розроблений Y-stemmer, який виконує морфологічний аналіз на основі попереднього анотування тегами частин мови. Це дозволяє враховувати тільки суфікси і закінчення, співвідносні з тією частиною мови, до якої відноситься дане слово. Крім того, в Y-Стеммер передбачено ототожнення неправильних форм дієслів, а також іменників і займенників, що утворюють множину не за правилами [Алгоритмы предварительной обработки текста 2009].

Ефективність функціонування програм морфологічного аналізу визначається поняттям потужності (strength) стеммера, яка вимірюється по співвідношенню кількості словоформ вихідного тексту і основ слів, що залишилися після стемінгу, а також за кількістю символів, які містяться в видалений суфіксах і закінченнях. Наприклад, потужність Y-стеммеру, протестована на відомому тексті The Ransom of Red Chief за формулою компресії індексу, (NS) / N = (1203-1083) / 1203 = 0,09975, де N - кількість унікальних словоформ в початковому тексті, a S - кількість унікальних основ після стемінгу. Потужність стеммеру Портера для того ж тексту дорівнює (1203-1080) / 1203 = 0,10224, тобто стеммер Портера потужніший Y-Стеммер на 0,00249. Дані для ланкастерського стеммеру: (1203-1081) / 1203 = = 0,10141. Зауважимо, що немає прямої залежності між потужністю стеммеру і його якістю. Проведене тестування показало, що якість Y-Стеммер на 9,83% вище, ніж якість ланкастерського Стеммер [Алгоритмы предварительной обработки текста, 2009].

Лематизація також передбачає ототожнення основ слів, однак проводиться з урахуванням частин мови, до яких відносяться словоформи. Наприклад, стеммер ототожнить read, reads, reader, readers з однією основою read, в той час як лематайзер ототожнить дієслівні форми read, reads з основою read, а іменні форми reader, readers - з лексемою reader. Під лемою розуміється лексема, завдання лематизації - ототожнити словоформи, співвідносні з однією лексемою.

З метою автоматичної обробки англійських текстів широко застосовуються лематизовані і нелематизовані списки, складені А. Кілгаріфом на основі Британського національного корпусу. Дані списки - це словники, в яких дається інформація про частотності і розподіленні лем і словоформ за текстами і частин мови. Ці дані можуть використовуватися для визначення імовірнісних характеристик лексичних одиниць, необхідних, наприклад, для автоматичного анотування тегами частин мови.

Словники лем широко використовуються в корпусних лінгвістиці з метою підтримки лінгвістичних досліджень. Розподіл за частинами мови також є істотним параметром при проведенні автоматичної класифікації і категоризації текстів [Santini, 2006].

Програмне забезпечення ПП "Політек-СОФТ" призначене для вищих навчальних закладів України від I до IV рівнів акредитації, охоплює майже всі аспекти їх діяльності, зареєстровано в Реєстрі виробників та розповсюджувачів програмного забезпечення, має інформаційні листи від Міністерства освіти і науки, молоді та спорту України щодо використання в вищих навчальних закладах України [4].

Складається з таких пакетів програм:

* Деканат
* Абітурієнт
* Колоквіум
* Бібліограф
* Персонал

Пакет програм "Деканат"

Деканат - пакет програм, що призначений для автоматизації планування та обліку навчального процесу.

Основні можливості Пакету:

* формування даних щодо структури навчального процесу;
* формування даних щодо всіх викладачів та їх планового навантаження, розклад їх роботи;
* формування даних щодо щоденних даних про фактичну роботу кожного викладача по кожній дисципліні;
* формування великого обсягу даних щодо всіх студентів та їх успішності за весь період навчання;
* формування даних щодо наявності корпусів та аудиторій, їх заповнення, розклад занять.

Особливості Пакету:

* великий обсяг та повнота інформації, яка зберігається в базі даних;
* великий обсяг звітів, які можна підготувати на основі даних з бази даних з урахуванням вимог Міністерства освіти і науки, молоді та спорту України;
* інформаційна сумісність з іншими продуктами ПП "Політек-СОФТ".

Пакет програм "ПС-Абітурієнт

ПС-Абітурієнт - пакет програм, що призначений для автоматизації діяльності приймальної комісії.

Основні можливості Пакету:

* формування даних щодо підрозділів;
* формування даних щодо галузей знань, напрямів підготовки, спеціальностей (спеціалізацій) та їх розподілу за підрозділами;
* формування даних щодо переліку дисциплін, з яких приймаються сертифікати зовнішнього незалежного тестування або складаються вступні іспити;
* формування даних щодо кожного абітурієнта;
* формування даних щодо балів сертифікатів абітурієнтів;
* формування даних, що необхідні для щоденної передачі до системи "Конкурс".

Особливості Пакету:

* великий обсяг та повнота інформації, яка зберігається в базі даних;
* великий обсяг звітів, які можна підготувати на основі даних з бази даних;
* інформаційна сумісність з іншими продуктами ПП "Політек-СОФТ".

Пакет програм "Колоквіум"

Колоквіум - пакет програм, що призначений для автоматизації тестування студентів.

* зручна оболонка для формування тестів;
* тестування студентів в різних режимах та на основі результатів тестування можливість оцінювати їх знання;
* контроль процесу тестування в реальному часі;
* забезпечення доступу до бази даних з результатами тестування з метою їх всебічного аналізу;
* генерація звітів за результатами тестування.

Особливості Пакету:

* інтуїтивно зрозумілий інтерфейс;
* великий обсяг інформації, яка зберігається в базі даних;
* велика кількість режимів тестування і оцінювання, що дає можливість підібрати оптимальні параметри тестування для кожного конкретного випадку залежно від вимог викладача та важливості тесту;
* можливість використання в будь-яких навчальних закладах;
* інформаційна сумісність з пакетом програм "Деканат".

Пакет програм "Бібліограф"

Бібліограф - пакет програм, що призначений для для автоматизації діяльності бібліотеки.

### 1.2.1. Алгоритми лексичного аналізу

Основне завдання лексичного аналізу - розпізнати лексичні одиниці тексту. На вході у програм цього типу - текст, на виході - список лексичних одиниць тексту. Одним з фундаментальних алгоритмів лексичного аналізу є лексична декомпозиція, яка передбачає розбивку тексту на токени; відповідно, програми, виконують лексичну декомпозицію, називаються токенайзерами. Як правило, токени збігаються зі словоформами, однак для позначення лексичних одиниць тексту використовується термін «токен», а не «слово», так в ряді випадків під токеном можуть розумітися одиниці менше, ніж слово або більше, ніж слово (словосполучення).

Токенайзери, розроблені для англійської мови, виконують декомпозицію на основі пробілів між словами і зазвичай розпізнають в якості окремих токенов апостроф і йдуть за ним символи (, 'v, 7 /,' d, 'т, \ е,' ге) \ від слів відокремлюються і видаляються знаки пунктуації. Проблема розпізнавання словосполучень і абревіатур вирішується застосуванням регулярних виразів. Очевидно, що скорочення типу eg слід розпізнавати як один токен; те ж саме відноситься до дат, наприклад 11.11.2111. Окрему проблему представляють ініціали та скорочення перед особистими іменами, наприклад, J. Smith, Dr. Smith, JB Smith. Якщо дані імена є кореферентними, то є менс розглядати ініціали як окремі токени: це дозволить розпізнати Smith як ім'я одного з персонажів і нарахувати адекватні вагові коефіцієнти залежно від його частотності. Якщо ж маються на увазі різні люди, слід розглядати прізвище та ініціали як один токен. Зазвичай лексична декомпозиція проводиться на основі списків скорочень. Крім того, в окремому файлі збираються стійкі словосполучення та ідіоми, які розпізнаються як один токен. Наприклад, because of доцільно розглядати як один токен, оскільки це союзне словосполучення виражає одне значення.

Лексична декомпозиція має фундаментальне значення для проведення автоматичного аналізу тексту, оскільки лежить в основі цілого ряду інших алгоритмів. Очевидно, що для проведення стемінгу слід спочатку розбити текст на токени; на основі списку токенов зазвичай виконується синтаксична декомпозиція, зважування, нарешті, анотування, також виконується на лексичному рівні.

Анотація проводиться тегерами, на вході у яких - список токенов, на виході - список, в якому кожному токені присвоюється умовне позначення (тег), що вказує на його лінгвістичні характеристики. Найбільш поширеним видом тегер є тегер частин мови (POS taggers), які розпізнають частина мови токена і приписують йому відповідний тег. Крім інформації про частини мови зазвичай вказується і інформація про лексико-граматичні і семантичні характеристики слова, наприклад, NN - загальний іменник в однині, NNS - загальний іменник у множині, AJC - прикметник в зі ступенем порівняння. Списки тегів частин мови розрізняються за ступенем дробності. У Британському національному корпусі використовується 58 тегів, а в проекті Penn Treebank – 36. Більш подрібнена класифікація дозволяє видавати користувачу більше інформації, однак обумовлює і більшу кількість помилок, знижуючи швидкодію програми.

Тегер частин мови послідовно виконують три основні операції: токенізацію, морфологічну класифікацію та зняття неоднозначності (disambiguation). Морфологічна класифікація передбачає зіставлення кожного токена вхідного тексту зі словником і приписування йому тегів частин мови. У словнику зазвичай містяться словоформи з можливими тегами частин мови. Досить велика кількість слів співвідноситься тільки з однією частиною мови (прийменники, артиклі, займенники), проте цілий ряд слів може використовуватися в якості різних частин мови. Для англійської мови типова омонімія дієслівних і іменних форм. Control може використовуватися і як дієслово, і як іменник, причому за даними нелематизованого списку словоформ Британського національного корпусу як іменник в однині (NN1) ця словоформа використовується 18 932 раз в 2 318 текстах, як інфінітив (VVI) -3 578 раз в 1 390 текстах, як особиста форма (VVB, без обліку використання в третій особі однини) - 957 раз в 559 текстах. До вказувальних відноситься можливість вживання цієї словоформи як прикметник - 1 випадок в 1 тексті. Дана статистична інформація важлива для подальшої обробки на стадії зняття неоднозначності.

Якщо будь-яке слово з вхідного тексту відсутнє в словнику, застосовуються спеціальні правила для розпізнавання частини мови, до якої воно відноситься. Якщо, наприклад, слово закінчується на -ious, то йому приписується тег прикметника, оскільки таке закінчення типове для англійських прикметників; словами, які починаються з великої літери, приписується тег власного імені. У тому випадку, якщо неможливо застосувати правила, токену приписується тег, який використовується за умовчанням, зазвичай - тег іменника. Іменники - найбільш частотна частина мови знаменних слів, і саме вони позначають нові об'єкти, імена яких можуть бути відсутніми в словнику. Якщо всі слова в тексті приписати теги іменників, то можна правильно проанотувати 14,6% слів [NLTK, 2011].

Токени, яким приписано більше одного тега, а також статистична інформація про них передаються для подальшої обробки в модуль зняття неоднозначності. Зняття неоднозначності передбачає вибір одного з двох або більше тегів, приписаних даному токені. Залежно від алгоритмів, що застосовуються для зняття неоднозначності, тегер частин мови класифікуються на стохастичні і засновані на правилах (rule-based). В стохастичних тегерах проводиться аналіз імовірнісних параметрів кожного з тегів (зазвичай на основі прихованих марківських моделей), в результаті якого вибирається один тег з найбільшим імовірнісним значенням. Поширений алгоритм двобічної інференції, виконання якого передбачає аналіз тегів токенів справа і зліва від поточного токену.

У тегерах, заснованих на правилах, аналіз імовірнісних характеристик не проводиться, хоча враховуються частотності використання тегів з тим чи іншим токеном. Такий тегер навчається на досить великому анотованому корпусі, запам'ятовуючи найбільш частотні теги морфологічно омонімічних словоформ, далі, для підвищення якості аннотування застосовуються спеціальні правила автоматичного виправлення помилок. Як приклад можна привести тегер, розроблений Еріком Брілом, який навчався на Браунівському корпусі, що містить понад мільйон словоформ. При налаштуванні тегер використовувалося три групи правил: правила, що враховують лексичні параметри поточного токена; правила, що враховують контекст токена; правила, що враховують відстань від поточного токена до іншого токена з певним лексичним параметром. Навчання тегеру проводилося на 90% текстів корпусу; 5% використовувалися для тестування і розпізнавання помилок: анотація тегеру порівнювалася з анотацією корпусу; на що залишилися 5% оцінювалася ефективність правил. Без застосування правил тегер допускав 7,9% помилок, а після застосування і доопрацювання правил кількість помилок знизилося до 3,5%. У Британському національному корпусі кількість помилок становить 1,5%, причому при анотування використовується гібридна технологія з використанням стохастичного тегеру і модифікуючих правил. Цікаво, що в даному корпусі застосовується технологія бінарних тегів: якщо для токену не вдалося знайти один тег, то йому приписується два тега, перший з яких є найбільш імовірним, наприклад, AJ0-AV0 вказує на те, що більш імовірним є тег прикметника. Остаточний вибір тега залишається на розсуд користувача.

В цілому стохастичні і гібридні технології істотно знижують кількість помилок, однак негативно впливають на швидкодію системи. Їх можна успішно використовувати для анотування статичних корпусів. Для динамічного анотування краще застосовувати тегери, засновані на правилах, оскільки вони забезпечують більшу швидкодію. Перспективним напрямком розвитку динамічного аннотирования є створення фактографічних пошукових систем. В даний час в таких системах використовується анотування такими семантичними тегами, як Person, Location, Organization.

InFact, одна з таких систем, розроблена в компанії Insightful Corporation, дозволяє отримувати інформацію за запитами. У відповідь на такий запит користувачеві буде видано клаузи тексту, в яких міститься інформація про покупку однією компанією іншій компанії за певну суму грошей. Для анотування семантичними тегами використовується програмне забезпечення Talent, розроблене фірмою IBM.

Ще одним напрямком є ​​анотування тегами когнітивних ролей (knowledge roles), яке застосовується в інтелектуальному аналізі тексту (text mining). В [Mustafaraj, 2007] проводилося анотування текстів діагностичних звітів про стан електроізоляції високовольтних ротаційних пристроїв такими ролями, як Observed Object, Symptom, Cause. В результаті була створена система, за допомогою якої інженер міг отримувати інформацію про ознаки неполадки конкретного об'єкта, причини та способи її усунення. Як лінгвістичної бази даних використовувалася лексикографічна інформація, розроблена в рамках проекту Framenet.

Слід зазначити, що анотування семантичними і когнітивними ролями передбачає розпізнавання як окремих слів, так і словосполучень. Таке анотування вимагає попередньої розробки і застосування спеціальних граматик фразової структури на синтаксичному рівні мовної системи.

### 1.2.2. Алгоритми синтаксичного і дискурсивного аналізу

Одним з фундаментальних алгоритмів, застосовуваних на синтаксичному рівні, є синтаксична декомпозиція (syntactic splitting). На вході у сплітера - текст, на виході - список речень тексту. Алгоритми синтаксичного декомпозиції розробляються з 1960-х рр. і передбачають розпізнавання речень на основі символів форматування тексту: прогалин, знаків пунктуації, знаків перекладу каретки. Розбивка тексту на речення ускладнюється відсутністю стандартного форматування тексту; точки, знаки оклику, знаки питання, які зазвичай застосовуються в якості роздільників, можуть використовуватися не тільки в кінці, але і в середині речення. Цілий ряд одиниць тексту, які форматуються як реченнями, насправді реченнями не є. До них відносяться такі елементи, як зміст, заголовки окремих розділів, назви малюнків, таблиць, текст, який використовується всередині самих таблиць і малюнків, колонтитули. Тим часом саме речення є основною одиницею аналізу в багатьох системах, а в системах автоматичного реферування і вихідний текст складається з пропозицій. Помилки в розпізнаванні пропозицій істотно знижують ефективність таких систем в цілому.

Була запропонована дедукційно-інверсійна архітектура декомпозиції тексту, відповідно до якої спочатку текст розбивається на абзаци, потім - на слова, потім з слів генеруються речення. Таким чином, декомпозиція починається з більшої одиниці (абзацу), потім здійснюється перехід до меншої одиниці (слова), потім - знову до більшої (речення). Дедукційно-інверсійна архітектура декомпозиції дозволяє ігнорувати такі компоненти тексту, як заголовки, підзаголовки, змісту, оскільки вони не входять до складу абзаців.

Синтаксична декомпозиція є основою для виконання цілого ряду алгоритмів розпізнавання фразової структури речень. Широко поширені алгоритми виділення n-gram - словосполучень, що складаються з двох (біграми), трьох (тріграми) і більше (тетраграми, пентаграми, гексаграми, гептаграми, октограми) токенів. Розбиття на словосполучення в даному випадку проводиться з урахуванням позиції токену в реченні. Наприклад, речення “John has a dog” включає 4 юніграма, 3 ді-грама (John has, has a, a dog), 2 тріграма (John has a, has a dog), 1 тетраграм - все речення. Кількість біграм для кожної пропозиції (ng (s)) становитиме n-1, тригарм - n-2, де n-кількість токенов в реченні, тобто ng (s) = Wi\_ (n\_1}, wKn\_2) ... wKn\_n), де w - порядковий рівень n-gram, починаючи з біграм. Розпізнавання n-gram проводиться на основі відповідних правил.

Аналіз розподілу n-gram дозволяє виявити статистично значущі словосполучення і часто застосовується в стохастичних алгоритмах анотування тегами частин мови. При цьому початок і кінець речення позначаються деякими умовними тегами (false tags), що дозволяє розглядати в якості тріграм навіть речення що складаються з одного токена і встановлювати імовірнісні параметри, необхідні для вибору того чи іншого тега. У корпусі сучасного американського варіанту англійської мови (СOCA) “I like to” зустрічається 4 810 раз, в той час як “I like women” - 29 разів, що вказує на набагато більшу ймовірність першого словосполучення. Розподіл n-gram використовуються з метою автоматичної класифікації і категоризації, оскільки виступають в якості важливого параметра, що дозволяє визначити приналежність тексту до певної категорії, типу, групи, жанру. При аналізі на синтаксичному рівні в якості основної одиниці виступають біграми і діграмми, оскільки рекурентність словосполучень з великою кількістю токенів малоймовірна. Аналіз n-gram більшого порядку застосовується в системах автоматичної корекції орфографії, а також в системах автоматичного розпізнавання текстів (Optical Character Recognition), де основною одиницею виступають символи в токенах.

Для аналізу морфологічно значущих словосполучень застосовуються чанкери (chunkers), які на виході видають списки фраз певного типу (іменні, дієслівні, ад'єктивних, адвербіальні). Найбільш поширені іменні (noun phrase) чанкери, що розпізнають словосполучення з керуючим іменником. Саме цим типом словосполучень позначаються об'єкти, що описуються в тексті, а їх рангування за ваговими коефіцієнтами дозволяє отримати список ключових слів, які відображають основний зміст тексту. Як було показано раніше, реферування тексту на основі словника іменників дозволяє отримати практично такі ж результати, як і реферування, що проводиться і з урахуванням слів, що відносяться до інших частин мови.

Правила фразової структури були розроблені для англійської мови в рамках концепції генеративної граматики, запропоновано Н. Хомським. Граматичні правила записуються у вигляді NP -> NN; NP -> DetNN; NP -> DetANN, де вказується склад словосполучення, в даному випадку іменного (noun phrase - NP), а також порядок слів. У першому випадку показано, що іменне словосполучення може складатися тільки з одного іменника (NN); у другому випадку воно складається з детермінанта (Det) і іменника, причому детермінант займає позицію перед іменником, а зворотний порядок слів неправильний; в третьому випадку словосполучення складається з детермінанта, прикметника (А), іменника, при цьому інші варіанти словопорядка неправильні.

До теперішнього часу на основі концепції Н. Хомського створений цілий ряд граматик, які діляться на два основних види - дериваційні та недериваційні. У дериваційних граматиках проводиться розмежування між поверхневою і глибинною структурою словосполучення і пропозиції і формулюються додаткові правила виводу (деривації) поверхневих структур та глибинних. Синтаксична структура представляється у вигляді ієрархічного дерева залежності. Недериваційні граматики описують поверхневі, як правило, лінійні синтаксичні структури. Вибір того чи іншого типу граматики обумовлюється завданнями конкретного дослідницького проекту.

Дериваційні граматики лежать в основі функціонування синтаксичних парсеров (syntactic parsers), які видають на виході граф синтаксичної структури речення. Так само, як і тегер частин мови, синтаксичні аналізатори навчаються на реченнях з розміченій вручну синтаксичною структурою; в них застосовуються правила для визначення найбільш ймовірного варіанту на основі прихованих моделей Маркова. Як приклад можна привести Lexparser, розроблений в Стенфордському університеті США.

Ієрархічні синтаксичні структури застосовуються в системах машинного перекладу для встановлення еквівалентності синтаксичних структур в двох мовах.

На синтаксичному рівні може проводитися декомпозиція не тільки на словосполучення і пропозиції, а й на клаузи - елементарні предикативні структури, що виражають судження. Поняття клауз певною мірою відповідає поняттю пропозиції в лінгвістиці, однак клауз виділяються за формальними ознаками, до яких може відноситися, наприклад, наявність іменний групи і наступної за нею дієслівної групи. Розбивка на клауз застосовується в системах інтелектуального аналізу для більш адекватної передачі змісту тексту, наприклад, описаний вище проект німецьких дослідників [Mustafaraj, 2007].

Найбільш поширеними алгоритмами, застосовуваними на дискурсивної рівні (рівні зв'язного тексту), є алгоритми вирішення анафори, які передбачають заміну анафорических займенників попередніми кореферентними іменами об'єктів.

До теперішнього часу склалося два основних підходи до розробки алгоритмів вирішення анафори: глобально-дискурсивний та статистичний. Глобально-дискурсивний підхід передбачає розпізнавання антецедентів на основі моделювання тематичної структури тексту, в той час як статистичний підхід, в рамках якого виконується більшість досліджень, грунтується на приписуванні можливим антецедентам вагових коефіцієнтів. В узагальненому вигляді алгоритм вирішення анафори в рамках статистичного підходу включає три етапи:

Перегляд контексту і розпізнавання за певними критеріями можливих антецедентов для поточного займенники. В якості контексту можуть виступати словосполучення в даному реченні / клаузі, попередні речення / клаузи, а також наступні речення / клаузи, якщо текст починається з займенника.

Приписування вагових коефіцієнтів кожного з можливих антецедентів на основі певних параметрів і факторів. У цьому сенсі статистичні алгоритми вирішення анафори можна віднести до методів факторного аналізу.

Вибір в якості кореферентного антецедента з найбільшою вагою.

В [Lappin, 1994] описується алгоритм, розроблений для аналізу технічних текстів. Алгоритм ґрунтується на чотирьох групах правил: правила фільтрації іменних словосполучень, які не можуть бути антецедентом займенники, правила розпізнавання вагових параметрів можливих іменних словосполучень-антецедентов, правила рангування можливих антецедентів за ваговими параметрами, правила вибору найбільш ймовірного з можливих антецедентів.

Фільтрація проводиться по ряду лексикограматичних і синтаксичних параметрів. У реченні “The woman said that he is funny” іменна група “The woman” не може бути антецедентом займенника he оскільки вони не узгоджуються в роді і використовуються з дієсловами в різній тимчасовій формі. В “John seems to want to see him” власне ім'я John не може бути антецедентом займенника him оскільки воно входить до складу обов'язкового аргументу дієслова, що займає позицію безпосередньо після власного імені. Для окремих видів займенників розробляються власні правила фільтрації.

Після проведення фільтрації ідентифікуються іменні групи, які можуть бути можливими антецедентами займенника в поточному реченні. Якщо можливих антецедентів більше одного, то для кожного з них визначаються параметри, за якими проводиться їх зважування. Антецедент з великою вагою вибирається в якості кореферентносго терміна. До параметрів, від яких залежить ваговий коефіцієнт іменної групи, відносяться: використання в суб'єктній позиції, позиції прямого доповнення, непрямого доповнення, прийменникового доповнення, використання в не-адвербіальний позиції, використання в якості керуючого іменника. Набольший коефіцієнт отримують унікальні (що не повторюються) іменні групи. У разі повторного використання коефіцієнт унікальності і всі інші коефіцієнти знижуються в два рази.

(1) You have not waited for the file to close.

(2) You may have asked to print on the virtual printer, but it can not print until the output file is closed.

Після фільтрації некореферентних імен залишається два можливих антецедента займенники it: file в (1) і printer в (2). Оскільки printer зустрічається в тексті 1 раз, то йому присвоюється коефіцієнт унікальності 100, а терміну file, що зустрічається 2 рази - коефіцієнт 50, причому коефіцієнти для всіх параметрів file також знижуються в два рази. Обидва іменників використовуються в неадвербальній позиції, тому printer нараховується додатковий коефіцієнт 50, a file - коефіцієнт 25. Обидва терміни є керуючими іменниками, за цим параметром printer нараховується коефіцієнт 80, а file - коефіцієнт 40. Крім того, printer нараховується коефіцієнт 40 за використання в позиції прийменниково доповнення, a file - коефіцієнт 40 - за використання в суб'єктній позиції і коефіцієнт 35 за використання в паралельних синтаксичних ролях в двох реченнях. Загальний коефіцієнт для printer = 210, для file = 90; відповідно в якості кореферента it вибирається printer.

Ефективність даного алгоритму була протестована на корпусі з 48 текстів інструкцій для користувачів комп'ютерів. З корпусу довільним чином були відібрані пропозиції, що містять хоча б одне займенник, а потім для кожного речення було виписано попереднє речення. В цілому у всіх реченнях виявилося 360 займенників, і для 310 з них були правильно обрані антецеденти, що дало якість в 86%.

Очевидно, що для виконання такого алгоритму необхідна досить складна граматика, за допомогою якої розпізнаються види фраз, синтаксичні ролі, лексікограматичні параметри. При цьому дозвіл анафори виконується з урахуванням не більше ніж 4 попередніх пропозицій, що істотно обмежує сферу застосування алгоритму.

Алгоритми дозволу анафори мають істотне значення для всіх напрямків автоматичного аналізу тексту. У системах автоматичного реферування та інформаційного пошуку заміна займенників кореферентними іменами дозволяє зробити більш адекватне зважування термінів і суттєво підвищити якість результату. Алгоритми дозволу анафори важливі і для розробки питально-відповідних систем, систем інтелектуального аналізу тексту, а також і для систем штучного інтелекту, оскільки дозвіл анафори передбачає формулювання логічного висновку (inference) [Barland 2009].

Дана стаття написана на основі досвіду розробки програм автоматичного аналізу англійських текстів в лабораторії комп'ютерної лінгвістики ХДУ ім. Н.Ф. Катанова і, зрозуміло, не претендує на їх вичерпний опис. Слід також мати на увазі, що запропонована класифікація алгоритмів в залежності від рівнів мовної системи є досить абстрактною. В реальності описані алгоритми можуть застосовуватися в різній послідовності. Наприклад, синтаксична декомпозиція може передувати лексичнії: спочатку розпізнаються речення, а потім виділятися токени, з яких вони складаються. У деяких випадках існує досить жорстка послідовність виконання алгоритмів, зокрема, анотування обов'язково передбачає попередню лексичну декомпозицію. Ряд однотипних алгоритмів може застосовуватись на різних рівнях мовної системи. Найбільш типовий приклад - зважування термінів: вагові коефіцієнти, що визначаються за результатами зважування, можуть підписуватися окремим словами, словосполученнями, пропозиціями, групам пропозицій, а також тексту в цілому.

Можлива класифікація алгоритмів автоматичного аналізу тексту і за іншими критеріями. Суттєвою характеристикою є виділення динамічних і статичних алгоритмів. Динамічні алгоритми виконуються «на льоту», у відповідь на запит користувача, в той час як статичні алгоритми виконуються в процесі попереднього аналізу тексту, до того, як до нього звертається користувач. Відповідно, істотно розрізняються вимоги до швидкодії, що в свою чергу впливає на вибір архітектури і мови використовуваного програмного забезпечення. Найбільша швидкодію досягається при застосуванні алгоритмів поверхневого рівня, до яких відносяться позиційно-статистичні алгоритми. Більш складні алгоритми семантичного рівня, що передбачають аналіз семантики мовних одиниць (наприклад, структурно-семантичних відносин), або аналіз структури зв'язного тексту, в тому числі моделювання його тематичної структури. Таким чином, можна виділити три групи алгоритмів: алгоритми поверхневого, семантико-синтаксичного і дискурсивного рівнів. Для підтримки цих алгоритмів використовуються і різні лексикографічні ресурси. Алгоритми поверхневого рівня виконуються на основі словників, що містять статистично-імовірнісні дані про розподіл мовних одиниць. Для виконання алгоритмів семантичного рівня потрібні словники-тезауруси, семантичні словники, онтології. Перспективним напрямком, виступає розробка словників, в яких вказуються семантико-синтаксичного і дискурсивного рівнів. Для підтримки цих алгоритмів використовуються і різні лексикографічні ресурси. Алгоритми поверхневого рівня виконуються на основі словників, що містять статистично-імовірнісні дані про розподіл мовних одиниць. Для виконання алгоритмів семантичного рівня потрібні словники-тезауруси, семантичні словники, онтології. Перспективним напрямком, виступає розробка словників, в яких вказуються семантико-синтаксичного і дискурсивного рівнів. Для підтримки цих алгоритмів використовуються і різні лексикографічні ресурси. Алгоритми поверхневого рівня виконуються на основі словників, що містять статистично-імовірнісні дані про розподіл мовних одиниць. Для виконання алгоритмів семантичного рівня потрібні словники-тезауруси, семантичні словники, онтології. Перспективним напрямком також виступає розробка словників, в яких вказуються такі семантичні ознаки слів, що виділяються в компонентному аналізі, як «абстрактність - конкретність», завдяки яким можна істотно підвищити ефективність виконання, наприклад, дозволу анафори. Взагалі, актуальним є більш широке застосування і алгоритмізація таких лінгвістичних методів, як компонентний, предікаціонний, падежно-рольовий аналіз.

При розробці лінгвістичного програмного забезпечення використовуються найрізноманітніші мови програмування. Найбільш популярні мови групи С (C++, С#). У США широко застосовується мова Python, зокрема в інструментальному ПО NLTK (Natural Language Toolkit), розробленому в Пенсильванському університеті [Mertz, 2004].

Істотна проблема, з якою стикаються розробники сучасного лінгвістичного ПО - погана якість текстів, що розміщуються в Інтернеті. До таких текстів, як чати досить важко, а часто і неможливо, застосувати традиційні алгоритми аналізу в силу численних відхилень від норм орфографії, пунктуації та граматики. Разом з тим саме такі жанри текстів, як чати, блоги, форуми є цінним джерелом інформації і є об'єктом аналізу в цілому ряді областей, в першу чергу в програмах інтелектуального аналізу тексту. Розробка алгоритмів аналізу діалогічних текстів також є перспективним напрямком в рамках автоматичної обробки природної мови.

# Розділ 2. Постановка задачі розробки програмного продукту для формування лексико-семантичного словника української мови

## 2.1. Загальна постановка задачі

Реалізувати програмний продукт, за допомогою якого можна класифікувати слова за певними ознаками. Ознаки мають формувати певну ієрархічну структуру, яку можна буде легко модифікувати. Повинна бути можливість побудови нових семантичних структур. Програма повинна надавати можливість перегляду слів по ознаках, до яких вони належать, а також зберігати приклади використання слів у цитатах для кращої наочності приналежності до ознак. Повинен існувати окремий модуль, який зможе розподіляти слова по ознакам.

## 2.2. Вибір інструментальних засобів

Розроблення програмного продукту здійснюватиметься в середовищі розробки -Microsoft Visual Studio . VS- серія продуктів фірми Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Presentation Foundation, а також веб-сайти, веб-застосунки, веб-служби .

Мовою розробки було обрано C#. Версія платформи – .NET Framework 4.5, яка включає в себе усі використані технології. Технологія WPF (Windows Presentation Foundation) був обраний для побудови графічного інтерфейсу користувача.

Технологія WPF є частиною екосистеми платформи .NET і являє собою підсистему для побудови графічних інтерфейсів.

Якщо при створенні традиційних додатків на основі WinForms за відображення елементів управління і графіки відповідали такі частини ОС Windows, як User32 і GDI+, то додатки WPF засновані на DirectX. У цьому полягає ключова особливість рендерингу графіки в WPF: використовуючи WPF, значна частина роботи по відображенні графіки, як найпростіших кнопок, так і складних 3D-моделей, лягає на графічний процесор на відеокарті, що також дозволяє скористатися апаратним прискоренням графіки.

Однією з важливих особливостей є використання мови декларативною розмітки інтерфейсу XAML, заснованої на XML: можна створювати насичений графічний інтерфейс, використовуючи або декларативне оголошення інтерфейсу, або код на керованих мовах C# і VB.NET, або поєднувати і те, і інше.

Для з'єднання бази даних до проекту використовувався MS SQL Server. MS SQL Server - це надійна , захищена та стабільна платформу для роботи програмного забезпечення, важливого для бізнесу, яка ідеально підходить для впровадження в додатки для мобільних пристроїв і настільних комп'ютерів.

## 2.3. Специфікація вимог до програмного продукту

### 2.3.1. Вступ

#### 2.3.1.1. Призначення, мета

Призначення даного продукту – відображення зв’язків між ознаками та приналежність слів до певних ознак.

Мета – створити семантико-стилістичний електронний словник української мови

#### 2.3.1.2. Продукти аналоги

Наразі аналогів даному продукту на українському ринку просто не існує. Є безліч електронних тлумачних словників української мови, однак в них слова не класифікуються по ознаках. Саме це дозволяє детально систематизувати семантику української мови.

### 2.3.2. Загальний опис

#### 2.3.2.1. Характеристики продукту

Функції, що будуть реалізовані у програмі:

* Добавлення нового слова до словника;
* Редагування та видалення слів;
* Добавлення до поточного слова певних ознак;
* Редагування ознак слова(видалення та зміна);
* Редагування дерева ознак;
* Пошук слів у словнику;
* Швидкий доступ до останніх доданих слів;
* Додавання нових слів із файлу;
* Автоматичне наповнення словника.

#### 2.3.2.2. Середовище функціонування

Апаратні вимоги:

* Частота процесора 1,5 GHz;
* Оперативна пам’ять 1024 Mb;
* Процесор Intel Pentium 4 / Athlon 64 або вищі версії.

Системні вимоги:

* Операційна система Microsoft Windows 7 або вищі версії ОС;

#### 2.3.2.3. Характеристики системи

##### 2.3.2.4.1. Добавлення нового слова до словника

Опис і пріоритет:

Можливість додавання нового слова в словник, з автоматичною перевіркою того, чи слово вже міститься у даному словнику.

Пріоритет – високий.

Послідовність дія\відгук:

Користувач натискає клавішу для додавання нового слова. З’являється вікно з можливістю створення групи слів. Можливість додати нове слово має бути надана практично з будь-якого основного вікна програма, оскільки це є ключовою функцією словника.

Функціональні вимоги:

REQ-1.1: Користувач натискаю на кнопку додавання слова до словника

REQ-1.2: Вводить одне або декілька слів для збереження

REQ-1.3: Підтверджує збереження

##### 2.3.2.4.2. Редагування та видалення слів

Опис і пріоритет:

Після додавання слів до словника, користувач може вибрати конкретне слово, видалити його або редагувати.

Пріоритет – високий.

Послідовність дія\відгук:

Користувач, використовуючи вікно зі списком останніх доданих слів чи пошук слова (стандартний або розширений). Після вибору слова відкривається нове вікно, де з’являться клавіші редагування та видалення. Після натиснення клавіші для редагування слово можна модифікувати. Після модифікації користувач натискає клавішу для збереження введених змін. Якщо ж натиснути кнопку видалення слова, слово буде вилучене із словника та відкриється вікно з останніми доданими словами.

Функціональні вимоги:

REQ-2.1: Вибір слова для редагування

REQ-2.2: Модифікація слова або його видалення

REQ-2.3: Підтвердження введених змін

##### 2.3.2.4.3. Добавлення до поточного слова певних ознак

Опис і пріоритет:

Після збереження слова у словник, надається можливість відкрити слово у новому вікні та додати до слова певні ознаки.

Пріоритет – високий.

Послідовність дія\відгук:

Користувач відкриває слово у новому вікні та додає до слова певні ознаки вже з існуючого дерева класифікацій. Після додавання ознаки ввесь ланцюжок приналежності ознаки (від кореневого класифікатора) повинен відобразитись.

Функціональні вимоги:

REQ-3.1: Користувач вибирає вже існуюче слово у словнику

REQ-3.2: Обирає необхідну ознаку, рухаючись від кореневого класифікатора до кінцевого, який і буде слугувати ознакою слова

REQ-3.3: Підтверджує зміни

##### 2.3.2.4.4. Редагування ознак слова(видалення та зміна)

Опис і пріоритет:

Користувач має можливість змінювати та видаляти приналежність слова до певних ознак.

Пріоритет – високий.

Послідовність дія\відгук:

Користувач переглядає ланцюжок ознак слова. Є можливість видалення та модифікації ознаки.

Функціональні вимоги:

REQ-4.1: Користувач натискає переходить до вкладки певного слова

REQ-4.2: Натискає клавішу видалення або редагування ознаки для слова

REQ-4.3: Підтверджує видалення або модифікацію ознаки

##### 2.3.2.4.5. Редагування дерева ознак

Опис і пріоритет:

Користувач має можливість змінювати та видаляти дерева ознак: додавати нові кореневі класифікатори, додавати дочірні класифікатори, модифікувати структуру дерев ознак.

Пріоритет – високий.

Послідовність дія\відгук:

Користувач відкриває діалогове вікно для редагування дерева ознак. Оскільки ця операція не має часто використовуватись, то й не має сенсу виносити її в основне вікно програми. Після відкриття діалогового вікна надається можливість редагування класифікаторів, якщо в параметрах програми вказаний пункт можливості редагування дерева ознак.

Функціональні вимоги:

REQ-6.1: Користувач в параметрах програми вказує опцію “Редагувати класифікатори”

REQ-6.2: Відкриває діалогове вікно редагування

REQ-6.3: Проводить необхідні модифікації

REQ-6.4: Підтверджує зміни

##### 2.3.2.4.7. Пошук слів у словнику

Опис і пріоритет:

Користувач має мати можливість швидкого пошуку слів та подальшої їх модифікації. Пошук має бути легкодоступний з усіх основних вікон програми.

Пріоритет – високий.

Послідовність дія\відгук:

Користувач, при відкритому основному вікні програми починає вводити певне слово. При введенні слова з’являється автодоповнення слова із уже наявних слів у словнику. Після введення, якщо дане слово є у словнику, програма переходить у вкладку “Слово”, показуючи шукане слово.

Функціональні вимоги:

REQ-7.1: Відкрити основне вікно програми

REQ-7.2: Ввести шукане слово

REQ-7.3: Натиснути клавішу пошуку

##### 2.3.2.4.8. Швидкий доступ до останніх доданих слів

Опис і пріоритет:

Окрім доступу до усіх слів словника користувач має мати можливість перегляду, редагування та додавання ознак до останніх створених слів (оскільки при створенні слова немає можливості одразу додати певні ознаки).

Пріоритет – високий.

Послідовність дія\відгук:

Відкривши вкладку “Головна” у колонці “Останні слова” надається можливість переглянути останні введені слова. Натиснення на одне з слів відкриває режим редагування для даного слова.

Функціональні вимоги:

REQ-8.1: Відкрити основне вікно програми

REQ-8.2: Перейти на вкладку “Головна”

REQ-8.3: Останні додані слова будуть відображені у колонці із заголовком “Останні слова”

##### 2.3.2.4.9. Додавання нових слів із файлу

Опис і пріоритет:

Для швидкого додавання слів у словник існує можливість додавати слова до словника із файлу. Слова у файлі мають бути у форматі, доволі схожому до формату CSV (comma separated value). Для відокремлення слів можна використовувати такі символи: “.”, “;”, “:”, пробіл, символ табуляції та символ нового рядка.

Пріоритет – високий.

Послідовність дія\відгук:

Відкривши вкладку “Головна” у колонці “Останні слова” надається можливість переглянути останні введені слова. Натиснення на одне з слів відкриває режим редагування для даного слова.

Функціональні вимоги:

REQ-9.1: Натиснути клавішу “Меню”

REQ-9.2: Натиснути пункт меню “Імпортувати > Слова із файлу”

REQ-9.3: У діалоговому вікні вибрати необхідний файл зі словами

##### 2.3.2.4.10. Автоматичне наповнення словника

Опис і пріоритет:

Існує можливість автоматичного аналізу тексту, розподіл слів по ознакам та наповнення бази даних словника відповідно даного розподілу.

Пріоритет – високий.

Послідовність дія\відгук:

Відкривши консольний додаток, вказати шлях до файлу. Після запуску відбувається аналіз тексту по словах та знаходження ознак приналежності до певних класифікаторів. Аналіз слів має відбуватись по морфологічній ознаці. Класифікатори для даної ознаки:

* Дієслово;
* Іменник;
* Прикметник.

Функціональні вимоги:

REQ-10.1: Відкрити консольний додаток для автоматичного наповнення слів

REQ-10.2: Вказати шлях до файлу

REQ-10.3: Запустити додаток та дочекатись завершення його роботи

### 2.3.3. Вимоги зовнішніх інтерфейсів

#### 2.3.3.1. Користувацькі інтерфейси

* Інтерфейс користувача – це веб-сервіс, що відображається у всіх відомих Інтернет браузерах, які підтримують HTML 5 та CSS 3, у вигляді сайту.
* Інтерфейс користувача реалізовуються за допомогою компонування такого набору технологій - HTML/CSS/JS.
* За допомогою фреймворка Twitter Bootstrap - сторінки сайту є адаптивними до екранів різної розмірності і мобільних пристроїв.

#### 2.3.3.2. Програмні інтерфейси

Сервер веб-сайту комунікує із зовнішнім сервером бази даних:

MS SQL – Система управління базами даних, що зберігає дані сайту. Комунікація здійснюється через ORM-технологію ADO.NET Entity Framework.

#### 2.3.3.3. Комунікаційні інтерфейси

Клієнт розгортається у веб-браузері. Серверна частина розгортається на сервері IIS. Зв’язок між клієнтською і серверною частиною здійснюється за допомогою протоколу HTTP.

### 2.3.4. Нефункціональні вимоги

#### 2.3.4.1. Вимоги продуктивності

Швидкість роботи програми буде прямо залежати від кількості слів та складності ієрархії ознак.

#### 2.3.4.2. Вимоги безпеки

* Захист від SQL ін'єкцій
* Доступність функціоналу для користувача залежно від його ролі.

#### 2.3.4.3. Вимоги надійності

У випадку збою роботи програми, користувач повинен отримати детальне повідомлення про зміст помилки і усі SQL-транзакції повинні відкотитись не впливаючи на роботу інших користувачів системи

# Розділ 3. Проектування та архітектура продукту для формування лексико-семантичного словника української мови

## 3.1. Архітектура програмного продукту

Відповідно до вимог, програма має формувати два логічних модуля для роботи із електронним лексико-семантичним словником:

* настільний додаток відображення на модифікації словника;
* консольний додаток автоматичного наповнення словника по морфологічній ознаці. До поділу по морфологічній ознаці відносяться такі частини української мови як іменник, прикметник та дієслово. Дана ідентифікація слів відбуватиметься з використанням модифікованого алгоритму Портера, спеціально підлаштованого під потреби словника.

Об’єктно-орієнтовна парадигма програмування найкраще підходить для роботи з ієрархічною моделлю ознак.

Настільний застосунок буде виконаний з використанням технології WPF – фреймворку для розробки користувацького інтерфейсу для операційної системи Windows. WPF являє собою логічне продовження WinForms. Використання XAML позволяє розділити код побудови графічного інтерфейсу користувача від коду, що формує логіку поведінки додатку.

## 3.2. Проектування бази даних

Для того, щоб створити базу даних, необхідно розпочати з роботу з її проектування. Результатом проектування повинна стати визначена структура бази даних, тобто перелік таблиць, логічні взаємозв’язки та сама структура.

Для роботи над дипломною роботою було розроблено схему бази даних з нуля (Додаток А).

Проектування даної бази було розпочато із вивчення технічного завдання. Оскільки замовник не володів відповідною термінологією, а саме він має надавати вище описане технічне завдання, було застосовано такі підходи:

* з’ясування кола задач і вимог електронного словника;
* узгодження переліку вхідних даних, які необхідні для управління структурою словника.

### 3.2.1. Концептуальна модель бази даних

Для реалізації функціоналу, щодо збереження слів та їх зв’язку з деревами ознак було прийнято рішення спроектувати базу даних з самого початку, не беручи до уваги вже існуючі рішення для електронних словників.

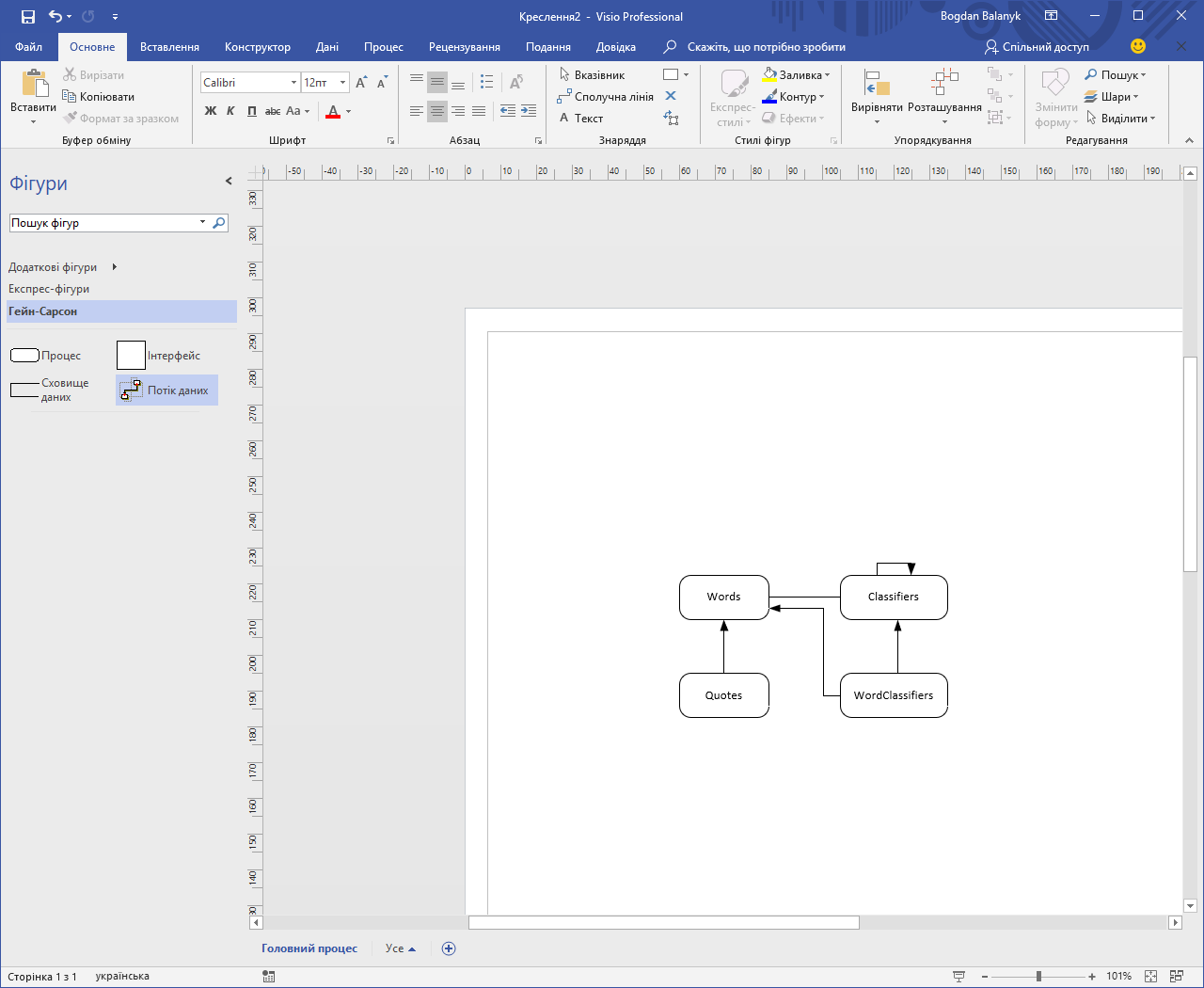


Рис. 3.1. Концептуальна модель бази даних

Першим етапом проектування є побудова концептуальної моделі, наступний етап – логічне проектування. Саме при логічному проектуванні таблиці деталізуються полями, первинними ключами.

### 3.2.2. Логічна модель бази даних

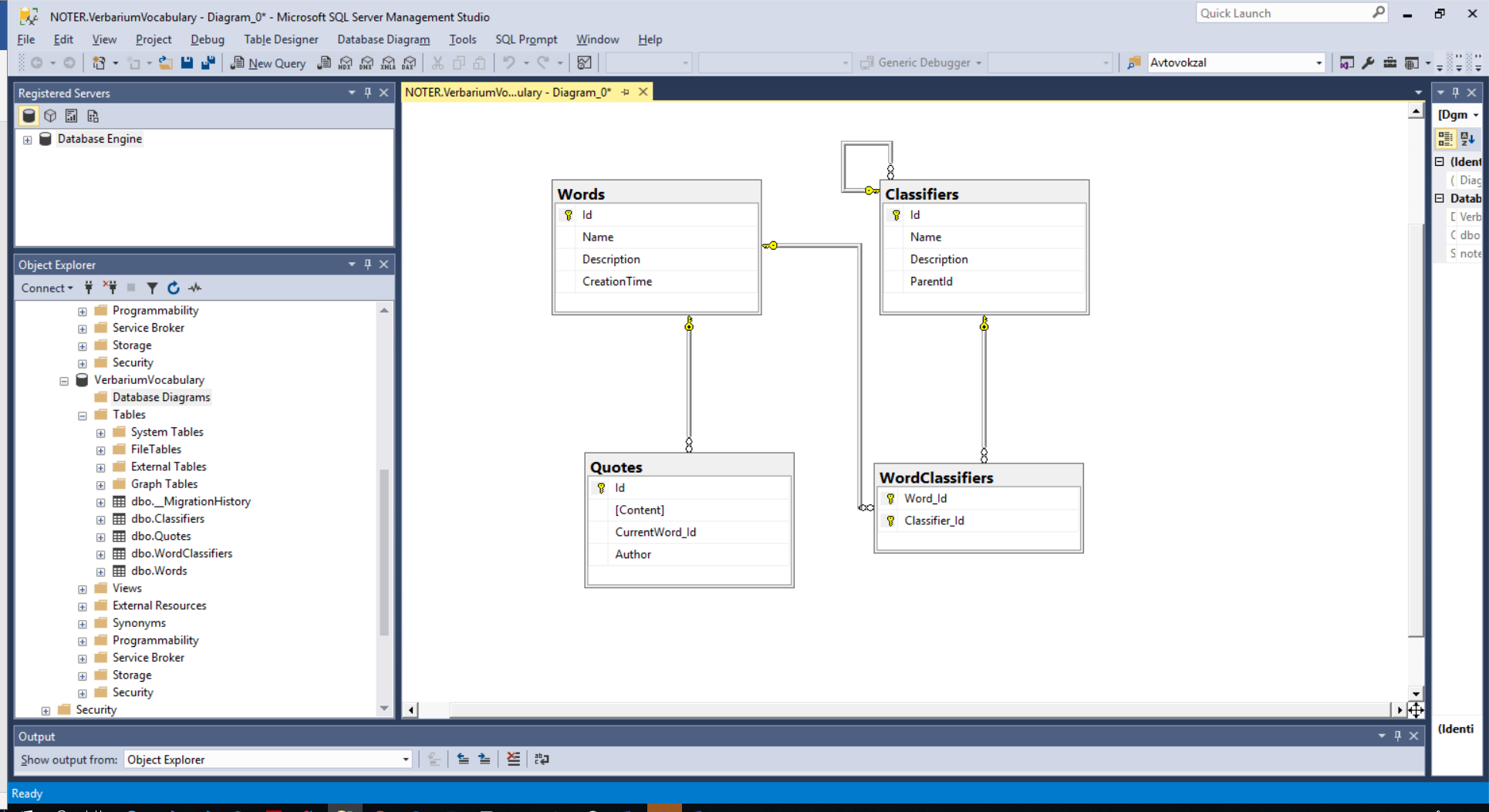


Рис. 3.2. Логічна модель бази даних

Як видно з рисунку, у таблиця «Classifiers» містить поле з посиланням на батьківську ознаку та поля з назвою та безпосереднім описом класифікатора.

Таблиця «Words» також містить назву та опис, а також додатково є поле, де зберігається інформацію про дату створення запису в таблиці. Це дозволить легко робити вибірки останніх створених слів. Швидкий доступ до останніх слів є важливою функцією у роботі з електронним словником.

Таблиця «WordClassifiers» відображає зв’язок між таблицями «Words» та «Classifiers» як «багато до багатьох». Тобто слово може відноситись до багатьох класифікаторів. В свою чергу кожен класифікатор може мати зв’язок із багатьма словами.

Таблиця «Quotes» містить інформацію про приклад використання слова в цитаті. Для зв’язку зі словом є поле з посиланням на таблицю «Words».

Кожна таблиця має первинний ключ id.

### 3.2.3. Фізична модель бази даних

Розробка фізичної моделі є останнім етапом при процесі проектування бази даних. Вона набагато більш деталізована за логічну модель, містить в собі усі таблиці, типи даних полів та їх обмеження.

Список основних таблиць, з якими проводилась робота:

* Таблиця «Words»
  + Id – числове поле, первинний ключ, обов’язкове;
  + Name – текстове поле, вміст слова, обов’язкове;
  + Description – текстове поле, опис слова, не обов’язкове.
* Таблиця «Classifiers»
  + Іd – числове поле, первинний ключ, обов’язкове;
  + Name – текстове поле, вміст слова, обов’язкове;
  + Description – текстове поле, опис слова, не обов’язкове;
  + ParentId – числове поле, зовнішній ключ, ієрархічний ідентифікатор.
* Таблиця «WordClassifiers» (зв’язок таблиць «Words» та «Classifiers» як багато до багатьох)
  + Word\_Id – числове поле, первинний та зовнішній ключ таблиці, обов’язкове;
  + Classifier\_Id – числове поле, первинний та зовнішній ключ таблиці, обов’язкове.
* Таблиця «Quotes»
  + Id – числове поле, первинний ключ таблиці, обов’язкове;
  + Name – текстове поле, вміст цитати, обов’язкове;
  + Author – текстове поле, ім’я автора цитати, не обов’язкове;
  + CurrentWord\_Id – числове поле, зовнішній ключ, обов’язкове.

## 3.3. Проектування об’єктної моделі

Для роботи з об’єктами бази даних, було використано ADO.NET Entity Framework Code First, який автоматично згенерував моделі усіх об’єктів бази даних, а також зв’язки між ними. В підході Code First основна увага приділяється домену програми. Тобто спочатку створюються класи моделей, що докорінно відрізняється від підходу Database First, де першочерговим джерелом інформації про моделі є безпосередньо інформація про таблиці в базі даних.

Перед початком розробки було спроектовано діаграму класів системи (Додаток В).

Основні класи системи:

* DataController – клас, через який відбувається доступ до сервісів, які напряму працюють з моделями бази даних;
* Classifier – клас-модель, яка містить інформацію про поля класифікаторів, відноситья до десктопного додатку;
* Word – клас-модель, яка містить інформацію про слово, відноситья до десктопного додатку;
* Quote – клас-модель, яка містить інформацію про цитати, відноситья до десктопного додатку;

# Розділ 4. Реалізація програми-словника - VERBARIUM

## 4.1. Реалізація настільного додатку

### 4.1.1. Проектування інтерфейсу користувача

Усі звіти було спроектовано у спеціальному форматі \*.rdlc, кожен з них має певний набір даних, які в ньому розміщуються, звіт може мітити як різні поля, в які можна записати певне значення, так. і таблиці які відображають певний набір даних.

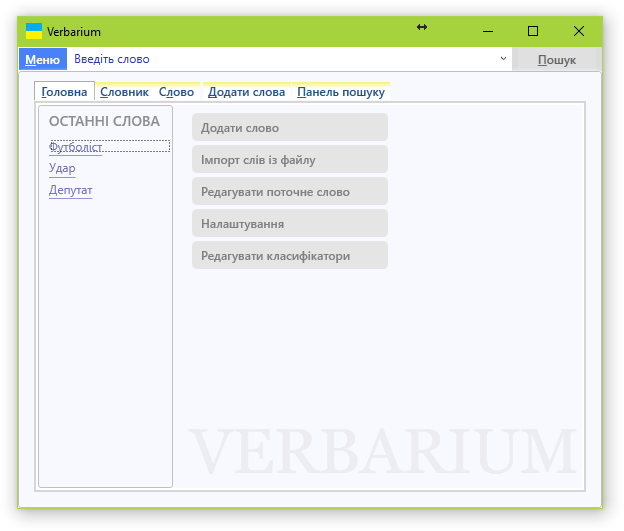


Рис. 4.1. Вигляд головного вікна програми «Verbarium»

Головне вікно програми складається з таких елементів:

* панель вкладок(Tabs panel) – використовується для зміни вмісту головного вікна;
* панель(стрічка) пошуку – дозволяє знайти необхідне слово просто ввівши кілька перший літер;
* кнопка «Пошук» – дозволяє перейти до вкладки пошук, у якій будуть відображені слова, найбільш схожі до слова введеного у стрічці пошуку;
* кнопка «Меню», після натискання якої з’являється випадний список з наступними елементами:
  + «Додати нове слово» – при натисканні на дану кнопку, вибрана вкладка буде змінена на вкладку «Додати слова»;
  + «Редагувати класифікатори» – при натисканні, відкриється нове вікно за аналогічною назвою, у якому буде можливість додати чи видалити класифікатори та ознаки;
  + «Імпортувати…» – підпункт меню, який дозволяє маніпулювати імпортом різноманітних даних у програму. На даний час підтримується тільки один варіант імпорту – імпорт слів з файлу. Однойменна команда присутня у випадному списку даного елементу. Вхідний файл із словами має бути у форматі «\*.txt». Слова всередині файлу мають бути розділені наступними роздільними символами: символом переходу на наступний рядок, символом табуляції, пробілом, крапкою, комою або крапкою з комою;
  + «Налаштування» – відкриває нове вікно у якому можна налаштувати різні параметри програми. У ньому є доступні дві вкладки: «Основні» та «Очищення». У першій вкладці доступні дві опції, що дозволяють налаштувати поведінку програми при видаленні класифікатора, та при видаленні слова. У другій вкладці доступні три кнопки: перша – видаляє всі слова із бази даних, друга видаляє всі класифікатори та їхні ознаки, а третя видаляє всі ознаки, які не прикріплені до жодного слова;
  + «Про програму» – відкриває нове вікно з інформацією про програму та її розробників;
  + «Вихід» – завершує роботу програми.

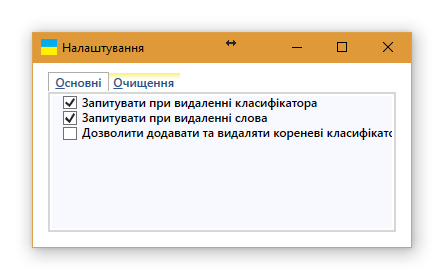


Рис. 4.2. Вигляд вкладки «Основні» у вікні «Налаштування»

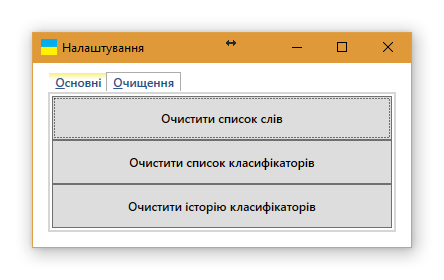


Рис. 4.3. Вигляд вкладки «Очищення» у вікні «Налаштування»

У панелі вкладок є доступними 5 різних вкладок. Перша вкладка – «Головна», яку можна побачити на рис. 3.1. На ній розташований список останніх доданих слів, а також панель швидкого доступу із функціональними кнопками, за допомогою якої, можна виконати одну із п’яти основних дій із словником.

Наступною є вкладка «Словник».

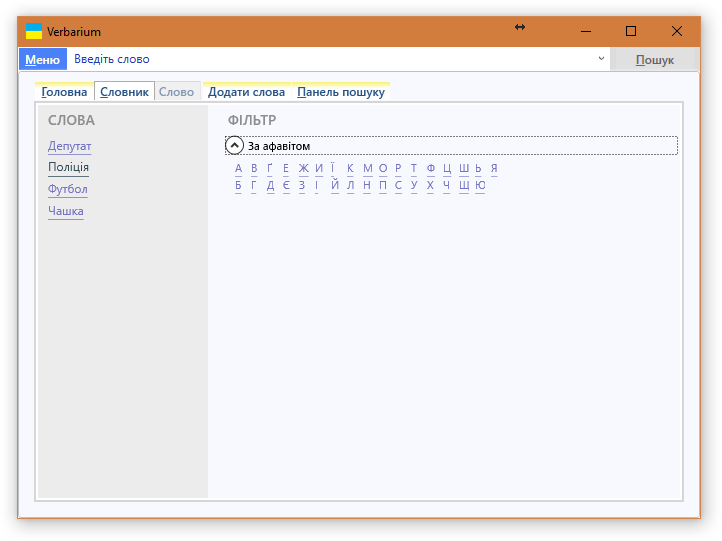


Рис. 4.3. Вигляд вкладки «Словник» у програмі «Verbarium»

На цій вкладці розташований список усіх слів відсортованих у алфавітному порядку, а також список фільтрів. На даний час, у програмі присутній тільки один фільтр – «За алфавітом». При натисканні на літеру із фільтру, у списку слів будуть відображатись лише слова, що починаються на дану літеру.

Третьою вкладкою, розміщеною на панелі вкладок є «Слово». При відкритті програми, вона буде не активною, але як тільки натиснути на будь яке слово у вкладці «Головна», «Словник» або «Панель пошуку», активна вкладка зміниться на дану вкладку.

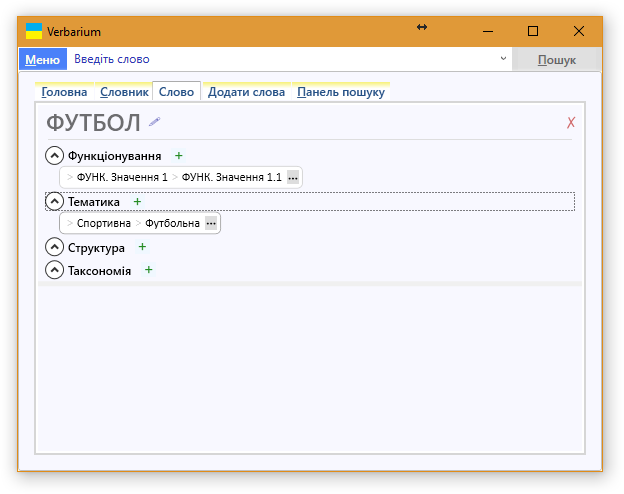


Рис. 4.4. Вигляд вкладки «Слово» у програмі «Verbarium»

У даній вкладці розміщене вибране слово. З ліва від нього, є кнопка «Редагувати», яка дозволяє його змінити. Під ним розташований дерево-подібний список усіх класифікаторів, а в них список ознак, які притаманні для даного слова. При натисканні на кнопку «+», яка розташована поруч з кожним класифікатором, у нижній частині даної вкладки відкриється список всіх ознак, у якому можна вибрати нову ознаку, і натиснувши на кнопку «зберегти», прикріпити її до даного слова.

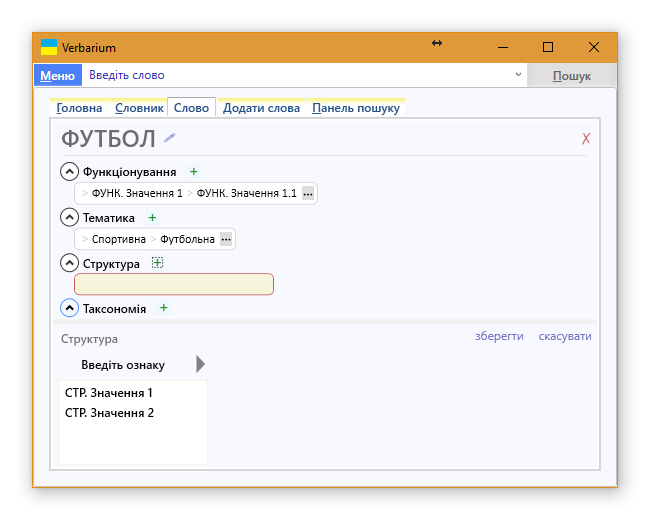


Рис. 4.5. Вигляд вкладки «Слово» в режимі прикріплення нової ознаки до даного слово

Біля кожної ознаки присутня кнопка «…», яка дозволяє відкрити список додаткових дій:

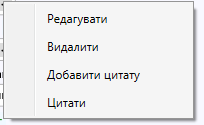


Рис. 4.6. Вигляд випадного списку кнопки «…»

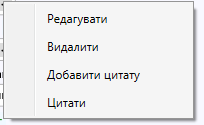
* «Редагувати» – дозволяє відредагувати ланцюжок даної ознаки, у панелі, аналогічній до панелі, яка відкривається при прикріпленні нової ознаки;
* «Видалити» – відкріпляє поточну ознаку від даного слова;
* «Додати цитату» – відкриває вікно, у якому можна додати цитату, яка буде закріплена для даної ознаки та для даного слова. У вікні можна додати текст самої цитати, від форматувавши його, а також додати автора;
* «Цитати» – відкриває список цитат даної ознаки при даному слові, у новому вікні.
* 

Рис. 4.7. Вигляд вікна «Додати цитату»

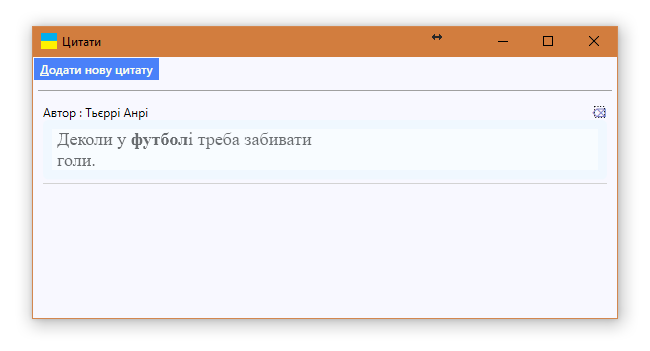


Рис. 4.8. Вигляд вікна «Додати цитату»

У вкладці розташована стрічка для вводу нового слова. Справа від неї кнопка підтвердження, після натискання на яку, введене слово буде додане у тимчасовий список нових слів. Поруч із кожним словом, є кнопка, яка дозволяє видалити його із цього списку. Внизу екрану розташовані три кнопки:

* «Додати слова» – додає усі слова із тимчасового списку у базу даних;
* «Очистити список» – очищує тимчасовий список слів;
* «Вставити із буфера» – дозволяє вставити слова раніше з тексту, що був раніше скопійований. Слова у тексті мають бути розділені такими роздільними символами: символом переходу на наступний рядок, символом табуляції, пробілом, крапкою, комою або крапкою з комою.

Остання вкладка – вкладка «Панель пошуку». У ній доступний розширений пошук за ознаками, які прикріплені до слів.

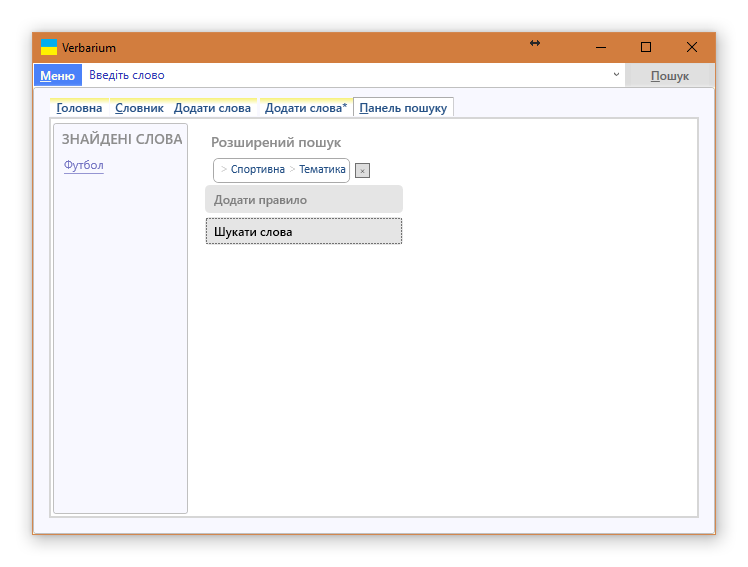


Рис. 4.9. Вигляд вкладки «Панель пошуку»

За допомогою кнопки «Додати правило» можна обрати ознаки, за якими має здійснюватися пошук, а після натискання кнопки «Шукати», у панелі «Знайдені слова», будуть відображені усі слова, до яких прикріплені дані ознаки.

### 4.1.2. Проектування внутрішньої структури побудови інтерфейсу користувача

Структуру розробленої програми можна умовно розділити на частини. Деякі з них, як наприклад вкладки, розташовані в основному коді програми, інші ж, виділено в окремі класи. Повний огляд всього коду, який охоплює тисячі рядків, займе дуже багато часу, тому пропоную розглянути модуль ClViewerControler, що відображає панель для вибору нових ознак для поточного слова.

Даний модуль можна побачити на вкладці «Слово», під час редагування існуючих чи створення ознак прив’язаних до поточного слова. Цей самий модуль, тільки з трохи іншою функціональністю розташовується на вкладці «Розширений пошук» та відповідає за вибір ознаки за якою буде здійснюватися пошук.

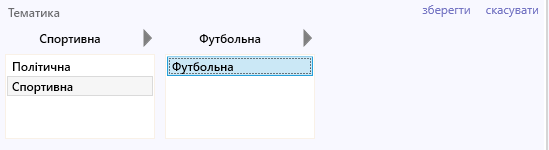


Рис. 4.10. Вигляд модуля ClViewerControler, що розташований на вкладці «Слово»

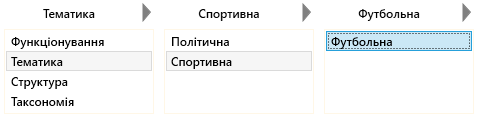


Рис. 4.11. Вигляд модуля ClViewerControler, що розташований на вкладці «Розширений пошук»

В основі даного модуля лежить елемент StackPanel, що зберігається у змінній uiMainStackPanel, та доступитись до якого із інших класів можна за допомогою властивості MainStackPanel. Створюється даний елемент за допомогою функції CreateMainStackPanel. В ній цьому елементові присвоюється унікальний ідентифікатор за допомогою властивості Name, встановлюється горизонтальний порядок відображення через задання параметру Orientation.Horizontal для властивості Orientation, а також задається відступ зверху у розмірі двох пікселів, у Margin. Інші параметри даного класу використовуються для:

* DataBase – об’єкт внутрішнього класу DataController, який відповідає за взаємодію із базою даних;
* param – об’єкт внутрішнього класу CurrParams, у якому зберігаються кількість відкритих елементів типу ListBox із списками ознак, а також список вибраних на даний час ознак;
* bHideFirstBox – параметр булевого типу, який відповідає за те, чи є необхідність відображати список кореневих класифікаторів;
* uiMainWindow – параметр типу MainWindow, який використовується для використання внутрішніх ресурсів програми, таких як стилі кнопок, для прикладу.

В конструкторі програми задаються всі вищеописані параметри, а також створюється перший елемент списку ознак, який заповнюється списком кореневих класифікаторів. Це відбувається за допомогою функції CreateAndFillNewListBox. Кореневим елементом даного блоку є StackPanel, внутрішніми елементами якого виступають ComboBox та ListBox. Перший – відповідає за відображення вибраної у даному блоці ознаки, а також для пошуку потрібної ознаки, за допомогою введення її початкових літер. Створюється даний елемент за допомогою функції CreateComboBox, параметром якої є унікальний ід номер. До створеного ComboBox задаються такі параметри:

* Text – задає текст, який буде відображатися при створенні;
* MinWidth – вказує на мінімальну ширину даного списку;
* FontSize – задає розмір тексту;
* IsEditable – задає можливість введення та редагування тексту;
* Style – застосовує стиль для даного елементу ComboBox. В даному випадку, стиль був отриманий із ресурсів MainWindow, за допомогою функції FindResource;
* DropDownOpened – додається функція, яка буде викликатися під час відкривання випадного списку. В цьому випадку це функція OnDropDownOpened, в якій блокується відкривання випадного списку, оскільки він наразі не є потрібен.

Також в даній функції додається обробник події TextBoxBase.TextChangedEvent , яка виникає при введенні тексту, а саме функції OnTextChanged, у якій відбувається пошук класифікатора за введеними літерами, а також вибір його у відповідному списку, якщо такий існує.

Елемент ListBox створюється за допомогою функції CreateListBoxForClassifiers, параметром якої також є унікальний ідентифікатор. В ній задаються такі параметри:

* BorderBrush – колір заливки контурів;
* Height – висота даного елемента.

Заповнюється даний список за допомогою циклу, який пробігається по усіх необхідних ознаках, та створює для кожної із них ListBoxItem за допомогою функції CreateListBoxItem, параметром якої є поточна ознака, який потім вставляється в кінець ListBox списку. У цьому методі, даному елементові задається параметр Content, якому встановлюється назва класифікатора. Також у ній до події Selected, додається ще один слухач, у вигляді функції OnListItemClick. Ця функція відповідає за створення та відображення наступного блоку із списком ознак, які є під елементами останньої вибраної. Спочатку в ній викликається функція DeleteNextListBoxes, яка відповідає за видалення непотрібних блоків, для прикладу коли вибраний пункт розміщений не у останньому блоці, а у передостанньому, тоді останній потрібно видалити, а на його місце вставити новий. Далі викликається вже знайома функція CreateAndFillNewListBox, параметром у якій виступає ідентифікатор батьківської, до списку який необхідно вивести, ознаки. Даний метод додає в кінець StackPanel, новий блок із списком наповненим необхідними ознаками. Залишились неописаними лише такі методи:

* GetListOfSelected – метод повертає список вибраних ознак;
* IsSelectedHasNoChildrens – повертає істину, якщо у вибраної ознаки немає внутрішніх підознак;
* UpdateStackPanel – повністю очищує головний StackPanel, за допомогою функції Clear та заповнює його початковим списком;
* SelectLineOfClassifiers – параметром даної функції виступає ознака, яка буде автоматично вибрана у відповідному блоці, після чого рекурсивно будуть обрані всі його підознаки.

## 4.2. Реалізація консольного додатку для аналізу слів автомитичного наповнення словника.

# Розділ 5. Економічна частина

## 5.1. Економічна характеристика проектного рішення (програмного продукту)

Дана бакалаврська кваліфікаційна робота передбачала розробуку підсистеми формування та друку документації та звітності у ІС «Деканат». Після завершення розробки отримуємо готовий програмний продукт, що дозволяє формувати різного типу довідки та зведені таблиці з даними студентів та їхньою успішністю.

Даний програмний продукт призначений для працівників деканатів університету, він повинен інтегрувати у вже існуючу інформаційну систему.

Розроблення даного модуля є економічно вигідне та доцільне, оскільки існуючий модуль застарілий, а аналогічні модулі на ринку не готові до інтеграції з уже існуючою системою.

## 5.2. Інформаційне забезпечення та формування гіпотези щодо потреби розроблення товару.

Зараз на ринку автоматизованих інформаційних систем для управління університетом, українського виробництва, є лише декілька продуктів, які можна використати для навчального закладу, вони платні, та вони не можуть інтегрувати з існуюючою системою.

Ось декілька із них:

* Автоматизована система управління вищим навчальним закладом III - IV рівня акредитації Uniteh+
  + Система складається з набору модулів. Усі модулі працюють в локальній мережі ВНЗ, мають спільне сховище даних. Кожен модуль має серверну та клієнтську частину, доступ до яких захищено паролем.
* АСУ «ВНЗ»
  + Дана АСУ підходить для вищих навчальних закладів різного рівня акредитації. Використовується більше ніж 70-тьма ВНЗ, відповідає усім вимогам ISO [3]. Однією з переваг системи є повна взаємодія з програмами ЄДЕБО, УЦОЯО, ІВС «Освіта».
* Пакет програм від Політек-Софт
  + Програмне забезпечення ПП "Політек-СОФТ" призначене для вищих навчальних закладів України від I до IV рівнів акредитації, охоплює майже всі аспекти їх діяльності, зареєстровано в Реєстрі виробників та розповсюджувачів програмного забезпечення, має інформаційні листи від Мiнiстерства освiти і науки, молоді та спорту України щодо використання в вищих навчальних закладах України [4].

Проаналіхувавши ринок автоматизованих систем управління університетом, я виявив потребу у розробці модуля формування та друку документації та звітності у ІС «Деканат».

## 5.3. Оцінювання та аналізування факторів зовнішнього та внутрішнього середовищ.

Результати оцінок та аналізу впливу факторів зовнішнього та внутрішнього середовищ на організацію наведено у табл. 5.1.

Таблиця 5.1

Результати експертних оцінок

|  |  |  |  |
| --- | --- | --- | --- |
| Фактори | Середня експертна оцінка, бали | Середня вагомість факторів | Зважений рівень впливу, бали |
| 1 | 2 | 3 | 4 |
| *Фактори зовнішнього середовища* |  | | |
| Споживачі | 4 | 0,11 | 0,44 |
| Постачальники | 0 | 0,1 | 0 |
| Конкуренти | 0 | 0,1 | 0,3 |
| Державні органи влади | 0 | 0,05 | 0 |
| Інфраструктура | 1 | 0,06 | 0,3 |
| Законодавчі акти | 0 | 0,1 | 0 |
| Профспілки, партії та інші громадські організації | 0 | 0,05 | 0 |
| Система економічних відносин в державі | 1 | 0,06 | 0,06 |
| Організації-сусіди | -2 | 0,01 | -0,02 |
| Міжнародні події | 0 | 0,01 | 0 |
| Міжнародне оточення | 1 | 0,03 | 0,03 |
| Науково-технічний прогрес | 3 | 0,07 | 0,21 |
| Політичні обставини | 0 | 0,06 | 0 |
| Соціально-культурні обставини | 1 | 0,05 | 0,1 |
| Рівень техніки та технологій | 3 | 0,04 | 0,12 |
| Особливості міжнародних економічних відносин | 0 | 0,02 | 0 |
| Стан економіки | 0 | 0,08 | 0 |
| Загальна сума | — | 1 | 0,70 |

*Продовження табл. 5.1*

*Фактори внутрішнього середовища*

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Цілі | 2 | 0,11 | 0,22 |
| Структура | 4 | 0,16 | 0,64 |
| Завдання | 3 | 0,07 | 0,21 |
| Технологія | 2 | 0,2 | 0,4 |
| Працівники | 4 | 0,21 | 0,84 |
| Ресурси | 3 | 0,25 | 0,75 |
| Загальна сума | — | 1 | 3,06 |

Одже, в результаті проведення оцінки та аналізу зовнішнього та внутрішнього середовища були зроблені такі висновки.

Споживачі та науковово-технічний прогрес є зовнішніми факторами, що мають найбільший вплив. Все більше і більше вищих навчальних закладів намагаються осучаснити свої інформаційні системи, особливо, це стосується осучаснення ведення документації.

Щодо внутрішніх факторів, то найбільший вплив на продук мають структура, працівники та ресурси, що логічно, оскільки продукт має пряме відношення до цих пунктів.

## 5.4. Формування стратегічних альтернатив

Стратегічні альтернативи, група №1.

Критеріями поділу альтернативних стратегій розвитку є існуючий продукт (програмний засіб) та новий, а також відповідні послуги.



Рис. 5.1. Стратегічні альтернативи, група №1

Стратегія розробки нового продукту (проектного рішення) - це стратегія, яка характеризується реалізацією абсолютно нового програмного забезпечення, якого немає на ринку і яке дає змогу вирішувати певні потреби користувача.

Стратегії розвитку нового продукту з супутніми послугами –це підхід, який включає в себе розробку нового програмного рішення і додаткові послуги.

Стратегіїя розвитку існуючого програмного продукту – це стратегія, яка передбачає модифікації та удосконалення програмного продукту.

Стратегія розвитку наявного на ринку продукту (проектного рішення) з супутніми послугами – це стратегія, яка передбачає удосконалення та оптимізацію програмного забезпечення із додатковими послугами, такими як встановлення, супроводження тощо.

Стратегічні альтернативи, група №2.

Основними критеріями, за якими поділяються альтернативні стратегії другої групи це існуючий або новий продукт (проектне рішення), а також наявний чи відсутній ринок.



Рис. 5.2. Група стратегічних альтернатив №2

Стратегія глибокого проникнення на ринок (концентрація) - загальна маркетингова стратегія, яка полягає в знаходженні шляхів збільшення збуту своїх товарів, що випускаються на вже освоєних ринках, та за допомогою більш агресивного маркетингу.

Стратегія розвитку товару передбачає збільшення обсягів збуту завдяки вдосконаленню наявних і розробці нових товарів для наявних ринків. Щодо ризикованості, то дана стратегія займає не низьку позицію.

**Стратегія розвитку ринку передбачає збільшення обсягу збуту завдяки виходу на новий ринок фірми з наявним товаром.** При цьому використовуються дві альтернативи - вихід на нові географічні ринки або орієнтація на нові сегменти ринку.

Стратегія диверсифікації передбачає розробку нових товарів для нових ринків. Зазначимо, що ця стратегія є найбільш ризиковою, разом з тим дозволяє перерозподілити ризики завдяки діяльності фірми на різних товарних ринках.

Проаналізувавши обидві групи стратегій, було обрано по одній стратегій, з кожної групи, які підходять для даного програмного продукту.

З першої групи було обрано стратегію розвитку нового продукту зі супетніми послугами, оскільки потрібно реалізувати продукт який буде забезпечувати увесь необхідний функціонал, а також потібно забезпечити його підтримку надалі.

У другій групі найкращою стратегією є стратегі ярозвитку товару, оскільки продукт може розвиватися, розширюватися, а також модуль може в майбутньому інтегрувати з іншими системами.

## 5.5. Бюджетування

Бюджетування – це необхідна складова фінансового планування, оскільки основою будь-якого оперативного чи поточного фінансового плану є система відповідних бюджетів. Як і поточне планування в цілому, бюджетування спрямоване на поетапну трансформацію стратегічного фінансового плану в систему поточних планів, послідовне їх виконання з метою досягнення стратегічних цілей підприємства.

Таблиця 5.2

Бюджет витрат матеріалів та комплектуючих виробів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Назва матеріалів та комплектуючих | Марка, тип, модель | Фактична кількість, шт. | Ціна за одиницю, грн. | Разом, грн. |
| Ноутбук | ASUS VX7SX | 1 | 9 100 | 9 100 |
| Середовище розробки | MS Visual Studio Premium  2012 | 1 | 8000 | 8000 |
| Разом: | | | | 17 100 |

Таблиця 5.3

Бюджет витрат на оплату праці

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Посада,  спеціальність | Кількість працівників, осіб | Час роботи, дні | Денна заробітна плата працівників, грн. | Сума витрат на оплату праці, грн. |
| Основна заробітна плата | | | | |
| Програміст | 1 | 14 | 400 | 5 600 |
| Тестер | 1 | 5 | 200 | 1 000 |
| Разом: | | | | 6 000 |

Таблиця 5.4

Бюджет загальновиробничих витрат

|  |  |
| --- | --- |
| Статті витрат | Сума, грн. |
| Змінні загальновиробничі витрати, у т.ч.: | |
| Заробітна плата допоміжного персоналу | 800 |
| Витрати на МШП | 120 |
| Витрати на електроенергію та технологічні цілі | 180 |
| Разом змінних витрат: | 1100 |
| Постійні загальновиробничі витрати, у т.ч.: |  |
| Комунальні послуги | 520 |
| Витрати на оренду | 2100 |
| Разом постійних витрат: | 2620 |
| Разом загальновиробничих витрат: | 3720 |

Таблиця 5.5

Бюджет адміністративних витрат та витрат на збут

|  |  |
| --- | --- |
| Статті витрат | Сума, грн. |
| Адміністративні витрати, у т.ч.: | |
| Витрати на МШП | 200 |
| Витрати на сплату податків і зборів | 450 |
| Знос адміністративного обладнання | 200 |
| Разом адміністративних витрат: | 800 |
| Витрати на збут, у т.ч.: |  |
| Заробітна плата менеджерів зі збуту | 2000 |
| Витрати на рекламу | 2100 |
| Разом витрат на збут: | 4100 |

Таблиця 5.6

Зведений кошторис витрат на розробку проектного рішення продукту

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Статті витрат | Одиниці виміру | Фактична кількість, шт. | Ціна одиниці, грн. | Разом, грн. |
| 1 | 2 | 3 | 4 | 5 |
| Сировина і матеріали | - | - | - | - |
| Комплектуючі вироби | грн | 3 | - | 17 100 |

Продовження табл. 5.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Зворотні відходи | - | - | - | - |
| Паливо та електроенергія на технологічні цілі | кВт/год | 400 | 0,45 | 180 |
| Основна заробітна плата | грн | - | - | 6000 |
| Додаткова заробітна плата | грн | - | - | - |
| Відрахування на соціальне страхування | % | 36,6 | - | 2196 |
| Витрати на утримання й експлуатацію устаткування | грн | - | - | - |
| Загальновиробничі витрати, у т.ч.: | грн | - | - | - |
| - змінні; | грн | - | - | 1100 |
| - постійні; | грн | - | - | 3720 |
| *Разом виробничих витрат:* | грн | - | - | 13196 |
| Адміністративні витрати | грн | - | - | 800 |
| Витрати на збут | грн | - | - | 4100 |
| Інші операційні витрати | грн | - | - | - |
| *Разом виробничих і операційних витрат:* | грн | - | - | 18096 |

Щоб визначити фінансові результати, нас потрібно обчислити ціну розроблюваного продукту. Вона визначається на основі суми операційних та виробничих видатків із взяттям до уваги рентабельності виробництва.

Ц = СБ \* Р.

де Ц – ціна одинці продукту, грн.

СБ – собівартість продукту, грн.

Р – рентабельність виробництва, 30%

Ц = 18096\* 0,30 = 5 430 грн.

Таблиця 5.8

Бюджет фінансових результатів

|  |  |
| --- | --- |
| Показники | Сума, грн. |
| Дохід від реалізації продукції (10 шт) | 54 300 |
| Податок на додану вартість (20%) | 10 860 |
| Чистий дохід від реалізації продукції | 43 440 |
| Собівартість реалізованої продукції | 18 096 |
| Валовий прибуток | 25 344 |
| Операційні витрати | |
| Адміністративні витрати | 800 |
| Витрати на збут | 4 100 |
| Фінансовий результат від операційної діяльності | 20 444 |
| Податок на прибуток (18%) | 3 679 |
| Чистий прибуток (збиток) | 16 769 |

## 5.6. Остаточний вибір стратегії

Провівши економічний аналіз доцільності розробки програмного продукту, можна зробити висновок, що розробка є прибутковую. Стратегією розробки було вибрано стратегію розробки нового продуку з супутніми послугами.

Даний продукт має основного кінцевого користувача – Національний університет «Львівська політехніка». Проте, модуль має перспективи до розвитку та розширення, та може знайти свою нішу на ринку серед автоматизованих систем управління.

Отже, виконавши бюджетування продукту, було встановлено собівартість продукту, яка становить 18 096 грн.  Планується реалізація 10 примірників модуля і відповідно до обрахунків бюджету чистий прибуток становитиме 16 769 грн при коефіцієнті рентабельності 30%.

# Висновки

Під час роботи я ознайомився з аналогічними системами,проаналізував їх, ознайомився з структурою існуючої ІС «Деканат».

Ознайомився із середовищами розробки MS Visual Studio 2012 та СУБД MS SQL Server 2014. Навчився налаштовувати їх. Поглибив свої знання у вивченні мови C#, а особливо технології ASP.NET MVC.

Навчився складати план робіт та слідував за ним. Виконуючи роботу я намагався максимально ефективно використовувати свій час та виконувати її як найкраще.

# Список використаної літератури

1. <https://uk.wikipedia.org/wiki/Автоматизована_система_керування_технологічним_процесом>
2. <http://www.unitex.com.ua/products/commercial-software/automated-system-for-higher-education-institution/>
3. <http://vuz.osvita.net/ua/>
4. <http://www.politek-soft.kiev.ua/>

# Додаток А

Classifiers

CREATE TABLE [dbo].[Classifiers](

[Id] [int] IDENTITY(1,1) NOT NULL,

[Name] [nvarchar](100) NOT NULL,

[Description] [nvarchar](max) NULL,

[ParentId] [int] NULL,

CONSTRAINT [PK\_dbo.Classifiers] PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

ALTER TABLE [dbo].[Classifiers] WITH CHECK ADD CONSTRAINT [FK\_dbo.Classifiers\_dbo.Classifiers\_Classifier\_Id] FOREIGN KEY([ParentId])

REFERENCES [dbo].[Classifiers] ([Id])

GO

ALTER TABLE [dbo].[Classifiers] CHECK CONSTRAINT [FK\_dbo.Classifiers\_dbo.Classifiers\_Classifier\_Id]

GO

Quotes

CREATE TABLE [dbo].[Quotes](

[Id] [int] IDENTITY(1,1) NOT NULL,

[Content] [nvarchar](max) NOT NULL,

[CurrentWord\_Id] [int] NULL,

[Author] [nvarchar](max) NULL,

CONSTRAINT [PK\_dbo.Quotes] PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

ALTER TABLE [dbo].[Quotes] WITH CHECK ADD CONSTRAINT [FK\_dbo.Quotes\_dbo.Words\_CurrentWord\_Id] FOREIGN KEY([CurrentWord\_Id])

REFERENCES [dbo].[Words] ([Id])

GO

ALTER TABLE [dbo].[Quotes] CHECK CONSTRAINT [FK\_dbo.Quotes\_dbo.Words\_CurrentWord\_Id]

GO

WordClassifiers

REATE TABLE [dbo].[WordClassifiers](

[Word\_Id] [int] NOT NULL,

[Classifier\_Id] [int] NOT NULL,

CONSTRAINT [PK\_dbo.WordClassifiers] PRIMARY KEY CLUSTERED

(

[Word\_Id] ASC,

[Classifier\_Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

ALTER TABLE [dbo].[WordClassifiers] WITH CHECK ADD CONSTRAINT [FK\_dbo.WordClassifiers\_dbo.Classifiers\_Classifier\_Id] FOREIGN KEY([Classifier\_Id])

REFERENCES [dbo].[Classifiers] ([Id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[WordClassifiers] CHECK CONSTRAINT [FK\_dbo.WordClassifiers\_dbo.Classifiers\_Classifier\_Id]

GO

ALTER TABLE [dbo].[WordClassifiers] WITH CHECK ADD CONSTRAINT [FK\_dbo.WordClassifiers\_dbo.Words\_Word\_Id] FOREIGN KEY([Word\_Id])

REFERENCES [dbo].[Words] ([Id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[WordClassifiers] CHECK CONSTRAINT [FK\_dbo.WordClassifiers\_dbo.Words\_Word\_Id]

GO

Words

CREATE TABLE [dbo].[Words](

[Id] [int] IDENTITY(1,1) NOT NULL,

[Name] [nvarchar](max) NOT NULL,

[Description] [nvarchar](max) NULL,

[CreationTime] [datetime] NULL,

CONSTRAINT [PK\_dbo.Words] PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

# Додаток Б

DataController.cs

using *System*.*Collections*.*Generic*;

using Verbarium.BLL.Desktop.Extensions.Mappers;

//using Verbarium.BLL.Desktop.Interfaces;

using Verbarium.BLL.Desktop.Interfaces;

using Verbarium.BLL.Desktop.Services;

using Verbarium.BLL.DTOs;

using VerbariumVocabulary.Extensions.Mappers;

namespace VerbariumVocabulary.Workspace

{

    public class DataController

    {

        private readonly IWordService \_wordService;

        private readonly IQuoteService \_quoteService;

        private readonly IClassifierService \_classifierService;

        public DataController()

        {

            \_wordService = new WordService();

            \_quoteService = new QuoteService();

            \_classifierService = new ClassifierService();

        }

        public bool AddClassifier(string sName, int nParentID, ref int IDClassif, string sDescription = "")

        {

            return \_classifierService.AddClassifier(sName, nParentID, ref nParentID, sDescription);

        }//

        public void DeleteAllClassif()

        {

            \_classifierService.DeleteAllClassifiers();

        }//

        public void DeleteAllWords()

        {

            \_wordService.DeleteAllWords();

        }//

        public bool DeleteClassifier(int nClassifID)

        {

            return \_classifierService.DeleteClassifier(nClassifID);

        }//

        public bool DeleteQuote(int nQuoteID)

        {

            return \_quoteService.DeleteQuote(nQuoteID);

        }//

        public bool DeleteWord(int nWordID)

        {

            return \_wordService.DeleteWord(nWordID);

        }//

        public bool DeleteWordFromClassifier(int nWordID, ClassifierForWord curClassif)

        {

            return \_classifierService.DeleteWordFromClassifier(nWordID, curClassif.ToDto());

        }//

        public bool DeleteWordFromClassifier(int nWordID, int nClassifID)

        {

            return \_classifierService.DeleteWordFromClassifier(nWordID, nClassifID);

        }//

        public void ExecuteDirectly(string sQuery)

        {

            \_wordService.ExecuteDirectly(sQuery);

        }//

        public *List*<ClassifierForWord> GetAllClassifiers(int nParentID = -1)

        {

            return \_classifierService.GetAllClassifiers(nParentID == -1 ? (int?)null : nParentID).ToClassifierList();

        }//

        public *List*<Word> GetAllWords()

        {

            return \_wordService.GetAllWords().ToWordDesktopList();

        }//

        public ClassifierForWord GetClassifier(int nClassifierID)

        {

            return \_classifierService.GetById(nClassifierID).ToClassifierDesktop();

        }//

        public int GetCountWordQuotes(int nWordID, int nClassifierID)

        {

            return \_quoteService.GetCountWordQuotes(nWordID, nClassifierID);

        }//

        public *List*<Word> GetLastWords()

        {

            return \_wordService.GetLastWords().ToWordDesktopList();

        }//

        public int GetParentID(int nClassifID)

        {

            return \_classifierService.GetParentId(nClassifID);

        }//

        public *List*<ClassifierForWord> GetRootClassifiers()

        {

            return \_classifierService.GetRootClassifiers().ToClassifierList();

        }//

        public Word GetWord(int nWordID)

        {

            return \_wordService.GetWord(nWordID).ToEntity();

        }//

        public *List*<Word> GetWordStartsWith(string startPart)

        {

            return \_wordService.GetWordStartsWith(startPart).ToWordDesktopList();

        }//

        public bool IsWordInClassifier(int nWordID, int nClassifierID)

        {

            return \_classifierService.IsWordInClassifier(nWordID, nClassifierID);

        }//

        public bool isWordPresent(string sWord)

        {

            return \_wordService.IsWordExists(sWord);

        }//

        public bool UpdateWord(int nWordID, string sNewWord)

        {

            return \_wordService.UpdateWord(nWordID, sNewWord);

        }//

        public void AddLastWord(int nWordID)

        {

            \_wordService.AddLastWord(nWordID);

        }//

        public bool AddQuote(int nWordID, int nClassifID, string sQuote, string sAuthor = null)

        {

            return \_wordService.AddQuote(nWordID, nClassifID, sQuote, sAuthor);

        }//

        public void ClearLastWords()

        {

            \_wordService.ClearLastWords();

        }//

        public void SetCountLastWords(int nCount)

        {

            \_wordService.SetCountLastWords(nCount);

        }//

        public *List*<Quote> GetQuotes(int nWordID, int nClassifID = -1)

        {

            return \_quoteService.GetAllQuotes(nWordID, nClassifID == -1 ? (int?)null : nClassifID).ToQuoteList();

        }//

        public bool NormalizeClassifiers()

        {

            return \_classifierService.NormalizeClassifiers();

        }//

        public bool AddParentsForWord(int nWordID, *List*<ClassifierForWord> parents)

        {

            return \_classifierService.AddWord(nWordID, parents.ToClassifierDtoList());

        }//

        public bool AddWord(string sWord, *List*<ClassifierForWord> parents, ref int nIDWord)

        {

            var word = \_wordService.Create(new WordDto

            {

                Name = sWord

            });

            nIDWord = word.Id;

            return AddParentsForWord(word.Id, parents);

        }//

        public *List*<Word> FindWords(*List*<int> arrClassifIDs, string sSearchCondition)

        {

            return \_wordService.FindWords(arrClassifIDs).ToWordDesktopList();

        }//

        public *List*<ClassifierForWord> GetWordParents(int nWordID, bool bMerge = true)

        {

            return \_classifierService.GetWordParents(nWordID, bMerge).ToClassifierList();

        }//

    }

}

Classifier.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace VerbariumVocabulary

{

    public class Classifier

    {

        private int id;

        public int Id

        {

            get { return id; }

        }

        private string sName;

        public string Name

        {

            get { return sName; }

            set { sName = *value*; }

        }

        private string sDescription;

        public string Description

        {

            get { return sDescription; }

            set { sDescription = *value*; }

        }

        public Classifier(int id, string sName, string sDescription = "")

        {

            this.id = id;

            Name = sName;

            Description = sDescription;

        }

    }

    public class ClassifierForWord

    {

        private int id;

        public int Id

        {

            get { return id; }

            set { id = *value*; }

        }

        private string sName;

        public string Name

        {

            get { return sName; }

            set { sName = *value*; }

        }

        private string sDescription;

        public string Description

        {

            get { return sDescription; }

            set { sDescription = *value*; }

        }

        public ClassifierForWord(int id, string sName, string sDescription = "")

        {

            this.id = id;

            Name = sName;

            Description = sDescription;

        }

        public ClassifierForWord()

        {

        }

        public *List*<ClassifierForWord> ChildClassifiers = new *List*<ClassifierForWord>();

    }

    public class ClassifierForWordTempBeforeSaving : ClassifierForWord

    {

        private int parentId;

        public int ParentId

        {

            get { return parentId; }

            set { parentId = *value*; }

        }

        public ClassifierForWordTempBeforeSaving(string sName, int nParentId, string sDescription = "")

        {

            Name = sName;

            Description = sDescription;

            parentId = nParentId;

        }

    }

}

Quote.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

namespace VerbariumVocabulary

{

    public class Quote

    {

        public Quote(string text, string author, int nID, int wordID, int classifID)

        {

            \_sText = text;

            \_sAuthor = author;

            \_nID = nID;

            \_nWordID = wordID;

            \_nClassifID = classifID;

        }

        public string \_sText { get; set; }

        public string \_sAuthor { get; set; }

        public int \_nWordID { get; set; }

        public int \_nClassifID { get; set; }

        public int \_nID { get; set; }

    }

}

Word.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace VerbariumVocabulary

{

    public class Word

    {

        public Word(string InsWord, int nID = -1) {

            sWord = InsWord;

            this.nID = nID;

        }

        public string sWord;

        public int nID;

        public *List*<ClassifierForWord> ListOfClassifiers = new *List*<ClassifierForWord>();

    }

}

ClassEditorListStackPanel.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

using *System*.*Windows*;

using *System*.*Windows*.*Controls*;

using *System*.*Windows*.*Data*;

using *System*.*Windows*.*Documents*;

using *System*.*Windows*.*Input*;

using *System*.*Windows*.*Media*;

using *System*.*Windows*.*Media*.*Imaging*;

using *System*.*Windows*.*Navigation*;

using *System*.*Windows*.*Shapes*;

using *System*.*IO*;

using *System*.*Windows*.*Markup*;

using VerbariumVocabulary.Workspace;

namespace VerbariumVocabulary

{

    /// <summary>

    /// Classificatory Editor List Controller

    /// </summary>

    public class ClEditorControler

    {

        public ClEditorControler(ref DataController DB, MainWindow CurrMainWindow)

        {

            DataBase = DB;

            param = new CurrParams();

            uiMainWindow = CurrMainWindow;

            uiMainStackPanel = CreateMainStackPanel();

            uiMainScroll = new *ScrollViewer*

            {

*VerticalScrollBarVisibility* = *ScrollBarVisibility*.*Auto*,

*HorizontalScrollBarVisibility* = *ScrollBarVisibility*.*Auto*,

            };

            uiMainScroll.*Content* = uiMainStackPanel;

            uiMainStackPanel.*ScrollOwner* = uiMainScroll;

            uiMainScroll.*ScrollChanged* += new *ScrollChangedEventHandler*(OnMainScrollVisibilityChanged);

            uiBackPointerButton = CreatePointerGradientButton(true);

            uiForwardPointerButton = CreatePointerGradientButton(false);

            uiMainGrid = new *Grid*();

            uiMainGrid.*Children*.*Add*(uiMainScroll);

            uiMainGrid.*Children*.*Add*(uiBackPointerButton);

            uiMainGrid.*Children*.*Add*(uiForwardPointerButton);

            CreateAndFillNewListBox();

        }

*ScrollViewer* uiMainScroll;

        DataController DataBase;

        CurrParams param;

        MainWindow uiMainWindow;

*StackPanel* uiMainStackPanel;

*Grid* uiMainGrid;

*Button* uiBackPointerButton;

*Button* uiForwardPointerButton;

        public *StackPanel* MainStackPanel

        {

            get { return this.uiMainStackPanel; }

        }

        public *Grid* MainGrid

        {

            get { return this.uiMainGrid; }

        }

        public *List*<ClassifierForWord> GetListOfSelected()

        {

            return param.ListChosenClassifiers;

        }

*StackPanel* CreateMainStackPanel()

        {

*Color* clrBG = *Colors*.*White*;

*StackPanel* currStackPanel = new *StackPanel*

            {

                Name = "uiStackPaneListClassEditExp",

*Orientation* = *Orientation*.*Horizontal*,

*HorizontalAlignment* = *HorizontalAlignment*.*Stretch*,

*VerticalAlignment* = *VerticalAlignment*.*Stretch*,

*Background* = new *SolidColorBrush*(clrBG),

*Margin* = new *Thickness*(0, 2, 0, 0)

            };

            return currStackPanel;

        }

*Button* CreatePointerGradientButton(bool isLeft)

        {

*LinearGradientBrush* myHorizontalGradient =

                new *LinearGradientBrush*();

            myHorizontalGradient.*StartPoint* = new *Point*(0, 0.5);

            myHorizontalGradient.*EndPoint* = new *Point*(3, 0.5);

            myHorizontalGradient.*GradientStops*.*Add*(

               new *GradientStop*(isLeft ? *Colors*.*Gray* : *Colors*.*Transparent*, 0.0));

            myHorizontalGradient.*GradientStops*.*Add*(

                new *GradientStop*(isLeft ? *Colors*.*Transparent* : *Colors*.*Gray*, 0.5));

*Button* Butt = new *Button*

            {

*FontSize* = 10,

*VerticalContentAlignment* = *VerticalAlignment*.*Center*,

                Content = isLeft ? "⇦" : "⇨",

*Width* = 20,

*HorizontalAlignment* = isLeft ? *HorizontalAlignment*.*Left* : *HorizontalAlignment*.*Right*,

*BorderThickness* = new *Thickness*(0),

*Background* = myHorizontalGradient,

*Visibility* = *Visibility*.*Collapsed*

            };

            Butt.*Click* += new *RoutedEventHandler*((object s, *RoutedEventArgs* reh) =>

            {

                if(isLeft)

                    uiMainScroll.*ScrollToLeftEnd*();

                else

                    uiMainScroll.*ScrollToRightEnd*();

            });

            return Butt;

        }

        void CreateAndFillNewListBox(int ClassID = 0)

        {

*List*<ClassifierForWord> LstOfCurrClassifiers;

            if (ClassID == 0)

            {

                LstOfCurrClassifiers = DataBase.GetRootClassifiers();

            }

            else

            {

                LstOfCurrClassifiers = DataBase.GetAllClassifiers(ClassID);

            }

*ListBox* uiCurrListBox = CreateListBoxForClassifiers(param.CountOfOpenedListBoxes.*ToString*());

*List*<string> CbList = new *List*<string>();

            foreach (ClassifierForWord var in LstOfCurrClassifiers)

            {

                CbList.*Add*(var.Name);

            }

*ListBoxItem* uiListBox = CreateComboBoxForNewClass((param.CountOfOpenedListBoxes - 1).*ToString*(), CbList);

            //Filling

            uiCurrListBox.*Items*.*Add*(uiListBox);

            foreach (ClassifierForWord cl in LstOfCurrClassifiers)

            {

*ListBoxItem* uiListItem = CreateListBoxItem(cl);

                uiCurrListBox.*Items*.*Add*(uiListItem);

            }

            uiMainStackPanel.*Children*.*Add*(uiCurrListBox);

        }

        public void OnMainScrollVisibilityChanged(object ob, *ScrollChangedEventArgs* arg)

        {

*ScrollViewer* sv = (*ScrollViewer*)ob;

            uiBackPointerButton.*Visibility* = sv.*HorizontalOffset* > 0 ? *Visibility*.*Visible* : *Visibility*.*Collapsed*;

            uiForwardPointerButton.*Visibility* = (sv.*HorizontalOffset*) == 0 && (sv.*ComputedHorizontalScrollBarVisibility* == *Visibility*.*Visible*) ? *Visibility*.*Visible* : *Visibility*.*Collapsed*;

        }

        public void UpdateStackPanel()

        {

*List*<ClassifierForWord> tempList = new *List*<ClassifierForWord>(param.ListChosenClassifiers);

            param.Reset();

            uiMainStackPanel.*Children*.*Clear*();

            CreateAndFillNewListBox();

            SelectLineOfClassifiers(tempList);

        }

        public void SelectLineOfClassifiers(*List*<ClassifierForWord> ClassifForSel)

        {

            for (int j = 0; j < uiMainStackPanel.*Children*.*Count*; j++)

            {

                bool bIsSelected = false;

                if (ClassifForSel.*Count* > j)

                {

*ListBox* uiCurrListBox = (*ListBox*)uiMainStackPanel.*Children*[j];

                    for (int i = 1; i < uiCurrListBox.*Items*.Count; i++)

                    {

*ListBoxItem* uiCurrLBI = (*ListBoxItem*)uiCurrListBox.*Items*[i];

                        string[] Ids = uiCurrLBI.*Name*.*Split*('\_');

                        int ClassifierId = *Convert*.*ToInt32*(Ids[1]);

                        if (ClassifForSel[j].Id.*Equals*(ClassifierId))//select

                        {

                            bIsSelected = true;

                            uiCurrListBox.*SelectedItem* = uiCurrLBI;

                        }

                    }

                }

                if (!bIsSelected)//if nothing has been selected stop

                    break;

            }

        }

        void DeleteNextListBoxes(int After)

        {

            for (int i = uiMainStackPanel.*Children*.*Count* - 1; i > After - 1; i--)

            {

                if (i > After)

                {

                    uiMainStackPanel.*Children*.*RemoveAt*(i);

                    param.CountOfOpenedListBoxes--;

                }

                if (i < param.ListChosenClassifiers.*Count*)

                    param.ListChosenClassifiers.*RemoveAt*(i);

            }

        }

*ListBox* CreateListBoxForClassifiers(string Id)

        {

            param.CountOfOpenedListBoxes++;

*Color* clr = (*Color*)*ColorConverter*.*ConvertFromString*("#FFCC66");

            clr.*A* = 40;

*ListBox* uiMainListBox = new *ListBox*

            {

*MinWidth* = 150,

*Margin* = new *Thickness*(5, 0, 5, 0),

*BorderBrush* = new *SolidColorBrush*(clr),

                Name = "uiListBoxClassId\_" + Id,

            };

            if (Id.*Equals*("0"))

                uiMainListBox.*Background* = new *SolidColorBrush*(*Colors*.*LightGray*);

            return uiMainListBox;

        }

*ListBoxItem* CreateListBoxItem(ClassifierForWord currClass)

        {

*WrapPanel* uiWrapPanel = new *WrapPanel*();

*TextBlock* uiTxtBox = new *TextBlock*();

            uiTxtBox.*Text* = currClass.Name;

*StringBuilder* sb = new *StringBuilder*();

            sb.*Append*(@"<Button xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation' Grid.Column='1' Height='15' BorderThickness='0' Content='✗' FontSize='12' Padding='1, -1, 1, 0' Margin='4, 0, 0, 0' Foreground='#FFF91E08'/>");

*Button* tButtonDel = (*Button*)*XamlReader*.*Parse*(sb.*ToString*());

            tButtonDel.*Background* = *Brushes*.*Transparent*;

            tButtonDel.*Click* += OnBtnDeleteClassifierClick;

            if (DataBase.GetParentID(currClass.Id).*Equals*(-1) && !VerbariumVocabulary.Properties.Settings.Default.AllowAddNewRootClass)

                tButtonDel.*Visibility* = *Visibility*.*Collapsed*;

            uiWrapPanel.*Children*.*Add*(uiTxtBox);

            uiWrapPanel.*Children*.*Add*(tButtonDel);

*ListBoxItem* uiCurrListItem = new *ListBoxItem*

            {

                Name = "uiLstItemClassId\_" + currClass.Id.*ToString*(),

                Content = uiWrapPanel,

            };

            uiCurrListItem.*Selected* += new *RoutedEventHandler*(OnListItemClick);

            return uiCurrListItem;

        }

*ListBoxItem* CreateComboBoxForNewClass(string Id, *List*<string> CbList)

        {

*WrapPanel* uiWrapPanel = new *WrapPanel*();

*ComboBox* uiCurrComboBox = new *ComboBox*

            {

*Text* = "Нове значення",

                Name = "uiComboBoxEditClassId\_" + Id,

*MinWidth* = 150,

*IsEditable* = true,

            };

            uiCurrComboBox.*ItemsSource* = CbList;

*StringBuilder* sb = new *StringBuilder*();

            sb.*Append*(@"<Button xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation' Grid.Column='1' Height='15' BorderThickness='0' Content='✔' FontSize='12' Padding='1, -1, 1, 0' Margin='4, 0, 0, 0' Foreground='#FF1EF908'/>");

*Button* tButtonAdd = (*Button*)*XamlReader*.*Parse*(sb.*ToString*());

            tButtonAdd.*Background* = *Brushes*.*Transparent*;

            tButtonAdd.*Click* += OnBtnAddNewClassifierClick;

            uiWrapPanel.*Children*.*Add*(uiCurrComboBox);

            uiWrapPanel.*Children*.*Add*(tButtonAdd);

*ListBoxItem* uiCurrListItem = new *ListBoxItem*

            {

                Content = uiWrapPanel,

            };

            if (Id.*Equals*("0") && !VerbariumVocabulary.Properties.Settings.Default.AllowAddNewRootClass)

                uiCurrListItem.*Visibility* = *Visibility*.*Collapsed*;

            return uiCurrListItem;

        }

        private void OnBtnDeleteClassifierClick(object sender, *RoutedEventArgs* e)

        {

            if (VerbariumVocabulary.Properties.Settings.Default.AskWhenClassIsDeleting)

            {

*MessageBoxResult* result = *MessageBox*.*Show*("Видалити класифікатор та його підкласифікатори?", "Підтвердження", MessageBoxButton.YesNo, MessageBoxImage.Question);

                if (result == *MessageBoxResult*.*No*)

                    return;

            }

*ListBoxItem* uiCurrListBoxItem = (*ListBoxItem*)((*WrapPanel*)((*Button*)sender).*Parent*).*Parent*;

*ListBox* uiCurrListBox = (*ListBox*)uiCurrListBoxItem.*Parent*;

            string[] sID = uiCurrListBoxItem.Name.*Split*('\_');

            int nListBoxItemID = *Convert*.*ToInt32*(sID[1]);

            DataBase.DeleteClassifier(nListBoxItemID);

            UpdateStackPanel();

        }

        private void OnBtnAddNewClassifierClick(object sender, *RoutedEventArgs* e)

        {

*WrapPanel* uiCurrWrapPanel = (*WrapPanel*)((*Button*)sender).*Parent*;

*ListBox* uiCurrListBox = (*ListBox*)((*ListBoxItem*)uiCurrWrapPanel.*Parent*).*Parent*;

            string[] sID = uiCurrListBox.Name.*Split*('\_');

            int nListBoxID = *Convert*.*ToInt32*(sID[1]);

*ComboBox* uiCurrComboBox = (*ComboBox*)uiCurrWrapPanel.*Children*[0];

            int NewId = 0;

            if(param.ListChosenClassifiers.*Count* == 0)

                DataBase.AddClassifier(uiCurrComboBox.*Text*, -1, ref NewId);

            else

                DataBase.AddClassifier(uiCurrComboBox.*Text*, param.ListChosenClassifiers[nListBoxID - 1].Id, ref NewId);

            UpdateStackPanel();

        }

        private void OnListItemClick(object sender, *RoutedEventArgs* e)

        {

*ListBoxItem* uiCurrListBoxItem = (*ListBoxItem*)sender;

            string[] Ids = uiCurrListBoxItem.Name.*Split*('\_');

            int ClassifierId = *Convert*.*ToInt32*(Ids[1]);

*ListBox* CurrListBox = (*ListBox*)uiCurrListBoxItem.*Parent*;

            string[] sID = CurrListBox.*Name*.*Split*('\_');

            int nListBoxID = *Convert*.*ToInt32*(sID[1]);

            DeleteNextListBoxes(nListBoxID);

            CreateAndFillNewListBox(ClassifierId);

            ClassifierForWord SelClass = DataBase.GetClassifier(ClassifierId);

            param.ListChosenClassifiers.*Add*(SelClass);

        }

    }

}

ClassOfWordViewerStPanel.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

using *System*.*Windows*;

using *System*.*Windows*.*Controls*;

using *System*.*Windows*.*Data*;

using *System*.*Windows*.*Documents*;

using *System*.*Windows*.*Input*;

using *System*.*Windows*.*Media*;

using *System*.*Windows*.*Media*.*Imaging*;

using *System*.*Windows*.*Navigation*;

using *System*.*Windows*.*Shapes*;

using *System*.*IO*;

using *System*.*Windows*.*Markup*;

using *System*.*ComponentModel*;

using VerbariumVocabulary.Workspace;

namespace VerbariumVocabulary

{

    public class WordQuoteData : *INotifyPropertyChanged*

    {

        private string myDataProperty;

        public WordQuoteData() { }

        public DataController DB { get; set; }

        public int WordID { get; set; }

        public int ClassifierID { get; set; }

        public WordQuoteData(DataController db, int nWordID, int nClassifID)

        {

            DB = db;

            WordID = nWordID;

            ClassifierID = nClassifID;

        }

        public *String* CountWordQuotes

        {

            get { return DB.GetCountWordQuotes(WordID, ClassifierID).*ToString*(); }

            set

            {

                myDataProperty = *value*;

                OnPropertyChanged("CountWordQuotes");

            }

        }

        public event *PropertyChangedEventHandler* PropertyChanged;

        private void OnPropertyChanged(string info)

        {

*PropertyChangedEventHandler* handler = PropertyChanged;

            if (handler != null)

            {

                handler(this, new *PropertyChangedEventArgs*(info));

            }

        }

    }

    /// <summary>

    /// Line Of Strings/Classifiers Controller

    /// </summary>

    ///

    public class ClLineController

    {

        WordQuoteData myDataObject = new WordQuoteData();

        public ClLineController(*List*<string> StringsInLineIn, int LineIDIn, MainWindow CurrMainWindow, bool ShowContextMenuButton = true)

        {

            StringsInLine = StringsInLineIn;

            Init(LineIDIn, CurrMainWindow, ShowContextMenuButton);

        }

        public ClLineController(ClassifierForWord ClassifInLineIn, int LineIDIn, MainWindow CurrMainWindow, bool ShowContextMenuButton = true)

        {

            ClassInLine = ClassifInLineIn;

            Init(LineIDIn, CurrMainWindow, ShowContextMenuButton);

        }

        MainWindow uiMainWindow;

*StackPanel* uiMainStackPanel;

*StackPanel* uiInsideStackPanel;

*ContextMenu* uiCurrContextMenu;

        Border uiMainBorder;

*List*<*UIElement*> lCustomControls;

        ClViewerControler ViewerController;

        Enums.UIObjectState CurrentState = Enums.UIObjectState.NORMAL;

        bool ShowContextMenuButton = true;

        public *StackPanel* MainStackPanel

        {

            get { return this.uiMainStackPanel; }

        }

        public Border MainBorder

        {

            get { return this.uiMainBorder; }

        }

        public Enums.UIObjectState LineState

        {

            get { return this.CurrentState; }

        }

*List*<string> StringsInLine;

        ClassifierForWord ClassInLine;

        int LineID;

        void Init(int LineIDIn, MainWindow CurrMainWindow, bool ShowContextMenuButton)

        {

            LineID = LineIDIn;

            this.ShowContextMenuButton = ShowContextMenuButton;

            lCustomControls = new *List*<*UIElement*>();

            uiMainWindow = CurrMainWindow;

            uiMainStackPanel = CreateMainStackPanel();

            uiMainBorder = CreateRoundedBorder();

            //uiMainWindow.SizeChanged += new SizeChangedEventHandler(OnSizeChanged);

            //OnSizeChanged(null, null);

            uiInsideStackPanel = CreateInsideStackPanel();

            uiMainBorder.*Child* = uiInsideStackPanel;

            uiMainStackPanel.*Children*.*Add*(uiMainBorder);

            uiCurrContextMenu = CreateContextMenuForMoreButton();

            Update();

        }

        void InitStandardContextMenu()

        {

            AddMoreMenuItem("Редагувати", "Редагувати класифікатор", OnEditClick);

            AddMoreMenuItem("Видалити", "Видалити класифікатор", OnDeleteClick);

            AddMoreMenuItem("Добавити цитату", "Добавити цитату для поточної ознаки", OnAddWordQuotesClick);

            var QuoteButton = AddMoreMenuItem("Цитати", "Відкрити список цитат класифікатора", OnWordQuotesClick);

            /\*ClassifierForWord LastClass = ClassInLine;

            while (LastClass.ChildClassifiers.Count > 0)

                LastClass = LastClass.ChildClassifiers[0];

            WordQuoteData myDataObject = new WordQuoteData(uiMainWindow.DB, uiMainWindow.currentSelectedWord.nID, LastClass.Id);

            Binding myBinding = new Binding("CountWordQuotes");

            myBinding.Source = myDataObject;

            QuoteButton.SetBinding(MenuItem.IconProperty, myBinding);\*/

        }

        public void Update()

        {

            uiInsideStackPanel.*Children*.*Clear*();

            if (StringsInLine != null)

            {

                for (int i = 0; i < StringsInLine.Count; i++)

                {

                    uiInsideStackPanel.*Children*.*Add*(CreatePointerTextBl());

                    uiInsideStackPanel.*Children*.*Add*(CreateTextBlItem(StringsInLine[i], i));

                }

            }

            else if (ClassInLine != null)

            {

                ClassifierForWord TempClass = ClassInLine;

                while (TempClass.ChildClassifiers.*Count* > 0)

                {

                    TempClass = TempClass.ChildClassifiers[0];

                    uiInsideStackPanel.*Children*.*Add*(CreatePointerTextBl());

                    uiInsideStackPanel.*Children*.*Add*(CreateTextBlItem(TempClass));

                }

            }

            if (ShowContextMenuButton)

            {

                uiInsideStackPanel.*Children*.*Add*(CreateMoreButton());

                if (ClassInLine != null)

                    InitStandardContextMenu();

            }

            foreach(var control in lCustomControls)

            {

                uiInsideStackPanel.*Children*.*Add*(control);

            }

        }

        public ClassifierForWord GetLastClassifier()

        {

            if (ClassInLine != null)

            {

                while (ClassInLine.ChildClassifiers.*Count* != 0)

                {

                    ClassInLine = ClassInLine.ChildClassifiers[0];

                }

                return ClassInLine;

            }

            else

                return null;

        }

        public *List*<string> GetCurrListOfString()

        {

            if (StringsInLine != null)

            {

                return StringsInLine;

            }

            else

                return null;

        }

        public bool HasStringsAsInput()

        {

            return StringsInLine.Count > 0;

        }

        public void SetState(Enums.UIObjectState state)

        {

            switch (state)

            {

                case Enums.UIObjectState.NORMAL:

                    uiMainBorder.*Background* = new *SolidColorBrush*(*Colors*.*White*);

                    uiMainBorder.*BorderBrush* = new *SolidColorBrush*(*Colors*.*LightGray*);

                    break;

                case Enums.UIObjectState.SELECTED:

                    uiMainBorder.*Background* = new *SolidColorBrush*(*Colors*.*Beige*);

                    uiMainBorder.*BorderBrush* = new *SolidColorBrush*(*Colors*.*IndianRed*);

                    break;

                case Enums.UIObjectState.HOWERED:

                    uiMainBorder.*Background* = new *SolidColorBrush*(*Colors*.*AliceBlue*);

                    uiMainBorder.*BorderBrush* = new *SolidColorBrush*(*Colors*.*LightGray*);

                    break;

                default:

                    break;

            }

            CurrentState = state;

        }

        public void SetLine(ClassifierForWord ClassifInLineIn)

        {

            StringsInLine = null;

            ClassInLine = ClassifInLineIn;

        }

        public void SetLine(*List*<string> StringsInLineIn)

        {

            ClassInLine = null;

            StringsInLine = StringsInLineIn;

        }

        /// <summary>

        /// Controller of referenced view of classifiers for automatically selecting current list of classifiers

        /// </summary>

        /// <param name="CtrlView"></param>

        public void SetRefViewerControler(ClViewerControler CtrlView)

        {

            ViewerController = CtrlView;

        }

        public *MenuItem* AddMoreMenuItem(string sHeader, string sToolTip = "", *RoutedEventHandler* ClickHandler = null)

        {

*MenuItem* currMenuItem = new *MenuItem*

            {

*Margin* = new *Thickness*(5, 3, 0, 3),

*Header* = sHeader,

*ToolTip* = sToolTip,

            };

            if (ClickHandler != null)

                currMenuItem.*Click* += ClickHandler;

            uiCurrContextMenu.*Items*.*Add*(currMenuItem);

            return currMenuItem;

        }

        public void AddCustomControlToEnd(*UIElement* CustomControl)

        {

            lCustomControls.*Add*(CustomControl);

            Update();

        }

        Border CreateRoundedBorder()

        {

            Border uiBorder = new Border

            {

*CornerRadius* = new *CornerRadius*(5),

*BorderBrush* = new *SolidColorBrush*(*Colors*.*DarkGray*),

*Background* = new *SolidColorBrush*(*Colors*.*White*),

*Padding* = new *Thickness*(3),

*BorderThickness* = new *Thickness*(1),

*MinHeight* = 22,

*HorizontalAlignment* = *HorizontalAlignment*.*Right*

            };

            return uiBorder;

        }

*StackPanel* CreateMainStackPanel()

        {

*StackPanel* currStackPanel = new *StackPanel*

            {

                Name = "uiStackPanelListOfClass\_" + LineID.*ToString*(),

*Orientation* = *Orientation*.*Horizontal*,

*HorizontalAlignment* = *HorizontalAlignment*.*Stretch*,

*VerticalAlignment* = *VerticalAlignment*.*Stretch*,

            };

            currStackPanel.*MouseLeftButtonUp* += new *MouseButtonEventHandler*(OnMouseUp);

            currStackPanel.*MouseLeave* += new *MouseEventHandler*(OnMouseLeave);

            currStackPanel.*MouseEnter* += new *MouseEventHandler*(OnMouseEnter);

            return currStackPanel;

        }

*StackPanel* CreateInsideStackPanel()

        {

*StackPanel* insideStackPanel = new *StackPanel*

            {

*Orientation* = *Orientation*.*Horizontal*,

*HorizontalAlignment* = *HorizontalAlignment*.*Stretch*,

*VerticalAlignment* = *VerticalAlignment*.*Stretch*,

            };

            return insideStackPanel;

        }

*TextBlock* CreateTextBlItem(ClassifierForWord CurrClass)

        {

            return CreateTextBlItem(CurrClass.Name, CurrClass.Id);

        }

*TextBlock* CreateTextBlItem(string Str, int ID)

        {

*TextBlock* uiCurrComboBox = new *TextBlock*

            {

*Text* = Str,

                Name = "uiCmbBoxMemberOfList\_" + ID.*ToString*(),

*FontFamily* = new *FontFamily*("Calibri"),

*Background* = new *SolidColorBrush*(*Colors*.*Transparent*),

            };

            return uiCurrComboBox;

        }

*TextBlock* CreatePointerTextBl()

        {

*Color* clr = *Colors*.*Gray*;

            clr.*A* = 80;

*TextBlock* uiCurrComboBox = new *TextBlock*

            {

*Text* = ">",

*FontSize* = 12,

*Foreground* = new *SolidColorBrush*(clr),

*FontFamily* = new *FontFamily*("Calibri"),

*Background* = new *SolidColorBrush*(*Colors*.*Transparent*),

*Margin* = new *Thickness*(4, 0, 4, 0)

            };

            return uiCurrComboBox;

        }

*Button* CreateMoreButton()

        {

*Button* currBut = new *Button*

            {

*Margin* = new *Thickness*(5, 0, 0, 0),

                Content = "•••",

*ToolTip* = "Більше...",

*FontSize* = 8,

*Height* = 14,

*BorderThickness* = new *Thickness*(0),

            };

            currBut.*ContextMenu* = uiCurrContextMenu;

            currBut.*Click* += new *RoutedEventHandler*(OnMoreBtnClick);

            return currBut;

        }

*ContextMenu* CreateContextMenuForMoreButton()

        {

*ContextMenu* currContextMenu = new *ContextMenu*

            {

*Margin* = new *Thickness*(5, 3, 0, 3),

            };

            return currContextMenu;

        }

        private void ShowWordQuotesWindow()

        {

            ClassifierForWord LastClass = ClassInLine;

            while (LastClass.ChildClassifiers.*Count* > 0)

                LastClass = LastClass.ChildClassifiers[0];

*List*<Quote> wordQuotes = uiMainWindow.DB.GetQuotes(uiMainWindow.currentSelectedWord.nID, LastClass.Id);

            WordQuotesWindow dialog = new WordQuotesWindow(uiMainWindow, uiMainWindow.currentSelectedWord.nID, LastClass.Id);

            dialog.*ShowDialog*();

        }

        private void OnSizeChanged(object sender, *EventArgs* e)

        {

            uiMainBorder.*Width* = uiMainWindow.*ActualWidth* - 100;

        }

        private void OnMoreBtnClick(object sender, *EventArgs* e)

        {

*Button* uiCurrButt = (*Button*)sender;

            uiCurrButt.*ContextMenu*.*IsOpen* = true;

        }

        private void OnDeleteClick(object sender, *EventArgs* e)

        {

            ClassifierForWord LastClass = ClassInLine;

            while (LastClass.ChildClassifiers.*Count* > 0)

                LastClass = LastClass.ChildClassifiers[0];

            uiMainWindow.DB.DeleteWordFromClassifier(uiMainWindow.currentSelectedWord.nID, LastClass.Id);

            Border ParentInStP = (Border)uiInsideStackPanel.*Parent*;

*StackPanel* StP = (*StackPanel*)ParentInStP.*Parent*;

            StP.*Children*.*Remove*(ParentInStP);

        }

        private void OnEditClick(object sender, *EventArgs* e)

        {

            OnMouseUp(sender, e);

        }

        private void OnAddWordQuotesClick(object sender, *EventArgs* e)

        {

            ClassifierForWord LastClass = ClassInLine;

            while (LastClass.ChildClassifiers.*Count* > 0)

                LastClass = LastClass.ChildClassifiers[0];

            uiMainWindow.ShowAddWordQuotesWindow(uiMainWindow.currentSelectedWord.nID, LastClass.Id);

        }

        private void OnWordQuotesClick(object sender, *EventArgs* e)

        {

            ShowWordQuotesWindow();

        }

        private void OnDropDownOpened(object sender, *EventArgs* e)

        {

*ComboBox* uiCurrComboBox = (*ComboBox*)sender;

            uiCurrComboBox.*IsDropDownOpen* = false;

        }

        private void OnMouseUp(object sender, *EventArgs* e)

        {

            if (ClassInLine != null && ViewerController != null)

            {

                if (uiMainWindow.ClassViewerListOfClassLines.bNewClassifAddings)

                {

                    uiMainWindow.ClassViewerListOfClassLines.EndAddingNewClass();

                    uiMainWindow.UpdateWordPanel();

                    return;

                }

                if (uiMainWindow.SelectedLine != null)

                    uiMainWindow.SelectedLine.SetState(Enums.UIObjectState.NORMAL);

                uiMainWindow.SelectedLine = this;

                ViewerController.SelectLineOfClassifiers(ClassInLine);

                uiMainWindow.uiTxtBoxEdClassifier.*Text* = ClassInLine.Name;

                uiMainWindow.uiBtnSaveClass.*Visibility* = *Visibility*.*Visible*;

                uiMainWindow.uiBtnCancelClass.*Visibility* = *Visibility*.*Visible*;

                SetState(Enums.UIObjectState.SELECTED);

            }

        }

        private void OnMouseEnter(object sender, *EventArgs* e)

        {

            if (!CurrentState.*Equals*(Enums.UIObjectState.SELECTED))

                SetState(Enums.UIObjectState.HOWERED);

        }

        private void OnMouseLeave(object sender, *EventArgs* e)

        {

            if (!CurrentState.*Equals*(Enums.UIObjectState.SELECTED))

                SetState(Enums.UIObjectState.NORMAL);

        }

    }

    /// <summary>

    /// Classificatory Viewer List Controller

    /// </summary>

    public class ClWordClassLinesController

    {

        public ClWordClassLinesController(ref DataController DB, MainWindow CurrMainWindow)

        {

            DataBase = DB;

            param = new CurrParams();

            uiMainWindow = CurrMainWindow;

            CurrentWord = new Word("");

            uiMainStackPanel = CreateMainStackPanel();

        }

        Word CurrentWord;

        DataController DataBase;

        CurrParams param;

        ClViewerControler ViewerController;

        MainWindow uiMainWindow;

        ClLineController SelectedLine;

        public bool bNewClassifAddings = false;

*StackPanel* uiMainStackPanel;

        public *StackPanel* MainStackPanel

        {

            get { return this.uiMainStackPanel; }

        }

        public void EndAddingNewClass()

        {

            bNewClassifAddings = false;

            uiMainWindow.uiBtnSaveClass.*Visibility* = *Visibility*.*Collapsed*;

            uiMainWindow.uiBtnCancelClass.*Visibility* = *Visibility*.*Collapsed*;

        }

        public void SetNewWord(Word word)

        {

            CurrentWord = word;

            UpdateStackPanel();

        }

        public void UpdateStackPanel()

        {

            MainStackPanel.*Children*.Clear();

            CurrentWord.ListOfClassifiers = DataBase.GetWordParents(CurrentWord.nID);

*List*<ClassifierForWord> tempList = DataBase.GetWordParents(CurrentWord.nID, false);

*List*<ClassifierForWord> root = DataBase.GetRootClassifiers();

            for (int i = 0; i < root.*Count*; i++)

            {

*Expander* uiCurrExpander = GetExpandeOfRootClassifierForWord(root[i]);

                var classif = CurrentWord.ListOfClassifiers.*Find*(x => *x*.Id == root[i].Id);

                if (classif != null)

                    FillExpandeOfRootClassifierForWord(uiCurrExpander, tempList);

                MainStackPanel.*Children*.*Add*(uiCurrExpander);

            }

        }

        /// <summary>

        /// Controller of referenced view of classifiers for automatically selecting current list of classifiers

        /// </summary>

        /// <param name="CtrlView"></param>

        public void SetRefViewerControler(ClViewerControler CtrlView)

        {

            ViewerController = CtrlView;

        }

*StackPanel* CreateMainStackPanel()

        {

*StackPanel* currStackPanel = new *StackPanel*

            {

                Name = "uiStackPanelWordClassExp",

*Orientation* = *Orientation*.*Vertical*,

*HorizontalAlignment* = *HorizontalAlignment*.*Stretch*,

*VerticalAlignment* = *VerticalAlignment*.*Stretch*,

*Margin* = new *Thickness*(0, 2, 0, 0)

            };

            return currStackPanel;

        }

*Button* CreatePlusButton()

        {

*Button* currBut = new *Button*

            {

                Content = "+",

*ToolTip* = "Додати нову ознаку",

*FontSize* = 15,

*Padding* = new *Thickness*(0, -4, 0,0),

*Margin* = new *Thickness*(10,0,0,0),

*Background* = new *SolidColorBrush*(*Colors*.*AliceBlue*),

*Foreground* = new *SolidColorBrush*(*Colors*.*ForestGreen*),

*FontWeight* = *FontWeights*.*Bold*,

*Height* = 15,

*Width* = 15,

*BorderThickness* = new *Thickness*(0),

            };

            currBut.*Click* += new *RoutedEventHandler*(OnButtonPlusClick);

            return currBut;

        }

*Expander* GetExpandeOfRootClassifierForWord(ClassifierForWord currClass)

        {

            //expander header text

*TextBlock* TxtHeader = new *TextBlock*

            {

*Text* = currClass.Name

            };

            //expander plus button

*Button* uiHeaderPlusButton = CreatePlusButton();

            //expander header

*StackPanel* uiStackPanelHeader = new *StackPanel*

            {

*Orientation* = *Orientation*.*Horizontal*

            };

            uiStackPanelHeader.*Children*.*Add*(TxtHeader);

            uiStackPanelHeader.*Children*.*Add*(uiHeaderPlusButton);

*StackPanel* uiStackPanel = new *StackPanel*

            {

*Margin* = new *Thickness*(15, 0, 5, 0),

*Orientation* = *Orientation*.*Vertical*

            };

*Expander* uiExpander = new *Expander*

            {

*Margin* = new *Thickness*(5, 0, 5, 0),

                Name = "uiExpRootClassId\_" + currClass.Id.*ToString*(),

*Header* = uiStackPanelHeader,

*HorizontalAlignment* = *HorizontalAlignment*.*Stretch*,

*FlowDirection* = *FlowDirection*.*LeftToRight*,

                Content = uiStackPanel,

*IsExpanded* = true

            };

            return uiExpander;

        }

        void FillExpandeOfRootClassifierForWord(*Expander* uiExpander, *List*<ClassifierForWord> listOfListsClass)

        {

*StackPanel* uiStackPanel = (*StackPanel*)uiExpander.Content;

            string[] Ids = uiExpander.Name.*Split*('\_');

            int ExpId = *Convert*.*ToInt32*(Ids[1]);

            for (int i = 0; i < listOfListsClass.*Count*; i++)

            {

                if (listOfListsClass[i].Id == ExpId)

                {

                    //new line controller

                    ClLineController CurrLine = new ClLineController(listOfListsClass[i], i, uiMainWindow);

                    //viewer controller

                    if (ViewerController != null)

                        CurrLine.SetRefViewerControler(ViewerController);

                    uiStackPanel.*Children*.*Add*(CurrLine.MainStackPanel);

                }

            }

        }

        private void OnButtonPlusClick(object sender, *EventArgs* e)

        {

            if (bNewClassifAddings)

                return;

*StackPanel* uiParentStackPanel = ((*StackPanel*)((*Button*)sender).*Parent*);

*Expander* uiParentExpander = (*Expander*)(uiParentStackPanel.*Parent*);

*StackPanel* uiStackPanel = (*StackPanel*)uiParentExpander.*Content*;

            string[] Ids = uiParentExpander.*Name*.*Split*('\_');

            int ClassifierId = *Convert*.*ToInt32*(Ids[1]);

            ClassifierForWord CurrClass = DataBase.GetClassifier(ClassifierId);

            SelectedLine = new ClLineController(new *List*<string>(), uiStackPanel.*Children*.Count, uiMainWindow, false);

            SelectedLine.MainBorder.*Width* = 200;

            //viewer controller

            if (ViewerController != null)

                SelectedLine.SetRefViewerControler(ViewerController);

            //select line in viewer

            ViewerController.SelectLineOfClassifiers(CurrClass);

            if (ViewerController.IsSelectedHasNoChildrens())

            {

*MessageBoxResult* result = *MessageBox*.*Show*("Відсутні внутрішні ознаки. Додати нову?", "Підтвердження", *MessageBoxButton*.YesNo, MessageBoxImage.Question);

                if (result == *MessageBoxResult*.*Yes*)

                    uiMainWindow.ShowEditClassDialog();

                return;

            }

            uiStackPanel.*Children*.*Add*(SelectedLine.MainStackPanel);

            SelectedLine.SetState(Enums.UIObjectState.SELECTED);

            uiMainWindow.uiTxtBoxEdClassifier.*Text* = CurrClass.Name;

            uiMainWindow.uiBtnSaveClass.*Visibility* = *Visibility*.*Visible*;

            uiMainWindow.uiBtnCancelClass.*Visibility* = *Visibility*.*Visible*;

            //if something already selected un select it

            if(uiMainWindow.SelectedLine != null)

            {

                uiMainWindow.SelectedLine.SetState(Enums.UIObjectState.NORMAL);

                uiMainWindow.SelectedLine = null;

            }

            bNewClassifAddings = true;

        }

    }

}

ClassViewerListStackPanel.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

using *System*.*Windows*;

using *System*.*Windows*.*Controls*;

using *System*.*Windows*.*Controls*.*Primitives*;

using *System*.*Windows*.*Data*;

using *System*.*Windows*.*Documents*;

using *System*.*Windows*.*Input*;

using *System*.*Windows*.*Media*;

using *System*.*Windows*.*Media*.*Imaging*;

using *System*.*Windows*.*Navigation*;

using *System*.*Windows*.*Shapes*;

using *System*.*IO*;

using *System*.*Windows*.*Markup*;

using VerbariumVocabulary.Workspace;

namespace VerbariumVocabulary

{

    /// <summary>

    /// Current Parameters for controller

    /// </summary>

    class CurrParams

    {

        public CurrParams()

        {

            ListChosenClassifiers = new *List*<ClassifierForWord>();

        }

        public void Reset()

        {

            ListChosenClassifiers.Clear();

            CountOfOpenedListBoxes = 0;

        }

        public int LastChosenClassId()

        {

           return ListChosenClassifiers[ListChosenClassifiers.Count - 1].Id;

        }

        public int CountOfOpenedListBoxes = 0;

        public *List*<ClassifierForWord> ListChosenClassifiers;

    }

    /// <summary>

    /// Classificatory Viewer List Controller

    /// </summary>

    public class ClViewerControler

    {

        public ClViewerControler(ref DataController DB, MainWindow CurrMainWindow, bool HideFirst = true)

        {

            DataBase = DB;

            param = new CurrParams();

            bHideFirstBox = HideFirst;

            uiMainWindow = CurrMainWindow;

            uiMainStackPanel = CreateMainStackPanel();

            CreateAndFillNewListBox();

        }

        DataController DataBase;

        CurrParams param;

        MainWindow uiMainWindow;

*StackPanel* uiMainStackPanel;

        bool bHideFirstBox = true;

        public *StackPanel* MainStackPanel

        {

            get { return this.uiMainStackPanel; }

        }

        public *List*<ClassifierForWord> GetListOfSelected()

        {

            return param.ListChosenClassifiers;

        }

        public bool IsSelectedHasNoChildrens()

        {

            int id = param.ListChosenClassifiers[0].Id;

            return DataBase.GetClassifier(id).ChildClassifiers.Count > 0;

        }

        public void UpdateStackPanel()

        {

            Clear();

            CreateAndFillNewListBox();

        }

        public void Clear()

        {

            param.Reset();

            uiMainStackPanel.*Children*.*Clear*();

        }

        void CreateAndFillNewListBox(int ClassID = 0)

        {

*StackPanel* uiCurrStackPanel = new *StackPanel*

            {

*Orientation* = *Orientation*.*Vertical*

            };

*List*<ClassifierForWord> LstOfCurrClassifiers;

            if (ClassID == 0)

            {

                LstOfCurrClassifiers = DataBase.GetRootClassifiers();

            }

            else

            {

                LstOfCurrClassifiers = DataBase.GetAllClassifiers(ClassID);

            }

            if (LstOfCurrClassifiers.*Count* < 1)

                return;

            string sCount = param.CountOfOpenedListBoxes.*ToString*();

*ListBox* uiCurrListBox = CreateListBoxForClassifiers(sCount);

            sCount = (param.CountOfOpenedListBoxes - 1).*ToString*();

*ComboBox* uiCbBox = CreateComboBox(sCount);

            uiCurrStackPanel.*Children*.*Add*(uiCbBox);

            uiCurrStackPanel.*Children*.*Add*(uiCurrListBox);

            if (param.CountOfOpenedListBoxes.*ToString*().*Equals*("1") && bHideFirstBox)

                uiCurrStackPanel.*Visibility* = *Visibility*.*Collapsed*;

*List* <string> CbList = new *List*<string>();

            foreach(ClassifierForWord var in LstOfCurrClassifiers)

            {

                CbList.*Add*(var.Name);

            }

            uiCbBox.*ItemsSource* = CbList;

            //Filling

            foreach (ClassifierForWord cl in LstOfCurrClassifiers)

            {

*ListBoxItem* uiListItem = CreateListBoxItem(cl);

                uiCurrListBox.*Items*.*Add*(uiListItem);

            }

            uiMainStackPanel.*Children*.*Add*(uiCurrStackPanel);

        }

        void DeleteNextListBoxes(int After)

        {

            for (int i = uiMainStackPanel.*Children*.*Count* - 1; i > After - 1; i--)

            {

                if(i > After)

                {

                    uiMainStackPanel.*Children*.*RemoveAt*(i);

                    param.CountOfOpenedListBoxes--;

                }

                if (i < param.ListChosenClassifiers.Count)

                    param.ListChosenClassifiers.*RemoveAt*(i);

            }

        }

        public void SelectLineOfClassifiers(ClassifierForWord ClassifForSel)

        {

            for(int j = 0; j < uiMainStackPanel.*Children*.*Count*; j++)

            {

                bool bIsSelected = false;

                if (ClassifForSel != null)

                {

*StackPanel* uiCurrStackPanel = (*StackPanel*)uiMainStackPanel.*Children*[j];

*ListBox* uiCurrListBox = (*ListBox*)uiCurrStackPanel.*Children*[1];

                    for (int i = 0; i < uiCurrListBox.*Items*.Count; i++)

                    {

*ListBoxItem* uiCurrLBI = (*ListBoxItem*)uiCurrListBox.*Items*[i];

                        string[] Ids = uiCurrLBI.Name.*Split*('\_');

                        int ClassifierId = *Convert*.*ToInt32*(Ids[1]);

                        if (ClassifForSel.Id.*Equals*(ClassifierId))//select

                        {

                            bIsSelected = true;

                            try

                            {

*ListBoxItem* item = (*ListBoxItem*)uiCurrListBox.*Items*[i];

                                uiCurrListBox.*SelectedItem* = item;

                            }

                            catch(*Exception* e) { }

                            uiCurrListBox.*ScrollIntoView*(uiCurrListBox.*SelectedItem*);

                            break;

                        }

                    }

                }

                if (ClassifForSel.ChildClassifiers.Count > 0)

                    ClassifForSel = ClassifForSel.ChildClassifiers[0];

                else

                    ClassifForSel = null;

                if (!bIsSelected || ClassifForSel == null)//if nothing has been selected stop

                    break;

            }

        }

*StackPanel* CreateMainStackPanel()

        {

*StackPanel* currStackPanel = new *StackPanel*

            {

                Name = "uiStackPaneListClassEditExp",

*Orientation* = *Orientation*.*Horizontal*,

*Margin* = new *Thickness*(0, 2, 0, 0),

            };

            return currStackPanel;

        }

*ListBox* CreateListBoxForClassifiers(string Id)

        {

            param.CountOfOpenedListBoxes++;

*Color* clr = (*Color*)*ColorConverter*.*ConvertFromString*("#FFCC66");

            clr.*A* = 40;

*ListBox* uiMainListBox = new *ListBox*

            {

*MinWidth* = 150,

*Margin* = new *Thickness*(5, 0, 5, 0),

*BorderBrush* = new *SolidColorBrush*(clr),

                Name = "uiListBoxClassId\_" + Id,

*Height* = 85

            };

            return uiMainListBox;

        }

*ListBoxItem* CreateListBoxItem(ClassifierForWord currClass)

        {

*ListBoxItem* uiCurrListItem = new *ListBoxItem*

            {

                Content = currClass.Name,

                Name = "uiLstItemClassId\_" + currClass.Id.*ToString*(),

            };

            uiCurrListItem.*Selected* += new *RoutedEventHandler*(OnListItemClick);

            return uiCurrListItem;

        }

*ComboBox* CreateComboBox(string Id)

        {

*ComboBox* uiCurrComboBox = new *ComboBox*

            {

*Text* = "Введіть ознаку",

                Name = "uiComboBoxViewClassId\_" + Id,

*MinWidth* = 150,

*FontSize* = 12,

*IsEditable* = true,

*Margin* = new *Thickness*(-4,0,-4,0)

            };

*Style* style = uiMainWindow.*FindResource*("ComboBox1") as *Style*;

            uiCurrComboBox.*Style* = style;

            uiCurrComboBox.*DropDownOpened* += new *EventHandler*(OnDropDownOpened);

*RoutedEventHandler* rteh = new *RoutedEventHandler*(OnTextChanged);

            uiCurrComboBox.*AddHandler*(*TextBoxBase*.*TextChangedEvent*, rteh);

            return uiCurrComboBox;

        }

        private void OnListItemClick(object sender, *RoutedEventArgs* e)

        {

*ListBoxItem* uiCurrListBoxItem = (*ListBoxItem*)sender;

            string[] Ids = uiCurrListBoxItem.Name.*Split*('\_');

            int ClassifierId = *Convert*.*ToInt32*(Ids[1]);

*ListBox* CurrListBox = (*ListBox*)uiCurrListBoxItem.*Parent*;

            string[] sID = CurrListBox.Name.*Split*('\_');

            int nListBoxID = *Convert*.*ToInt32*(sID[1]);

            DeleteNextListBoxes(nListBoxID);

            CreateAndFillNewListBox(ClassifierId);

            ClassifierForWord SelClass = DataBase.GetClassifier(ClassifierId);

            param.ListChosenClassifiers.*Add*(SelClass);

*ListBox* uiParentListBox = (*ListBox*)uiCurrListBoxItem.*Parent*;

*StackPanel* uiParentStPanel = (*StackPanel*)uiParentListBox.*Parent*;

            ((*ComboBox*)uiParentStPanel.*Children*[0]).*Text* = SelClass.Name;

        }

        private void OnDropDownOpened(object sender, *EventArgs* e)

        {

*ComboBox* uiCurrComboBox = (*ComboBox*)sender;

            uiCurrComboBox.*IsDropDownOpen* = false;

*ListBox* uiCurrListBox = (*ListBox*)uiCurrComboBox.*Parent*;

            for (int i = 1; i < uiCurrListBox.*Items*.Count; i++)

            {

                string content = ((*ListBoxItem*)uiCurrListBox.*Items*[i]).Content.*ToString*();

                if (content.*Equals*(uiCurrComboBox.*Text*))

                {

                    uiCurrListBox.*SelectedItem* = ((*ListBoxItem*)uiCurrListBox.*Items*[i]);

                    uiCurrListBox.*ScrollIntoView*(uiCurrListBox.*SelectedItem*);

                }

            }

        }

        private void OnTextChanged(object sender, *EventArgs* e)

        {

*ComboBox* uiCurrComboBox = (*ComboBox*)sender;

*StackPanel* uiCurrStackPanel = (*StackPanel*)uiCurrComboBox.*Parent*;

*ListBox* uiCurrListBox = (*ListBox*)uiCurrStackPanel.*Children*[1];

            bool Found = false;

            for (int i = 0; i < uiCurrListBox.*Items*.Count; i++)

            {

                string content = ((*ListBoxItem*)uiCurrListBox.*Items*[i]).Content.*ToString*();

                if (content.*Equals*(uiCurrComboBox.*Text*))

                {

                    uiCurrListBox.*SelectedItem* = ((*ListBoxItem*)uiCurrListBox.*Items*[i]);

                    uiCurrListBox.*ScrollIntoView*(uiCurrListBox.*SelectedItem*);

                    Found = true;

                }

            }

        }

    }

}

DictionaryTab.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

using *System*.*Windows*;

using *System*.*Windows*.*Controls*;

using *System*.*Windows*.*Data*;

using *System*.*Windows*.*Documents*;

using *System*.*Windows*.*Input*;

using *System*.*Windows*.*Media*;

using *System*.*Windows*.*Media*.*Imaging*;

using *System*.*Windows*.*Navigation*;

using *System*.*Windows*.*Shapes*;

using *System*.*IO*;

using *System*.*Windows*.*Markup*;

using *Microsoft*.*Win32*;

namespace VerbariumVocabulary

{

    /// <summary>

    /// Interaction logic for Dictionary Tab

    /// </summary>

    public partial class MainWindow : *Window*

    {

        private void InitAllWords(*StackPanel* CurrStackPanel, *List*<Word> CurrWordList)

        {

            CurrStackPanel.*Children*.*Clear*();

            for (int i = 0; i < CurrWordList.Count; i++)

                CurrStackPanel.*Children*.*Add*(GetWordButton(CurrWordList[i]));

        }

        private void InitAlphabetPanel()

        {

            for (int i = 0; i < sAlphabet.*Length*; i++)

                uiAlphabetWrapPanel.*Children*.*Add*(GetAlphabetLetterButton(sAlphabet[i].*ToString*()));

        }

    }

}

MainTab.cs

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.Windows;

namespace VerbariumVocabulary

{

    /// <summary>

    /// Interaction logic for MainWindow.xaml

    /// </summary>

    public partial class MainWindow : Window

    {

        private void InitLastWords()

        {

            uiLastWordsStackPanel.Children.Clear();

            for (int i = listOfWords.Count - 1; i > listOfWords.Count - 7 && i >= 0; i--)

                uiLastWordsStackPanel.Children.Add(GetWordButton(listOfWords[i]));

        }

        private void ShowPropDialog()

        {

            PrefsWindow dialog = new PrefsWindow(ref DB);

            dialog.ShowDialog();

            listOfAllClassifiers = DB.GetAllClassifiers();

            InitAllClassifiers();

            InitAllWords(uiAllWordsStackPanel, listOfWords);

            InitLastWords();

        }

        public void ShowEditClassDialog()

        {

            EditClassifiersWindow dialog = new EditClassifiersWindow(ref DB, this);

            dialog.ShowDialog();

        }

        private void OnBtn\_Prefs\_Click(object sender, RoutedEventArgs e)

        {

            ShowPropDialog();

        }

        private void OnBtn\_EditClass\_Click(object sender, RoutedEventArgs e)

        {

            ShowEditClassDialog();

        }

        private void OnBtn\_AddNewWord\_MouseClick(object sender, RoutedEventArgs e)

        {

            uiMainTabControl.SelectedItem = uiAddNewWordsTabItem;

        }

        private void OnMenuItem\_ImprtFrTxt\_Click(object sender, RoutedEventArgs e)

        {

            System.Windows.MessageBox.Show("Слова у файлі повинні бути розділені одним з наступних символів: '.' , ';' , ':' , пробілом, символом нового рядка або символом табуляції!");

            OpenFileDialog openFileDialog = new OpenFileDialog();

            openFileDialog.Filter = "Текстові файли (\*.txt)|\*.txt";

            if (openFileDialog.ShowDialog() == true)

            {

                uiMainTabControl.SelectedItem = uiAddNewWordsTabItem;

                try

                {

                    List<string> listOfNewWords = GetWordsFromTxtFile(openFileDialog.FileName);

                    FillNewWordsPanel(listOfNewWords);

                }

                catch (Exception)

                {

                }

            }

        }

        private void OnBtn\_EditCurrWord\_Click(object sender, RoutedEventArgs e)

        {

            uiMainTabControl.SelectedItem = uiWordTabItem;

            uiBtnWordEdit.Content = "режим перегляду";

            uiMainWordTextBox.IsEnabled = true;

            isWordInEditMode = true;

            //RedrawWordClassStackPanel();

            //AddComboBoxForEachListInStackPanelWithClass();

        }

    }

}

MainWindowDataHelper.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*IO*;

using *System*.*Windows*;

using VerbariumVocabulary.Workspace;

namespace VerbariumVocabulary

{

    public partial class MainWindow

    {

        private void InitDatabase()

        {

            DB = new DataController();

        }

        private void InitStartVar()

        {

            listOfWords = DB.GetAllWords();

        }

        private void InitAllClassifiers()

        {

            listOfAllClassifiers = DB.GetAllClassifiers();

            listOfRootClassifiers = DB.GetAllClassifiers();

            listOfListsOfAllCurrentClassifiers.*Add*(listOfRootClassifiers);

        }

        //data base

        public DataController DB;

        //current data variables

        public List<string> listOfLastAddedWords = new List<string>();

        public List<ClassifierForWord> listOfAllClassifiers = new List<ClassifierForWord>();

        public List<ClassifierForWord> listOfRootClassifiers = new List<ClassifierForWord>();

        public List<Word> listOfWords = new List<Word>();

        public Word currentSelectedWord = new Word("СЛОВО");

        //for classifier of current word

        public List<List<ClassifierForWord>> listOfListsOfCurrentClassifiers = new List<List<ClassifierForWord>>();

        public List<ClassifierForWord> listOfCurrentClassifiers = new List<ClassifierForWord>();

        public ClassifierForWord currClassifier = new ClassifierForWord();

        //for panel of all classifiers

        public List<List<ClassifierForWord>> listOfListsOfAllCurrentClassifiers = new List<List<ClassifierForWord>>();

        public List<ClassifierForWord> listOfAllCurrentClassifiers = new List<ClassifierForWord>();

        public ClassifierForWord currClassifierInAllClass = new ClassifierForWord();

        //////////////////////////////////////////////////////

        //edit word mode

        public List<ClassifierForWord> listWordNewTempClassifier = new List<ClassifierForWord>();

        public bool isWordInEditMode = false;

        //static variables

        public string sAlphabet = "АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ";

        public void OpenClassifiersFile(ref List<ClassifierForWord> ListOfClassifiers, string sPath = "")

        {

            string FullTextFile = "";

            try

            {

                if (sPath.*Length* > 0)

                    if (*File*.*Exists*(sPath))

                    {

                        FullTextFile = *File*.*ReadAllText*(sPath);

                    }

                    else

                    {

                        return;

                    }

                else

                {

                    byte[] myFile = Properties.Resources.classifiers;

                    FullTextFile = *System*.*Text*.*Encoding*.*UTF8*.*GetString*(myFile);

                }

            }

            catch (*Exception*)

            {

*System*.*Windows*.*MessageBox*.*Show*("Файл неможливо прочитати!");

                return;

            }

            string[] stringArr1 = FullTextFile.*Split*('$');

            try

            {

                for (int i = 1; i < stringArr1.*Length*; i++)

                {

                    string[] stringArr2 = stringArr1[i].*Split*('\*');

                    ClassifierForWord tClassifier = new ClassifierForWord(*Convert*.*ToInt32*(stringArr2[0]), stringArr2[1]);

                    ListOfClassifiers.*Add*(tClassifier);

                }

            }

            catch (*Exception*)

            {

*System*.*Windows*.*MessageBox*.*Show*("Невірний файл!");

                return;

            }

        }

        public void OpenWordsFile(ref List<Word> ListOfWords, string sPath = "")

        {

            string FullTextFile = "";

            try

            {

                if (sPath.*Length* > 0)

                    if (*File*.*Exists*(sPath))

                    {

                        FullTextFile = *File*.*ReadAllText*(sPath);

                    }

                    else

                    {

                        return;

                    }

                else

                {

                    byte[] myFile = Properties.Resources.words;

                    FullTextFile = *System*.*Text*.*Encoding*.*UTF8*.*GetString*(myFile);

                }

            }

            catch (*Exception*)

            {

*MessageBox*.*Show*("Файл неможливо прочитати!");

                return;

            }

            string[] stringArr1 = FullTextFile.*Split*('#');

            try

            {

                for (int i = 1; i < stringArr1.*Length*; i++)

                {

                    string[] stringArr2 = stringArr1[i].*Split*('$');

                    Word tempWord = new Word(stringArr2[0].*Remove*(stringArr2[0].*Length* - 2, 2));

                    string[] stringSeparators = new string[] { "\r\n" };

                    for (int j = 1; j < stringArr2.*Length*; j++)

                    {

                        string[] stringArr3 = stringArr2[j].*Split*(stringSeparators, *System*.*StringSplitOptions*.*RemoveEmptyEntries*);

                        if (*Convert*.*ToInt32*(stringArr3[0]) <= listOfAllClassifiers.Count)

                            tempWord.ListOfClassifiers.*Add*(listOfAllClassifiers[*Convert*.*ToInt32*(stringArr3[0]) - 1]);

                        else

                            continue;

                        for (int g = 1; g < stringArr3.*Length*; g++)

                        {

                            AddClassifierToList(ref tempWord.ListOfClassifiers, stringArr3[g]);

                        }

                    }

                    ListOfWords.*Add*(tempWord);

                }

            }

            catch (*Exception*)

            {

*MessageBox*.*Show*("Невірний файл!");

                return;

            }

        }

        public void AddClassifierToList(ref List<ClassifierForWord> rClassifiers, string sClassifier)

        {

            if (sClassifier[0] == '\t')

                sClassifier = sClassifier.*Remove*(0, 1);

            if (sClassifier[0] == '\t' && rClassifiers.*Count* > 0 && rClassifiers[rClassifiers.*Count* - 1].ChildClassifiers.*Count* > 0)

            {

                AddClassifierToList(ref rClassifiers[rClassifiers.*Count* - 1].ChildClassifiers, sClassifier);

            }

            else

            {

                while (sClassifier[0] == '\t')

                    sClassifier = sClassifier.*Remove*(0, 1);

                rClassifiers[rClassifiers.*Count* - 1].ChildClassifiers.*Add*(new ClassifierForWord(0, sClassifier));

            }

        }

    }

}

MainWindowUIHelper.cs

using System.Collections.Generic;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Markup;

using System.Windows.Media;

namespace VerbariumVocabulary

{

    public partial class MainWindow

    {

        Button GetWordButton(Word currWord)

        {

            Button currBut = new Button

            {

                Margin = new Thickness(0, 3, 0, 3),

                Content = currWord.sWord,

                Name = "uiBtnWordId\_" + currWord.nID.ToString(),

            };

            Style style = this.FindResource("LinkButton") as Style;

            currBut.Style = style;

            currBut.Click += new RoutedEventHandler(OnBtn\_Word\_MouseClick);

            return currBut;

        }

        Button CreateDeleteButton()

        {

            Button currBut = new Button

            {

                Margin = new Thickness(6, 0, 0, 0),

                Content = "✘",

                ToolTip = "Видалити",

                FontSize = 11

            };

            Style style = FindResource("LinkButtonNoUnderline") as Style;

            currBut.Style = style;

            currBut.Click += new RoutedEventHandler(OnBtn\_CheckNoAddWord\_MouseClick);

            return currBut;

        }

        TextBlock GetTextBlockForClassStackListView(ClassifierForWord currClass, int nGrinColNum)

        {

            StringBuilder sb = new StringBuilder();

            sb.Append(@"<TextBlock xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation' Grid.Column='" + nGrinColNum.ToString() + "' TextTrimming='CharacterEllipsis'/>");

            TextBlock uiCurrTxtBox = (TextBlock)XamlReader.Parse(sb.ToString());

            uiCurrTxtBox.Text = currClass.Name;

            uiCurrTxtBox.Name = "uiClassTxtBlockId\_" + nGrinColNum.ToString();

            return uiCurrTxtBox;

        }

        Button GetAlphabetLetterButton(string sCurrStr)

        {

            Button currBut = new Button

            {

                Margin = new Thickness(1, 1, 5, 1),

                Content = sCurrStr,

                Width = 10

            };

            Style style = this.FindResource("LinkButton") as Style;

            currBut.Style = style;

            currBut.Click += new RoutedEventHandler(OnBtn\_Alphabet\_MouseClick);

            return currBut;

        }

        public static ListBox GetListBoxForClassifiers(string sIDNumber)

        {

            Color clr = (Color)ColorConverter.ConvertFromString("#FFCC66");

            clr.A = 40;

            ListBox uiMainListBox = new ListBox

            {

                MinWidth = 130,

                Margin = new Thickness(5, 0, 5, 0),

                BorderBrush = new SolidColorBrush(clr),

                Name = "uiLstBoxClassId\_" + sIDNumber,

            };

            return uiMainListBox;

        }

        public static ListBox GetListBoxForEditExpClassifiers(string sIDNumber)

        {

            Color clr = (Color)ColorConverter.ConvertFromString("#FFC5C5C5");

            clr.A = 40;

            ListBox uiMainListBox = new ListBox

            {

                MinWidth = 130,

                Margin = new Thickness(5, 0, 5, 0),

                BorderBrush = new SolidColorBrush(clr),

                Name = "uiLstBoxClassifEditExp\_" + sIDNumber,

            };

            if (sIDNumber.Equals("0"))

            {

                uiMainListBox.Background = new SolidColorBrush(clr);

            }

            return uiMainListBox;

        }

        Expander GetExpandeOfRootClassifierForWord(string sIDNumber, ClassifierForWord currClass, List<ClassifierForWord> listOfListsClass)

        {

            Color clr = (Color)ColorConverter.ConvertFromString("#FFCC66");

            clr.A = 40;

            StackPanel uiStackPanel = new StackPanel

            {

                Margin = new Thickness(15, 0, 5, 0),

                Orientation = Orientation.Vertical

            };

            foreach (ClassifierForWord temp in listOfListsClass)

            {

                if (temp.Id == currClass.Id)

                {

                    uiStackPanel.Children.Add(GetStackPanelListOfCurrClassif(temp));

                }

            }

            Expander uiExpander = new Expander

            {

                Margin = new Thickness(5, 0, 5, 0),

                BorderBrush = new SolidColorBrush(clr),

                Name = "uiExpRootClassId\_" + sIDNumber,

                Header = currClass.Name,

                HorizontalAlignment = HorizontalAlignment.Left,

                FlowDirection = FlowDirection.LeftToRight,

                Content = uiStackPanel

            };

            return uiExpander;

        }

        StackPanel GetStackPanelListOfCurrClassif(ClassifierForWord currClass)

        {

            Color clrBG = Colors.White;

            clrBG.A = 100;

            StackPanel uiCurrStPanel = new StackPanel

            {

                Orientation = Orientation.Horizontal,

                Background = new SolidColorBrush(clrBG),

                HorizontalAlignment = HorizontalAlignment.Stretch

            };

            while (currClass.ChildClassifiers.Count > 0)

            {

                Color txtBackGClr = new Color();

                txtBackGClr.A = 0;

                TextBox uiTextBoxSign = new TextBox

                {

                    Margin = new Thickness(5, 0, 0, 0),

                    Text = ">",

                    IsEnabled = false,

                    IsReadOnly = true,

                    Background = new SolidColorBrush(txtBackGClr),

                    BorderBrush = new SolidColorBrush(txtBackGClr)

                };

                uiCurrStPanel.Children.Add(uiTextBoxSign);

                TextBox uiTextBox = new TextBox

                {

                    Margin = new Thickness(5, 0, 0, 0),

                    Text = currClass.ChildClassifiers[0].Name,

                    IsReadOnly = true,

                    Background = new SolidColorBrush(txtBackGClr),

                    BorderBrush = new SolidColorBrush(txtBackGClr)

                };

                uiCurrStPanel.Children.Add(uiTextBox);

                currClass = currClass.ChildClassifiers[0];

            }

            return uiCurrStPanel;

        }

        Grid GetNewWordGrid(string sContent = "")

        {

            StringBuilder sb = new StringBuilder();

            sb.Append(@"<Grid xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation' Height='23'>

                                        <Grid.ColumnDefinitions>

                                            <ColumnDefinition></ColumnDefinition>

                                            <ColumnDefinition Width='23'></ColumnDefinition>

                                            <ColumnDefinition Width='23'></ColumnDefinition>

                                        </Grid.ColumnDefinitions>

                                    </Grid>");

            Grid uiTempGrig = (Grid)XamlReader.Parse(sb.ToString());

            sb.Clear();

            sb.Append(@"<TextBox xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation' Grid.Column='0' Background='WhiteSmoke' BorderThickness='0'/> ");

            TextBox uiCurrTextBox = (TextBox)XamlReader.Parse(sb.ToString());

            uiCurrTextBox.KeyDown += OnTxtBox\_NewWord\_KeyDown;

            uiCurrTextBox.Text = sContent;

            uiTempGrig.Children.Add(uiCurrTextBox);

            sb.Clear();

            sb.Append(@"<Button xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation' Grid.Column='1' Height='23' BorderThickness='0' Content='✓' FontSize='18' Padding='1, -1, 1, 0' Foreground='#FF25C900'/>");

            Button uiButtonYes = (Button)XamlReader.Parse(sb.ToString());

            uiButtonYes.Click += new RoutedEventHandler(OnBtn\_CheckYesOnAddWord\_MouseClick);

            uiButtonYes.Background = Brushes.Transparent;

            uiTempGrig.Children.Add(uiButtonYes);

            sb.Clear();

            sb.Append(@"<Button xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation' Grid.Column='2' Height='23' BorderThickness='0' Content='✗' FontSize='18' Padding='1, -1, 1, 0' Foreground='#FFF91E08'/>");

            Button uiButtonNo = (Button)XamlReader.Parse(sb.ToString());

            uiButtonNo.Background = Brushes.Transparent;

            uiButtonNo.Click += new RoutedEventHandler(OnBtn\_CheckNoAddWord\_MouseClick);

            uiTempGrig.Children.Add(uiButtonNo);

            return uiTempGrig;

        }

    }

}

NewWordTab.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*IO*;

using *System*.*Windows*;

using *System*.*Windows*.*Controls*;

using *System*.*Windows*.*Input*;

namespace VerbariumVocabulary

{

    /// <summary>

    /// Interaction logic for Word Tab

    /// </summary>

    public partial class MainWindow : *Window*

    {

*List*<ClLineController> lNewWordsLines = new *List*<ClLineController>();

        string sAddNewWordsLog = "";

        bool bFromFile = false;

        private void EmptyListOfNewWords()

        {

            uiNewWordsStackPanel.*Children*.*RemoveRange*(1, uiNewWordsStackPanel.*Children*.*Count* - 1);

            lNewWordsLines.*Clear*();

            uiAddNewWordsTabHeaderText.*Text* = uiAddNewWordsTabHeaderText.*Text*.*Replace*("\*", "");

        }

        private *List*<string> GetWordsFromTxtFile(string sPath)

        {

            string sContent = *File*.*ReadAllText*(sPath);

            return GetWordsFromTxt(sContent);

        }

        private *List*<string> GetWordsFromTxt(string text)

        {

            string[] sArr = text.*Split*('\n', '\t', '\r', ';', ',', '.', ' ');

            return new *List*<string>(sArr);

        }

        private void FillNewWordsPanel(*List*<string> listOfWords)

        {

            bFromFile = true;

            foreach (string var in listOfWords)

                if (var.*Length* != 0)

                {

                    uiTxtBoxNewWord.*Text* = var;

                    OnBtn\_CheckYesOnAddWord\_MouseClick(uiButtonCheckYes, null);

                }

            bFromFile = false;

            if (sAddNewWordsLog.*Length* > 0)

            {

*System*.*Windows*.*MessageBox*.*Show*("Дані слова не були додані:\n" + sAddNewWordsLog);

                sAddNewWordsLog = "";

            }

        }

        private void OnBtn\_CheckYesOnAddWord\_MouseClick(object sender, *RoutedEventArgs* e)

        {

*Button* CurrButton = (*Button*)sender;

*Grid* ParentGrig = (*Grid*)CurrButton.*Parent*;

            TextBox CurrTxtBox = (TextBox)ParentGrig.*Children*[0];

            if (CurrTxtBox.*Text*.*Length* < 2)

            {

                if (!bFromFile)

*System*.*Windows*.*MessageBox*.*Show*("У слові повинно міститись більше однієї літери!");

                else

                    sAddNewWordsLog += CurrTxtBox.*Text* + " - У слові повинно міститись більше однієї літери!\n";

                return;

            }

            if (DB.isWordPresent(CurrTxtBox.*Text*) || (lNewWordsLines.*Find*(x => *x*.GetCurrListOfString()[0].*Equals*(CurrTxtBox.*Text*)) != null))

            {

                if (bFromFile)

                    sAddNewWordsLog += CurrTxtBox.*Text* + " - " + "Слово вже існує у словнику!\n";

                else

*System*.*Windows*.*MessageBox*.*Show*("Слово \"" + CurrTxtBox.Text + "\" вже існує у словнику!");

                return;

            }

            ClLineController CurrLine = new ClLineController(new *List*<string>() { CurrTxtBox.*Text* }, lNewWordsLines.*Count*, this, false);

            CurrLine.AddCustomControlToEnd(CreateDeleteButton());

            lNewWordsLines.*Add*(CurrLine);

            uiNewWordsStackPanel.*Children*.*Add*(CurrLine.MainStackPanel);

            CurrTxtBox.*Text* = "";

*FocusManager*.*SetFocusedElement*(this, uiTxtBoxNewWord);

            if (!uiAddNewWordsTabHeaderText.*Text*.*Contains*("\*"))

                uiAddNewWordsTabHeaderText.*Text* += "\*";

        }

        private void OnBtn\_CheckNoAddWord\_MouseClick(object sender, *RoutedEventArgs* e)

        {

*Button* CurrButton = (*Button*)sender;

*StackPanel* CurrStackPanel = (*StackPanel*)CurrButton.*Parent*;

            Border ParentBorder = (Border)CurrStackPanel.*Parent*;

*StackPanel* CStackPanel = (*StackPanel*)ParentBorder.*Parent*;

            uiNewWordsStackPanel.*Children*.*Remove*(CStackPanel);

            lNewWordsLines.*Remove*(lNewWordsLines.*Find*(x => *x*.MainStackPanel.*Equals*(CStackPanel)));

        }

        private void OnBtn\_SaveListOfNewWords\_Click(object sender, *RoutedEventArgs* e)

        {

*List*<ClassifierForWord> lst = new *List*<ClassifierForWord>();

            int id = 0;

            foreach (ClLineController cline in lNewWordsLines)

            {

                string str = cline.GetCurrListOfString()[0];

                DB.AddWord(str, lst, ref id);

            }

            listOfWords.Clear();

            listOfWords = DB.GetAllWords();

            InitAllWords(uiAllWordsStackPanel, listOfWords);

            InitLastWords();

            EmptyListOfNewWords();

        }

        private void OnBtn\_CancelListOfNewWords\_Click(object sender, *RoutedEventArgs* e)

        {

            EmptyListOfNewWords();

        }

        private void OnBtn\_FromBufferListOfNewWords\_Click(object sender, *RoutedEventArgs* e)

        {

*String* returnText = null;

            if (*Clipboard*.*ContainsText*(*TextDataFormat*.*UnicodeText*))

            {

                returnText = *Clipboard*.*GetText*(*TextDataFormat*.*UnicodeText*);

            }

*List*<string> listOfNewWords = GetWordsFromTxt(returnText);

            FillNewWordsPanel(listOfNewWords);

        }

    }

}

SearchTab.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Windows*;

using *System*.*Windows*.*Controls*;

namespace VerbariumVocabulary

{

    /// <summary>

    /// Interaction logic for Search Tab

    /// </summary>

    public partial class MainWindow : *Window*

    {

*StackPanel* uiSearchStackPanel;

        ClViewerControler ClassViewerSearchAdvanced;

        //ScrollViewer uiSearchScroll;

*List*<Stack<int>> arrClassifIdPaths = new *List*<Stack<int>>();

        private void RedrawAdvancedSearchStackPanel(*StackPanel* uiCurrStackPanel, *List*<ClassifierForWord> listOfClassif)

        {

            CreateSearchControls();

            if (uiSearchStackPanel == null)

            {

                uiSearchStackPanel = uiCurrStackPanel;

            }

            uiScrollClassifChoose.Content = ClassViewerSearchAdvanced.MainStackPanel;

            //uiCurrStackPanel.Children.Clear();

            //uiCurrStackPanel.Children.Add(btn\_AddSearchRule);

        }

        private void CreateSearchControls()

        {

            if (ClassViewerSearchAdvanced == null)

            {

                ClassViewerSearchAdvanced = new ClViewerControler(ref DB, this, false);

            }

        }

        private void OnBtn\_SearchByRules\_MouseClick(object sender, *RoutedEventArgs* e)

        {

            string sCondition = " AND ";

            var arrClassifToSearch = from cl in arrClassifIdPaths where cl.Count != 0 select cl.*Peek*();

*List* < Word > foundWords = DB.FindWords(arrClassifToSearch.*ToList*(), sCondition);

            uiFindedWordsStackPanel.*Children*.Clear();

            if (foundWords.Count == 0)

            {

*Label* noWords = new *Label*();

                noWords.Content = "Не знайдено слів";

                uiFindedWordsStackPanel.*Children*.*Add*(noWords);

                return;

            }

            foreach (Word word in foundWords)

            {

                uiFindedWordsStackPanel.*Children*.*Add*(GetWordButton(word));

            }

            try

            {

                uiBtnClearRulesId\_.*Visibility* = *Visibility*.*Collapsed*;

                uiBtnAcceptSearchRuleId\_.*Visibility* = *Visibility*.*Collapsed*;

            }

            catch (*Exception*)

            {

            }

        }

        private void OnBtn\_AddSearchRule\_MouseClick(object sender, *RoutedEventArgs* e)

        {

            uiBtnAddSearchRuleId\_.*Visibility* = *Visibility*.*Collapsed*;

            uiScrollClassifChoose.*Visibility* = *Visibility*.*Visible*;

            uiBtnAcceptSearchRuleId\_.*Visibility* = *Visibility*.*Visible*;

            if (arrClassifIdPaths.*Count* != 0)

            {

                uiBtnSearchByRulesId\_.*Visibility* = *Visibility*.*Visible*;

                uiBtnClearRulesId\_.*Visibility* = *Visibility*.*Visible*;

            }

        }

        private void OnBtn\_ClearSearchRules\_MouseClick(object sender, *RoutedEventArgs* e)

        {

            arrClassifIdPaths.*Clear*();

            uiStackPanelSearchCases.*Children*.*Clear*();

            uiBtnClearRulesId\_.*Visibility* = *Visibility*.*Collapsed*;

            uiBtnSearchByRulesId\_.*Visibility* = *Visibility*.*Collapsed*;

        }

        private void OnBtn\_AcceptSearchRule\_MouseClick(object sender, *RoutedEventArgs* e)

        {

            var newPath = new Stack<int>(from elem in ClassViewerSearchAdvanced.GetListOfSelected() select elem.Id);

            if (newPath.Count == 0)

                return;

            arrClassifIdPaths.*Add*(newPath);

            ClassViewerSearchAdvanced.UpdateStackPanel();

            ClWordClassLinesController line = new ClWordClassLinesController(ref DB, this);

*StackPanel* curPanel = line.MainStackPanel;

            curPanel.*Orientation* = *Orientation*.*Horizontal*;

*List*<string> arrClassifNames = new *List*<string>(from nIndex in newPath select DB.GetClassifier(nIndex).Name);

            ClLineController lineControl = new ClLineController(arrClassifNames, curPanel.*Children*.Count, this, false);

            //lineControl.AddCustomControlToEnd(GetDeletePathButton(uiStackPanelSearchCases.Children.Count));

            curPanel.*Children*.*Add*(lineControl.MainStackPanel);

            uiBtnAddSearchRuleId\_.*Visibility* = *Visibility*.*Visible*;

            uiBtnAcceptSearchRuleId\_.*Visibility* = *Visibility*.*Collapsed*;

            uiScrollClassifChoose.*Visibility* = *Visibility*.*Collapsed*;

            //curPanel.Children.Add(GetEditPathButton(uiStackPanelSearchCases.Children.Count));

            curPanel.*Children*.*Add*(GetDeletePathButton(uiStackPanelSearchCases.*Children*.*Count*));

            uiStackPanelSearchCases.*Children*.*Add*(curPanel);

            if (arrClassifIdPaths.*Count* != 0)

            {

                uiBtnSearchByRulesId\_.*Visibility* = *Visibility*.*Visible*;

                uiBtnClearRulesId\_.*Visibility* = *Visibility*.*Visible*;

            }

        }

        private *Button* GetEditPathButton(int nSearchIndex)

        {

*Button* CurrButton = new *Button*();

            CurrButton.Name = "EditPath\_" + nSearchIndex.*ToString*();

            CurrButton.Content = "✎";

            CurrButton.*ToolTip* = "Редагувати";

            CurrButton.*Height* = 18;

            CurrButton.*Width* = 18;

            CurrButton.*FontSize* = 10;

            CurrButton.*Click* += new *RoutedEventHandler*(OnBtn\_EditSearchCase\_MouseClick);

            return CurrButton;

        }

        private void OnBtn\_EditSearchCase\_MouseClick(object sender, *RoutedEventArgs* e)

        {

*Button* curBtn = (*Button*)sender;

            int nSearchIndex = *Convert*.*ToInt32*(curBtn.Name.*Split*('\_').*Last*());

            if (arrClassifIdPaths.*Count* > nSearchIndex && uiStackPanelSearchCases.*Children*.*Count* > nSearchIndex)

            {

                arrClassifIdPaths.*RemoveAt*(nSearchIndex);

                uiStackPanelSearchCases.*Children*.*RemoveAt*(nSearchIndex);

            }

        }

        private *Button* GetDeletePathButton(int nSearchIndex)

        {

*Button* CurrButton = new *Button*();

            CurrButton.Name = "DeletePath\_" + nSearchIndex.*ToString*();

            CurrButton.Content = "✕";

            CurrButton.*ToolTip* = "Видалити";

            CurrButton.*Height* = 15;

            CurrButton.*Width* = 15;

            CurrButton.*FontSize* = 5;

            CurrButton.*Margin* = new *Thickness*(5);

            CurrButton.*Click* += new *RoutedEventHandler*(OnBtn\_DeleteSearchCase\_MouseClick);

            return CurrButton;

        }

        private void OnBtn\_DeleteSearchCase\_MouseClick(object sender, *RoutedEventArgs* e)

        {

*Button* curBtn = (*Button*)sender;

            int nSearchIndex = *Convert*.*ToInt32*(curBtn.Name.*Split*('\_').*Last*());

            if (arrClassifIdPaths.*Count* > nSearchIndex && uiStackPanelSearchCases.*Children*.*Count* > nSearchIndex)

            {

                arrClassifIdPaths.*RemoveAt*(nSearchIndex);

                uiStackPanelSearchCases.*Children*.*RemoveAt*(nSearchIndex);

                if (arrClassifIdPaths.*Count* == 0)

                {

                    uiBtnSearchByRulesId\_.*Visibility* = *Visibility*.*Collapsed*;

                    uiBtnClearRulesId\_.*Visibility* = *Visibility*.*Collapsed*;

                }

            }

        }

    }

}

WordTab.cs

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

namespace VerbariumVocabulary

{

    /// <summary>

    /// Interaction logic for Word Tab

    /// </summary>

    public partial class MainWindow : Window

    {

        ClViewerControler ClassViewerForWordPanel;

        public ClWordClassLinesController ClassViewerListOfClassLines;

        public ClLineController SelectedLine;

        public void ShowAddWordQuotesWindow(int wordID, int classifID)

        {

            AddQuoteWindow dialog = new AddQuoteWindow(this, wordID, classifID);

            dialog.ShowDialog();

        }

        private void OnBtn\_WordDelete\_MouseClick(object sender, RoutedEventArgs e)

        {

            if (VerbariumVocabulary.Properties.Settings.Default.AskWhenWordIsDeleting)

            {

                MessageBoxResult result = MessageBox.Show("Видалити слово та його класифікатори?", "Підтвердження", MessageBoxButton.YesNo, MessageBoxImage.Question);

                if (result == MessageBoxResult.No)

                    return;

            }

            //uiStackPanelWordListClass.Children.Clear();

            SetWordViewToReadMode();

            DB.DeleteWord(currentSelectedWord.nID);

            listOfWords.Remove(currentSelectedWord);

            if (uiBtnWordEdit.IsEnabled)

                uiBtnWordEdit.IsEnabled = false;

            uiMainWordTextBox.Text = "ВИБЕРІТЬ СЛОВО";

            uiWordTabItem.IsEnabled = false;

            uiMainTabControl.SelectedItem = uiMainTabItem;

            InitAllWords(uiAllWordsStackPanel, listOfWords);

            InitLastWords();

        }

        private void OnBtn\_AddClassifierToWord\_MouseClick(object sender, RoutedEventArgs e)

        {

            List<ClassifierForWord> TempList = new List<ClassifierForWord>();

            List<ClassifierForWord> ListOfSel = ClassViewerForWordPanel.GetListOfSelected();

            if(ListOfSel.Count <= 1)

            {

                MessageBox.Show("Оберіть класифікатор!");

                return;

            }

            //if something already selected delete it

            if (SelectedLine != null)

            {

                ClassifierForWord currClass = SelectedLine.GetLastClassifier();

                if (currClass != null)

                    DB.DeleteWordFromClassifier(currentSelectedWord.nID, currClass.Id);

            }

            TempList.Add(ListOfSel[ListOfSel.Count - 1]);

            DB.AddParentsForWord(currentSelectedWord.nID, TempList);

            uiTxtBoxEdClassifier.Text = "";

            ClassViewerListOfClassLines.EndAddingNewClass();

            UpdateWordPanel();

        }

        private void OnBtn\_CancelAddClassifierToWord\_MouseClick(object sender, RoutedEventArgs e)

        {

            ClassViewerForWordPanel.Clear();

            ClassViewerListOfClassLines.EndAddingNewClass();

            uiTxtBoxEdClassifier.Text = "";

            //if something already selected un select it

            if (SelectedLine != null)

            {

                SelectedLine.SetState(Enums.UIObjectState.NORMAL);

                SelectedLine = null;

            }

            UpdateWordPanel();

        }

        private void SetWordViewToReadMode()

        {

            isWordInEditMode = false;

            uiMainWordTextBox.IsEnabled = false;

            uiBtnWordEdit.Content = "🖉";

            uiBtnWordCancel.Visibility = Visibility.Hidden;

            uiWordTabHeaderText.Text = uiWordTabHeaderText.Text.Replace("\*", "");

        }

        public void UpdateWordPanel()

        {

            //uiStackPanelWordListClass.Children.Clear();

            if (currentSelectedWord == null)

                return;

            ClassViewerListOfClassLines.SetNewWord(currentSelectedWord);

            List<ClassifierForWord> tempList = DB.GetWordParents(currentSelectedWord.nID, false);

            for (int i = 0; i < currentSelectedWord.ListOfClassifiers.Count; i++)

            {

                Expander uiCurrExpander = GetExpandeOfRootClassifierForWord(i.ToString(), currentSelectedWord.ListOfClassifiers[i], tempList);

                //uiStackPanelWordListClass.Children.Add(uiCurrExpander);

            }

            ClassViewerForWordPanel.UpdateStackPanel();

        }

    }

}

# Додаток В

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
| Виконав: | Стефанців М.І. |  |  | UML діаграма класів | Сторінка 1 з 1 |
| Перевірив: | Квятковський Б.О. |  |  |
| ІКНІ кафедра ПЗ ПІ-41 | | | |