Зміст

[Анотація 5](#_Toc530837744)

[Abstract 6](#_Toc530837745)

[Вступ 7](#_Toc530837746)

[Розділ 1. Огляд електронних словників та інструментів для їх побудови 9](#_Toc530837747)

[1.1. Поняття електронного словника 9](#_Toc530837748)

[1.1.1. Сучасна лексикографія: можливості електронних словників 9](#_Toc530837749)

[1.2 Алгоритми опрацювання тексту 13](#_Toc530837750)

[1.2.1. Алгоритми морфологічного аналізу 14](#_Toc530837751)

[1.2.2. Динамічні та статичні алгоритми аналізу тексту 15](#_Toc530837752)

[Розділ 2. Постановка задачі розробки програмного продукту для формування лексико-семантичного словника української мови 18](#_Toc530837753)

[2.1. Загальна постановка задачі 18](#_Toc530837754)

[2.2. Вибір інструментальних засобів 18](#_Toc530837755)

[2.3. Специфікація вимог до програмного продукту 19](#_Toc530837756)

[2.3.1. Вступ 19](#_Toc530837757)

[2.3.1.1. Призначення, мета 19](#_Toc530837758)

[2.3.1.2. Продукти аналоги 19](#_Toc530837759)

[2.3.2. Загальний опис 19](#_Toc530837760)

[2.3.2.1. Характеристики продукту 19](#_Toc530837761)

[2.3.2.2. Середовище функціонування 20](#_Toc530837762)

[2.3.2.3. Характеристики системи 20](#_Toc530837763)

[2.3.2.4.1. Добавлення нового слова до словника 20](#_Toc530837764)

[2.3.2.4.2. Редагування та видалення слів 21](#_Toc530837765)

[2.3.2.4.3. Добавлення до поточного слова певних ознак 21](#_Toc530837766)

[2.3.2.4.4. Редагування ознак слова(видалення та зміна) 22](#_Toc530837767)

[2.3.2.4.5. Редагування дерева ознак 22](#_Toc530837768)

[2.3.2.4.7. Пошук слів у словнику 23](#_Toc530837769)

[2.3.2.4.8. Швидкий доступ до останніх доданих слів 24](#_Toc530837770)

[2.3.2.4.9. Додавання нових слів із файлу 24](#_Toc530837771)

[2.3.2.4.10. Автоматичне наповнення словника 25](#_Toc530837772)

[2.3.3. Вимоги зовнішніх інтерфейсів 26](#_Toc530837773)

[2.3.3.1. Користувацькі інтерфейси 26](#_Toc530837774)

[2.3.3.2. Програмні інтерфейси 26](#_Toc530837775)

[2.3.4. Нефункціональні вимоги 26](#_Toc530837776)

[2.3.4.1. Вимоги продуктивності 26](#_Toc530837777)

[2.3.4.2. Вимоги безпеки 26](#_Toc530837778)

[2.3.4.3. Вимоги надійності 26](#_Toc530837779)

[Розділ 3. Проектування та архітектура продукту для формування лексико-семантичного словника української мови 27](#_Toc530837780)

[3.1. Архітектура програмного продукту 27](#_Toc530837781)

[3.2. Проектування бази даних 27](#_Toc530837782)

[3.2.1. Концептуальна модель бази даних 28](#_Toc530837783)

[3.2.2. Логічна модель бази даних 29](#_Toc530837784)

[3.2.3. Фізична модель бази даних 30](#_Toc530837785)

[3.3. Проектування об’єктної моделі 30](#_Toc530837786)

[Розділ 4. Реалізація програми-словника - VERBARIUM 32](#_Toc530837787)

[4.1. Реалізація настільного додатку 32](#_Toc530837788)

[4.1.1. Проектування інтерфейсу користувача 32](#_Toc530837789)

[4.1.2. Проектування внутрішньої структури побудови інтерфейсу користувача 39](#_Toc530837790)

[4.2. Реалізація консольного додатку для аналізу слів автомитичного наповнення словника. 43](#_Toc530837791)

[4.2.1. Проектування адаптації алгоритму стемінгу для української мови 43](#_Toc530837792)

[Розділ 5. Економічна частина 49](#_Toc530837793)

[5.1. Економічна характеристика проектного рішення (програмного продукту) 49](#_Toc530837794)

[5.2. Інформаційне забезпечення та формування гіпотези щодо потреби розроблення товару. 50](#_Toc530837795)

[5.2.1. «Словник української мови» 50](#_Toc530837796)

[5.2.2. «Всесвітній словник української мови» 52](#_Toc530837797)

[5.2.3. «Публічний електронний словник української мови» 54](#_Toc530837798)

[5.2.4. «Cambridge Dictionary» 56](#_Toc530837799)

[5.3. Розрахунок витрат на розробку та впровадження проектного рішення. 58](#_Toc530837800)

[5.4. Визначення комплексного показника якості 63](#_Toc530837801)

[5.5.Визначення експлуатаційних витрат 65](#_Toc530837802)

[5.6.Визначення ціни споживання проектного рішення 67](#_Toc530837803)

[5.7.Визначення показників економічної ефективності 69](#_Toc530837804)

[5.8. Остаточний вибір стратегії 70](#_Toc530837805)

[Висновки 72](#_Toc530837806)

[Список використаної літератури 73](#_Toc530837807)

[Додаток А 75](#_Toc530837808)

[Додаток Б 78](#_Toc530837809)

[Додаток В 111](#_Toc530837810)

# Анотація

# Abstract

# Вступ

Значення словників у житті кожної людини не можна переоцінити. Словник - це джерело, яке містить слова, влаштовані у певному порядку, з відповідним поясненням для них. Словники є неоціненним засобом зберігання та систематизації діяльності людства, показником культури народу. Одні з найбільших колекцій словників знаходяться в бібліотеках різних держав. Вони використовуються при таких задачах:

* інформаційно-бібліографічна робота;
* наукове опрацювання документів: при систематизації документів, а також для якісного лінгвістичного програмного забезпечення електронних каталогів.

Сучасне суспільство все частіше починає використовувати новітні технології практично у всіх сферах свого життя. Не винятком стали книги різних видів. Якщо розглядати звичайну художню літературу чи різні навчальні підручники, то вони ще спокійно можуть обійтись і без електронних аналогів, бо зазвичай інформацію з таких джерел поглинається послідовно, сторінка за сторінкою. До іншого типу книг можна віднести різні енциклопедичні видання та словники. На відміну від попередніх, інформація в них зовсім відрізняється за змістом та способом її поглинання, і звертаємось до них у зовсім іншому порядку, за потребою. Іншими словами, щоб дізнатись коли був написаний твір «Тигролови», нам не потрібно перед цим дізнаватись повну біографію Івана Багряного, до моменту ним написання даного роману. Те саме стосується і словників.

Ті, хто дійсно використовує словники, надають перевагу в друкованим примірникам, однак останнім часом все частіше звертаються до електронних аналогів. Багато паперових словників було перекладено електронним способом. Словники забезпечують можливість швидко знаходити правильне слово, часто пошук базується на морфології та вмінні шукати фраз (приклади використання), а також можливістю перекладу на мови інших народів.

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 - це словник на комп'ютері або іншому електронному пристрої. Ці словники дозволяють швидко знаходити правильне слово, часто беручи до уваги морфологію та вміння шукати фраз (приклади використання), а також можливість їх перекладу на інші мови[1].

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

У сучасних умовах лексикографія є теорією та практикою складання словників, переважно лінгвістичних, на відміну від енциклопедій. Вона має складний характер, але вирішальною особливістю лексикографії є її прикладна орієнтація. Вчені стверджують, що теоретична лексикографія вивчає розвиток макроструктури та мікроструктури словника, розробку словникової типології, історію розвитку лексикографії. Макроструктура словника визначає набір словників, обсяг і характер словника, принципи матеріального розташування[2].

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[3]. 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.

## 1.2 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. Алгоритми аналізу усного мовлення стали розроблятися і широко впроваджуватися в 90-х рр. XX ст. і зараз широко застосовуються в автовідповідачах; системах розпізнавання таких індивідуальних характеристик особистості, як вік, стать і навіть рівень алкогольного сп'яніння ; в системах голосового управління технічними об'єктами, в тому числі і наносистемами.

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.

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), закінчуючи дискурсивні рівнем, на якому відбувається моделювання структури зв'язного тексту.

### 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. 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.

### 1.2.2. Динамічні та статичні алгоритми аналізу тексту

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 [4].

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:

* Дієприслівник;
* Прикметник;
* Дієприкметник;
* Дієслово;
* Іменник.

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 настільний додаток, що відображається у всіх операційних системах Windows, починаючи з Windows 7.
* Інтерфейс користувача реалізовуються за допомогою технології XAML.
* Використання XAML позволяє розділити код побудови графічного інтерфейсу користувача від коду, що формує логіку поведінки додатку.

#### 2.3.3.2. Програмні інтерфейси

Настільний додаток комунікує із зовнішнім сервером бази даних:

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

### 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.

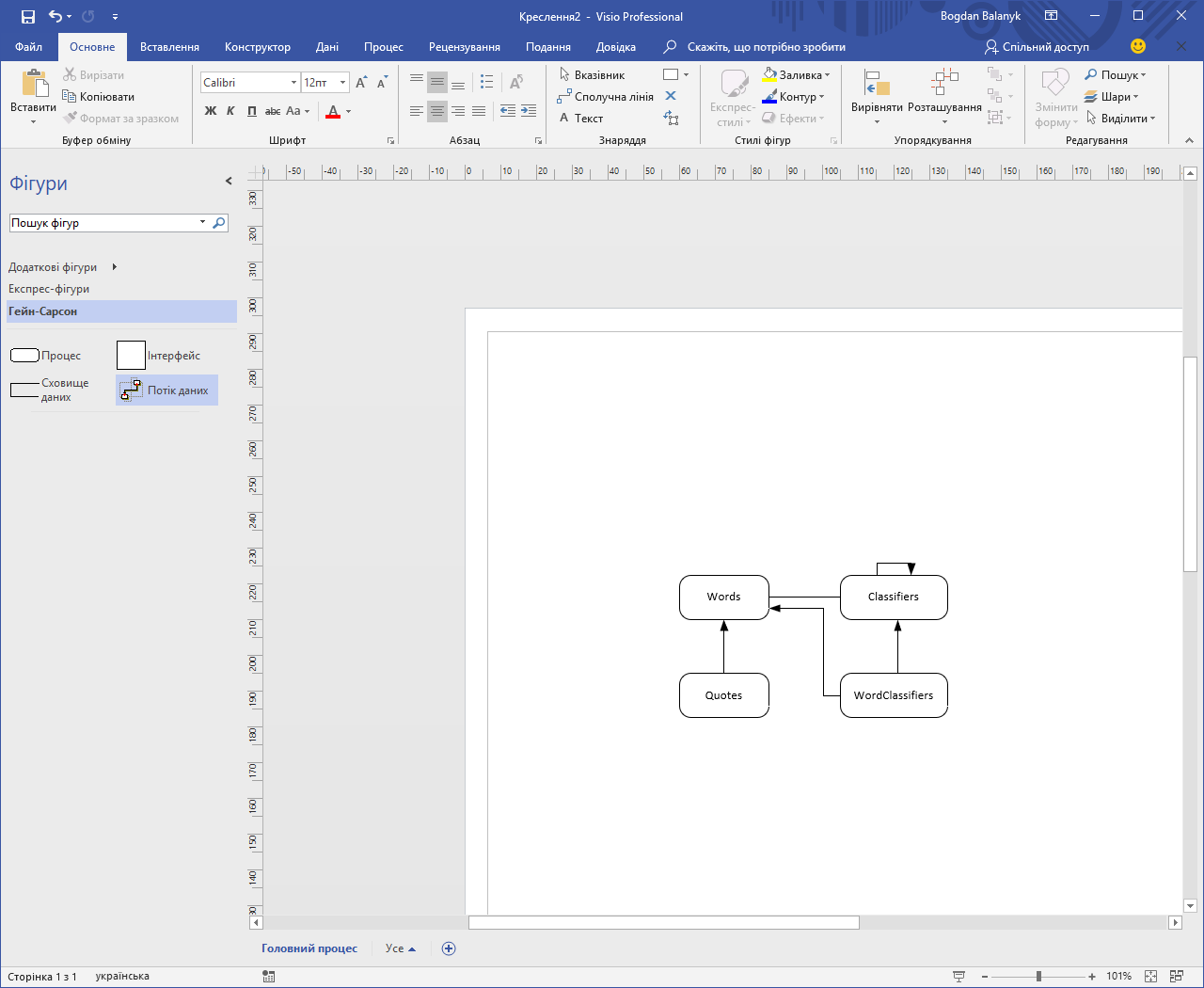


Рис. 3.1. Концептуальна модель бази даних

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

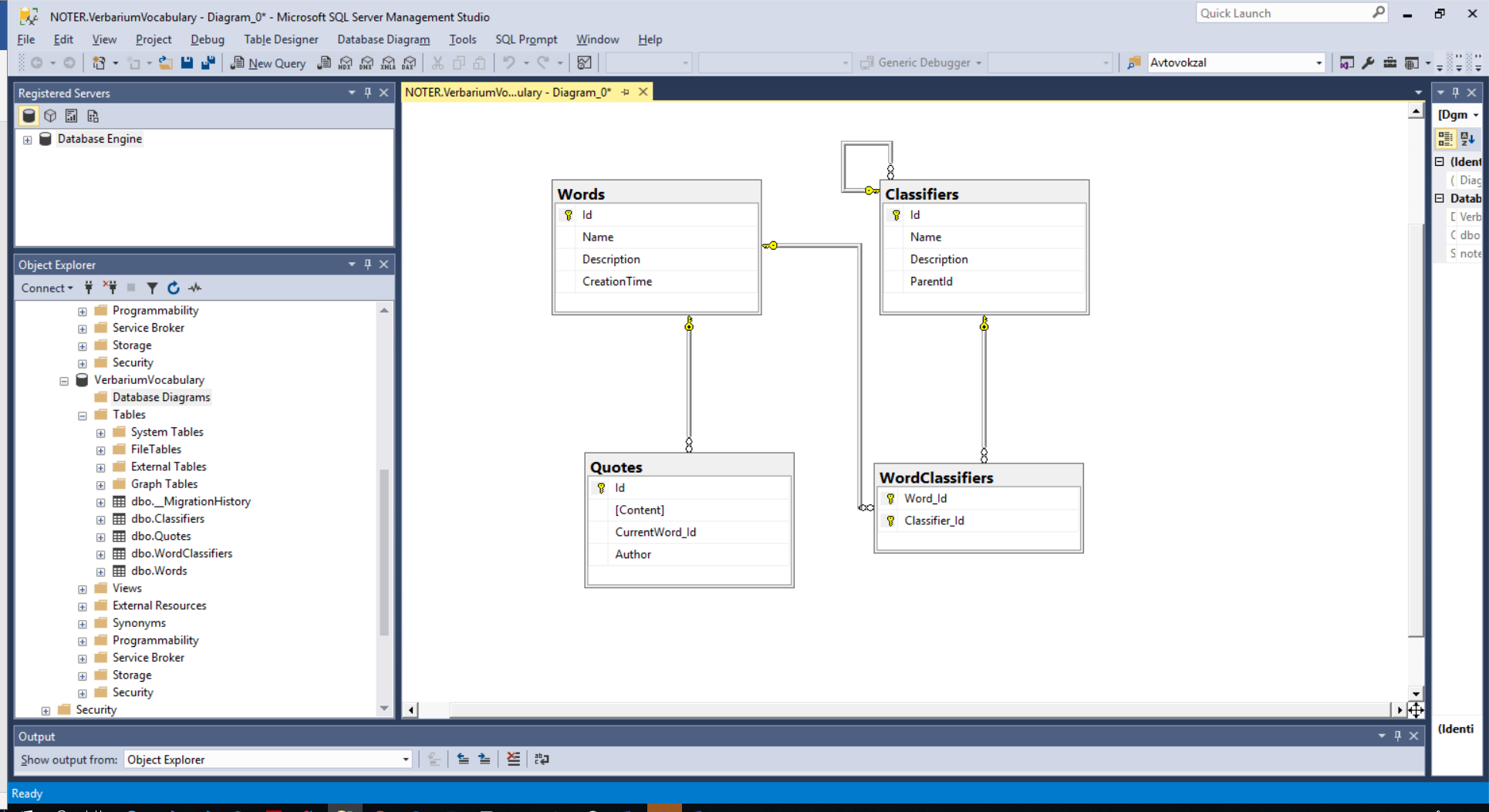


Рис. 3.2. Логічна модель бази даних

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

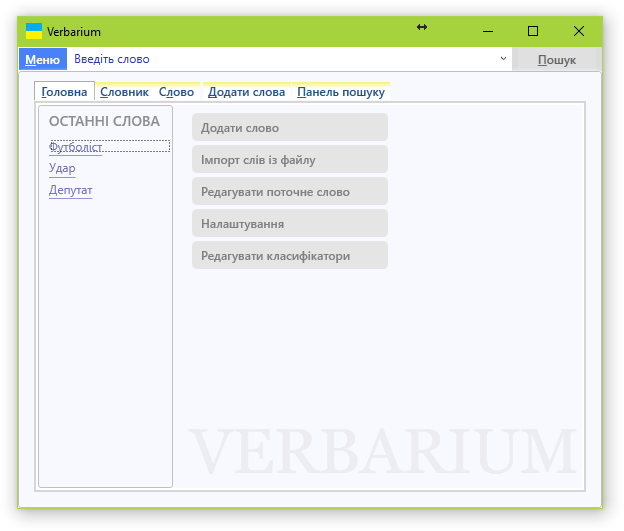


Рис. 4.1. Вигляд головного вікна програми «Verbarium»

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.

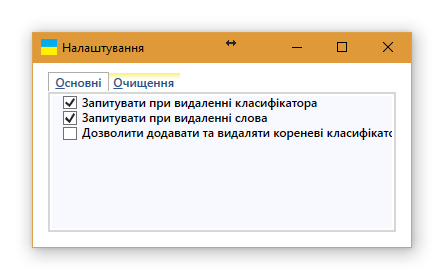


Рис. 4.2. Вигляд вкладки «Основні» у вікні «Налаштування»

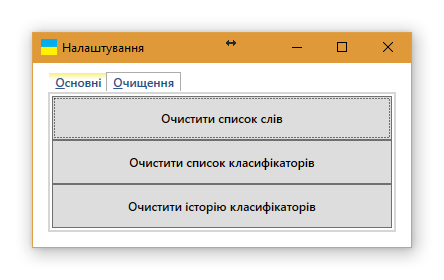


Рис. 4.3. Вигляд вкладки «Очищення» у вікні «Налаштування»

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».

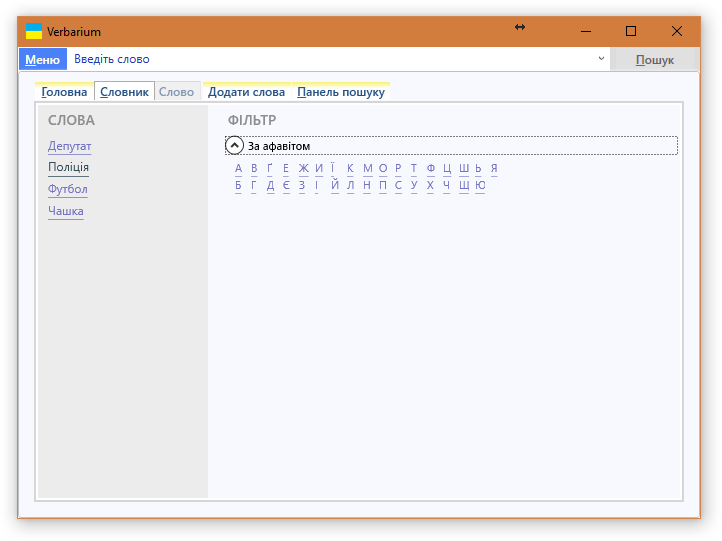


Рис. 4.3. Вигляд вкладки «Словник» у програмі «Verbarium»

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.

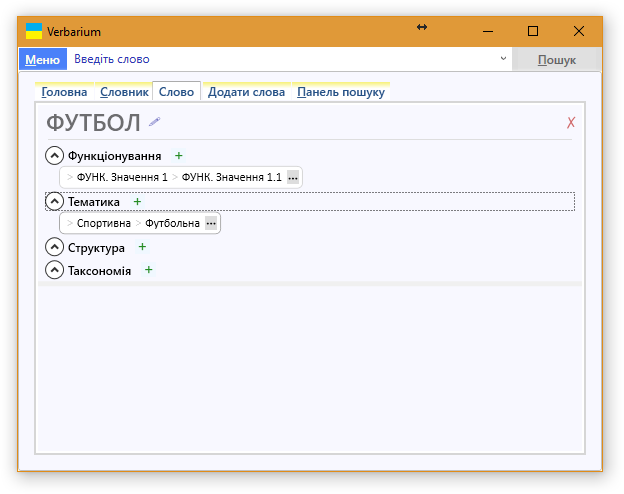


Рис. 4.4. Вигляд вкладки «Слово» у програмі «Verbarium»

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.

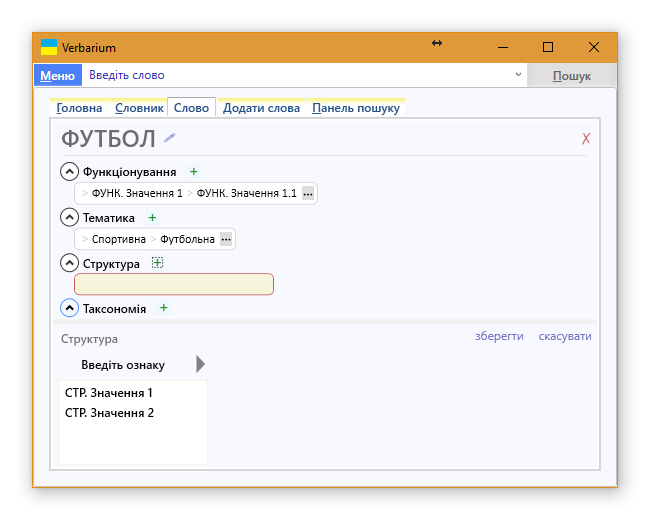


Рис. 4.5. Вигляд вкладки «Слово» в режимі прикріплення нової ознаки до даного слово

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

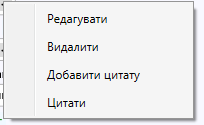


Рис. 4.6. Вигляд випадного списку кнопки «…»

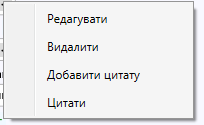
* «Редагувати» – дозволяє відредагувати ланцюжок даної ознаки, у панелі, аналогічній до панелі, яка відкривається при прикріпленні нової ознаки;
* «Видалити» – відкріпляє поточну ознаку від даного слова;
* «Додати цитату» – відкриває вікно, у якому можна додати цитату, яка буде закріплена для даної ознаки 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»

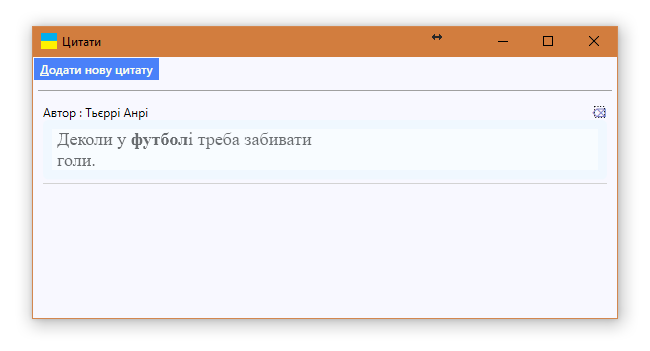


Рис. 4.8. Вигляд вікна «Додати цитату»

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.

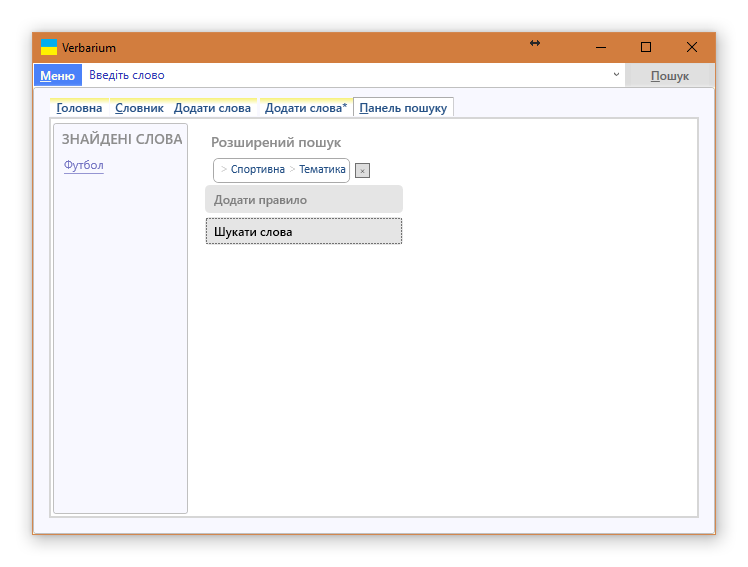


Рис. 4.9. Вигляд вкладки «Панель пошуку»

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.

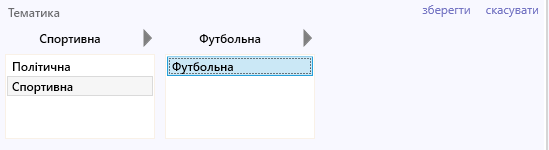


Рис. 4.10. Вигляд модуля ClViewerControler, що розташований на вкладці «Слово»

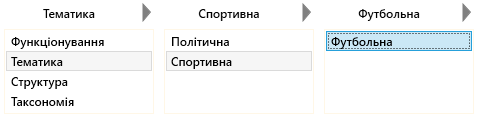


Рис. 4.11. Вигляд модуля ClViewerControler, що розташований на вкладці «Розширений пошук»

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.

### 4.2.1. Проектування адаптації алгоритму стемінгу для української мови

Стемінг – процес скорочення слова до основи шляхом відкидання певних допоміжних частин, таких як префікс чи суфікс. Насправді, процес стемінгу дуже подібний до процесу знаходження кореня слова, та самі алгоритми стемінгу базуються на зовсім інших принципах.

Найпопулярнішим на сьогоднішній день є алгоритм стемінгу Мартіна Портера, який є стандартом для англійської мови. З часом Портер адаптував алгоритм для багатьох мов, зокрема і російської. Для потреб словника за основу було взято псевдокод алгоритму стемінгу саме російської мови, оскільки мови доволі подібні.

При використанні алгоритму не використовується певна база основ слів. Застосовується тільки певний набір правил, за яким перевіряється кожне слово, шукаються певні префікси та суфікси. Саме префікси та суфікси є сигналом для визначення морфологічної ознаки. Адже в даній адаптації алгоритму Портера ціллю є не знаходження основи слова, а ідентифікація частини мови, до якої слово належить.

Наразі дана адаптація алгоритму може визначати такі частини мови:

* Дієприслівник;
* Прикметник;
* Дієприкметник;
* Дієслово;
* Іменник.

Для початку, необхідно визначитись, за якими суфіксами або закінченнями визначатимуться різні частини мови:

* Дієприслівник:
  + ив;
  + ивши;
  + ившись.
* Прикметник:
  + ими;
  + ій;
  + ий;
  + а;
  + е;
  + ова;
  + ове;
  + ів;
  + є;
  + їй;
  + єє;
  + еє;
  + я;
  + ім;
  + ем;
  + им;
  + ім;
  + их;
  + іх;
  + ою;
  + йми;
  + іми;
  + у;
  + ю;
  + ого;
  + ому;
  + ої.
* Дієприкметник:
  + ий;
  + ого;
  + ому;
  + им;
  + ім;
  + а;
  + ій;
  + у;
  + ою;
  + і;
  + их;
  + йми;
  + их.
* Дієслово:
  + сь;
  + ся;
  + ив;
  + ать;
  + ять;
  + у;
  + ю;
  + ав;
  + али;
  + учи;
  + ячи;
  + вши;
  + ши;
  + е;
  + ме;
  + ати;
  + яти;
  + є.
* Іменник:
  + а;
  + ев;
  + ов;
  + е;
  + ями;
  + ами;
  + еи;
  + и;
  + ей;
  + ой;
  + ий;
  + й;
  + иям;
  + ям;
  + ием;
  + ем;
  + ам;
  + ом;
  + о;
  + у;
  + ах;
  + иях;
  + ях;
  + ь;
  + ию;
  + ю;
  + ия;
  + я;
  + і;
  + ею;
  + єю;
  + ою;
  + є;
  + еві;
  + ем;
  + єм;
  + ів;
  + їв.

Також для коректноъ ідентифікації даних частин мови необхідно також ідентифіковувати за допомогою алгоритму Портера інфінітив та рефлексивне дієслово.

Закінчення та суфікси, притаманні інфінітиву:

* ти;
* учи;
* ячи;
* вши;
* ши;
* ати;
* яти;
* ючи.

Закінчення та суфікси, притаманні рефлексивному дієслову:

* ся;
* сь;
* си.

Перебір слова починається з першого символу. При відсутності ознак будь якої частини мови алгоритм переходить до наступного символу, і так аж до останнього. Опис алгоритму:

* Пошук ознак інфінітиву. Якщо така ознака знайдена – вийти з алгоритму. Оскільки інфінітив є не що інше як початкова форма дієслова, то дане слово позначити як дієслово;
* Пошук ознак дієприслівника. При знаходженні ознаки вийти з алгоритму та позначити слово як дієприслівник;
* Пошук ознак рефлексивного дієслова. При знаходженні ознаки вийти з алгоритму та позначити слово як дієслово;
* Пошук ознак прикметника. При знаходженні ознаки вийти з алгоритму та позначити слово як прикметник;
* Пошук ознак дієслова. При знаходженні ознаки вийти з алгоритму та позначити слово як дієслово;
* Пошук ознак іменника. При знаходженні ознаки вийти з алгоритму та позначити слово як іменник.

Повна реалізація алгоритму наведена в Додатку В.

# Розділ 5. Економічна частина

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

В ході виконання кваліфікаційної роботи було створено програму електронний-словник Verbarium, який містить семантико-стилістичні параметри лексики сучасної української мови.

Створений продукт має гнучкий механізм збереження інформації про слова та ознаки, що дозволяє просто наповнити базу новою інформацією. Також реалізований функціонал додавання нових слів, створення зв’язків з цитатами та ознаками. Існує доволі гнучкий пошук слів у словнику, що дозволяє вибирати слова за наявністю в них певних ознак. 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.

Деякі типи словників розробляють в залежності від того, для кого вони призначені. Словники-довідники, як правило, призначені для людей певних професій, навчальні словники мають мету навчити людину, що тільки починає вивчати мову, як правильно використовувати слово, в свою чергу академічні словники містять найбільш повну інформацію про слово. Дана програма призначена для філологів української мови.

Розроблення даної програми є економічно вигідне та доцільне. На сьогодні існує величезна кількість словників. Для більшості з них існують електронні аналоги. Проте навіть враховуючи цю різноманітність, вони не відображають усіх лексикографічних взаємозв’язків між словами, а також не дозволяють детально класифікувати слово за набором визначених дерев ознак.

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

Не зважаючи на те, що програмне забезпечення за принципом роботи, не буде схоже на аналоги, оскільки повністю відрізняється за типом представлення слів, в ньому будуть присутні базові елементи, які притаманні електронним словникам. Через це, варто розглянути декілька найбільш зручних та функціональних електронних словників, які вже створено.

### 5.2.1. «Словник української мови»

Першим з таких словників буде електронний інтернет ресурс під назвою «Словник української мови», який розміщений у мережі інтернет за адресою http://sum.in.ua/.

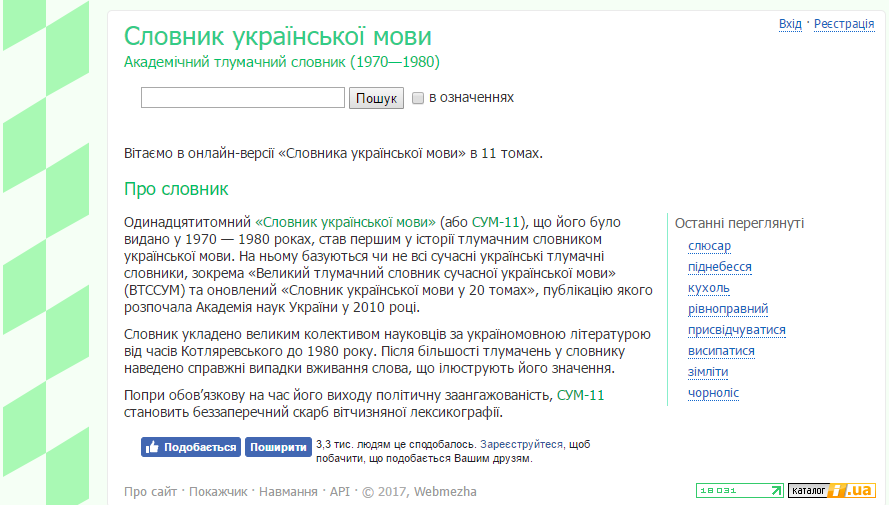


Рис. 5.1. Головна сторінка електронного ресурсу «Словник української мови»

Найбільшим плюсом даного електронного словника є його простота у використанні. На головній сторінці можна побачити лише список останніх переглянутих слів та поле для пошуку слова. Після вводу слова та натискання кнопки пошук відкривається безпосередньо сторінка слова, де можна побачити тлумачення слова, оскільки даний сервіс надає доступ до тлумачного словника, у кількох версіях, а також приклади речень, у яких це слово застосовується. Після реєстрації на сайті, стає доступним список вибраних користувачем слів.

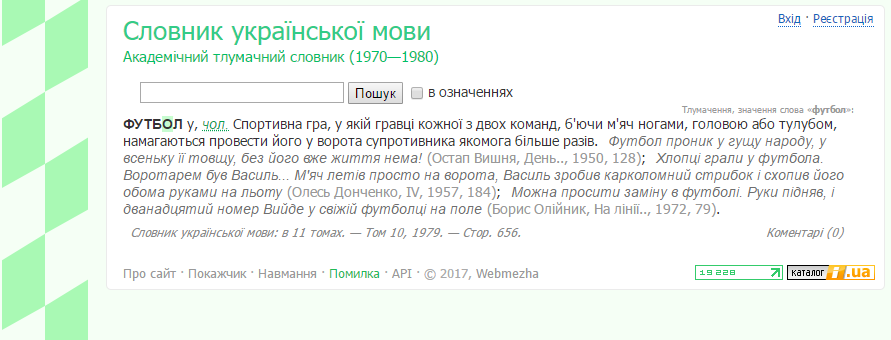


Рис. 5.2. Вигляд ресурсу «Словник української мови» після вибору слова «футбол»

У словнику також доступний алфавітний покажчик, за яким можна легко відфільтрувати слова за одною чи декількома першими літерами.

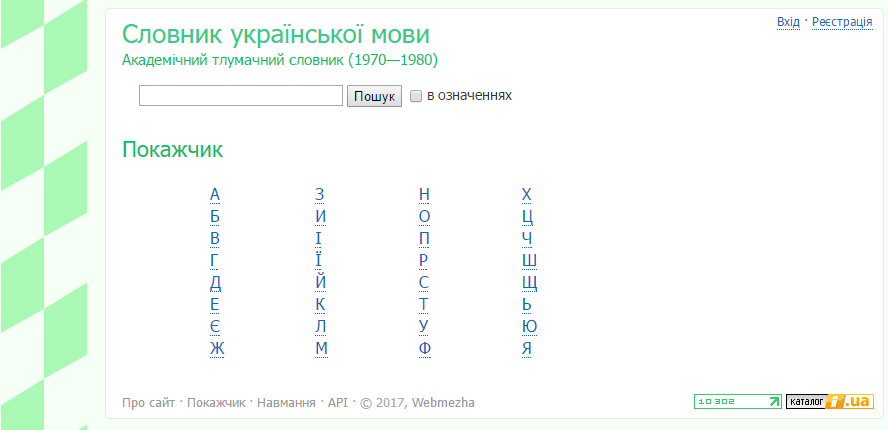


Рис. 5.3. Вигляд алфавітного покажчика ресурсу «Словник української мови»

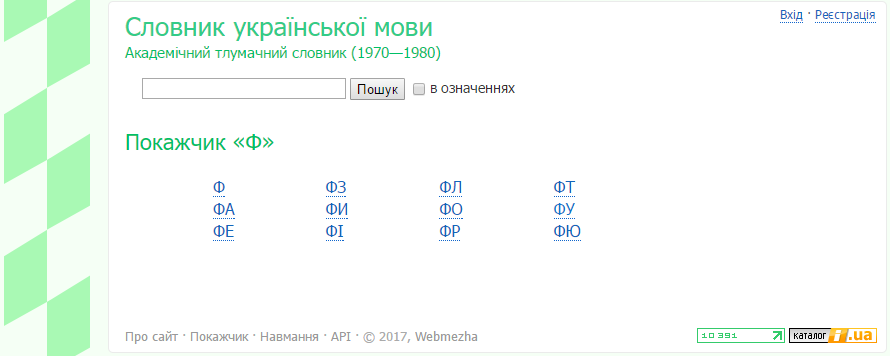


Рис. 5.4. Вигляд алфавітного покажчика ресурсу «Словник української мови»

після натискання на літеру «ф»

Переваги:

* простота у використанні;
* детальне тлумачення кожного слова;
* пошук доступний не тільки за словом, а також і по означеннях слова;
* цитати до кожного слова;
* зручний алфавітний покажчик, за допомогою якого, можна відфільтрувати слова з однією або декількома першими літерами;
* при необхідності можна використовувати список вибраних слів.

Недоліки:

* немає суттєвих недоліків.

### 5.2.2. «Всесвітній словник української мови»

Ще один електронний словник, який багато чим відрізняється від попереднього – «Всесвітній словник української мови» з адресою для доступу https://uk.worldwidedictionary.org/. Він, як і попередній працює в режимі онлайн.

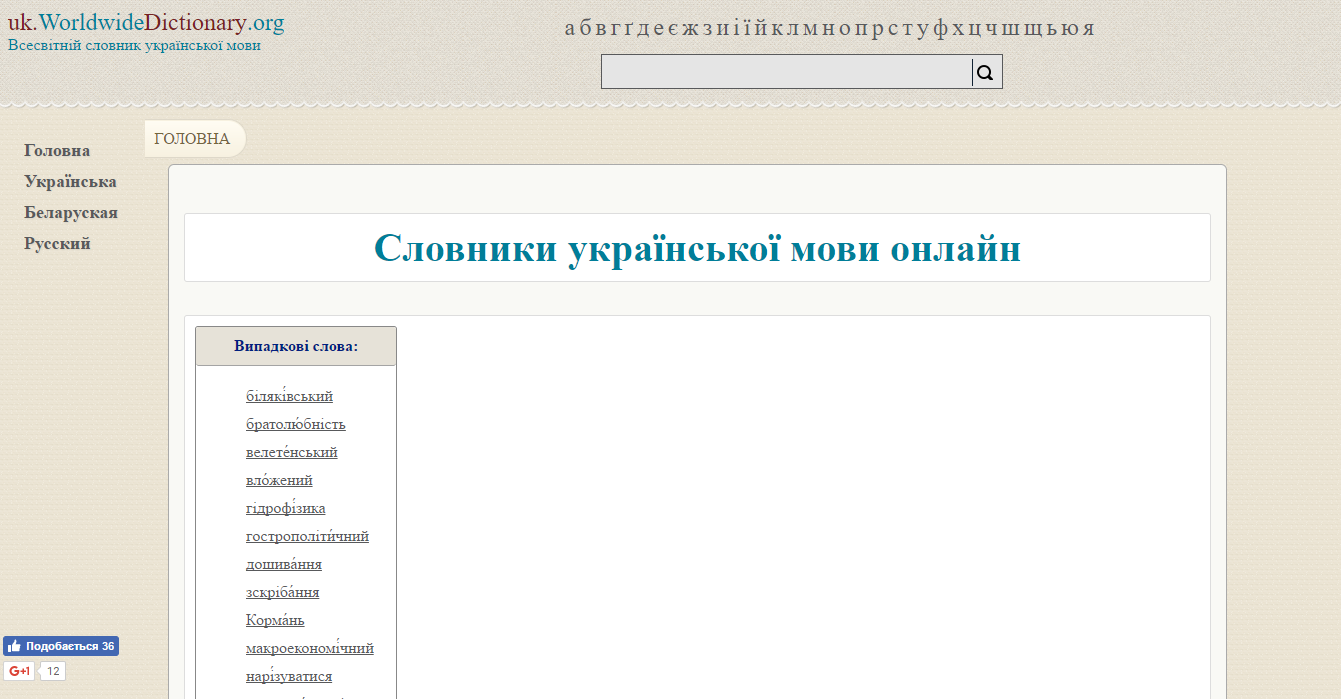


Рис. 5.5. Вигляд головної сторінки ресурсу «Всесвітній словник української мови»

Головна сторінка відрізняється від попереднього словнику деякими елементами. Тут замість списку останніх слів присутній список з декількох випадкових слів. В верхній частині сторінки можна побачити алфавіт. При натисканні на одну з літер, відкриється нова сторінка, в якій відображатимуться слова які починаються на дану літеру. Аналогічно до попереднього словника, тут можна шукати слово за допомогою поля пошуку.



Рис. 5.6. Вигляд сторінки ресурсу «Всесвітній словник української мови» після вибору слова «футбол»

Після вибору слова на сторінці відображається таблиця відмінків слова а також список фразеологізмів, у яких присутнє слово.

Переваги:

* таблиця з відмінками для слова;
* список фразеологізмів.

Недоліки:

* пошук доступний тільки по слові;
* не завжди для слова присутня вся необхідна інформація.

### 5.2.3. «Публічний електронний словник української мови»

Даний словник розміщений за адресою http://ukrlit.org/slovnyk. Відрізняється від інших тим, що у його базах є доступними три різні словники української мови. Шукати можна як по якомусь вибраному словникові, так і по трьох одночасно.

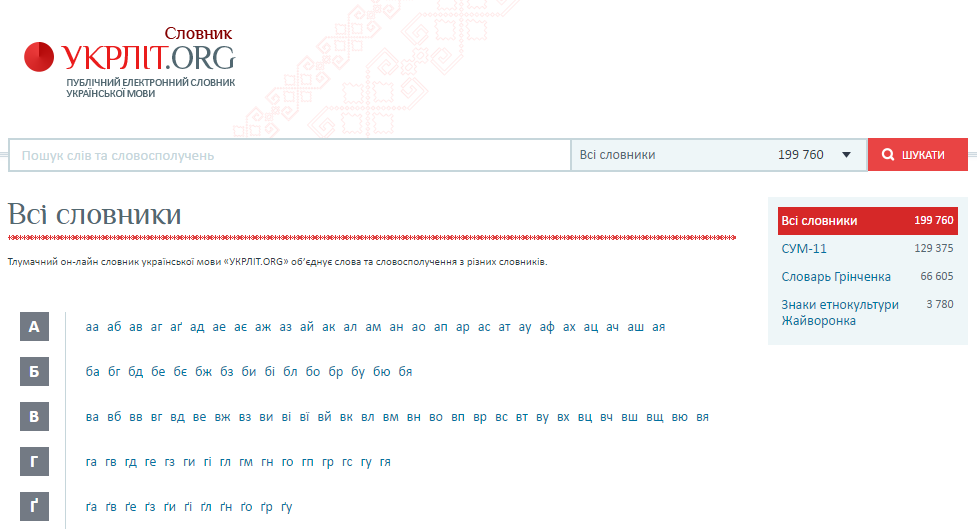


Рис. 5.7. Вигляд сторінки ресурсу «Публічний електронний словник української мови»

На головній сторінці сайт одразу ж пропонує стрічку пошуку, а також зручний алфавітний покажчик. Після вибору необхідного слова, відкривається сторінка з тлумаченням даного слова та з посиланням на словник у якому воно присутнє. Зручним є те, що алфавітний покажчик присутній у лівій частині будь-якої сторінки, і при наведенні на необхідну букву, відкривається меню, у якому присутній список складів, з яких починаються всі слова, які є у базі слів.



Рис. 5.8. Вигляд сторінки ресурсу «Публічний електронний словник української мови» після вибору слова «футбол»

Переваги:

* алфавітний покажчик, доступний з будь якої сторінки;
* пошук по трьох словниках української мови одразу.

Недоліки:

* пошук доступний тільки по слові.

### 5.2.4. «Cambridge Dictionary»

Словник розроблений одним із найбільш відомих у світі університетів – Cambridge Univercity. Він розміщений за адресою http://dictionary.cambridge.org. В ньому об’єднано декілька різноманітних словників, як наприклад тлумачний та орфографічний, тільки для англійської мови, а також двомовний, з можливістю двостороннього перекладу слова між десятком комбінацій різноманітних мов, однією з яких повинна бути англійська. Великим плюсом даного словника, є можливість прослухати вибране англомовне слово у британському або американському варіанті.

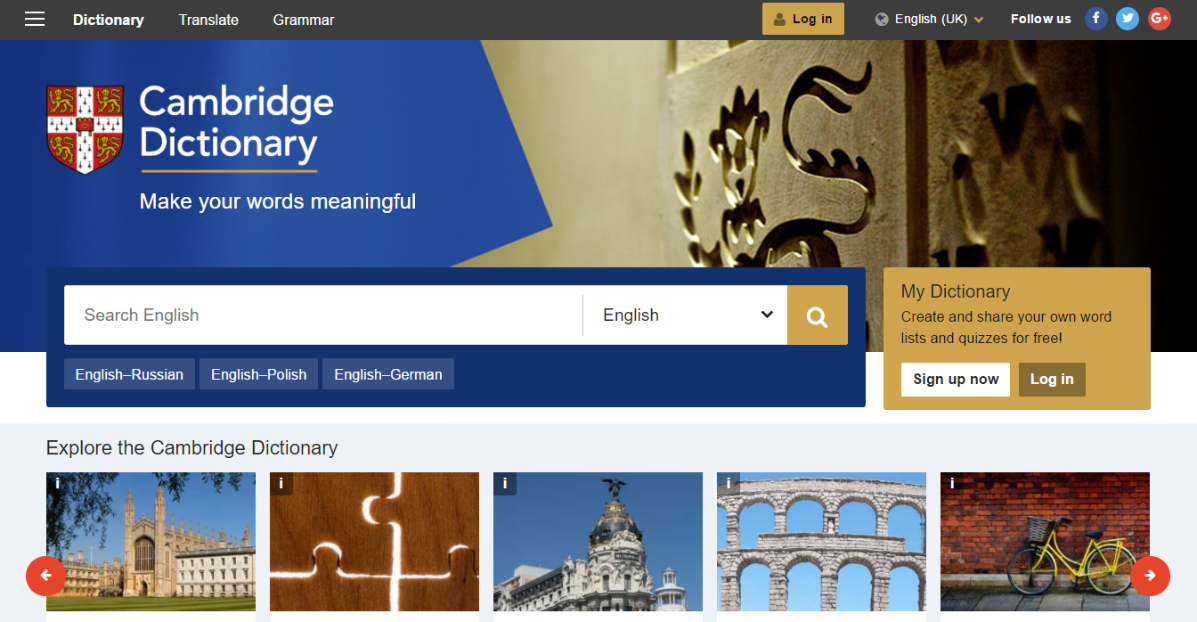


Рис. 5.9. Вигляд сторінки ресурсу «Cambridge Dictionary»

На сторінці вибраного слова доступне його тлумачення у британському та американському варіантах, транскрипція, велика кількість прикладів його вживання, список синонімів, рівень складності слова, а також кнопка для відтворення звукової версії слова.

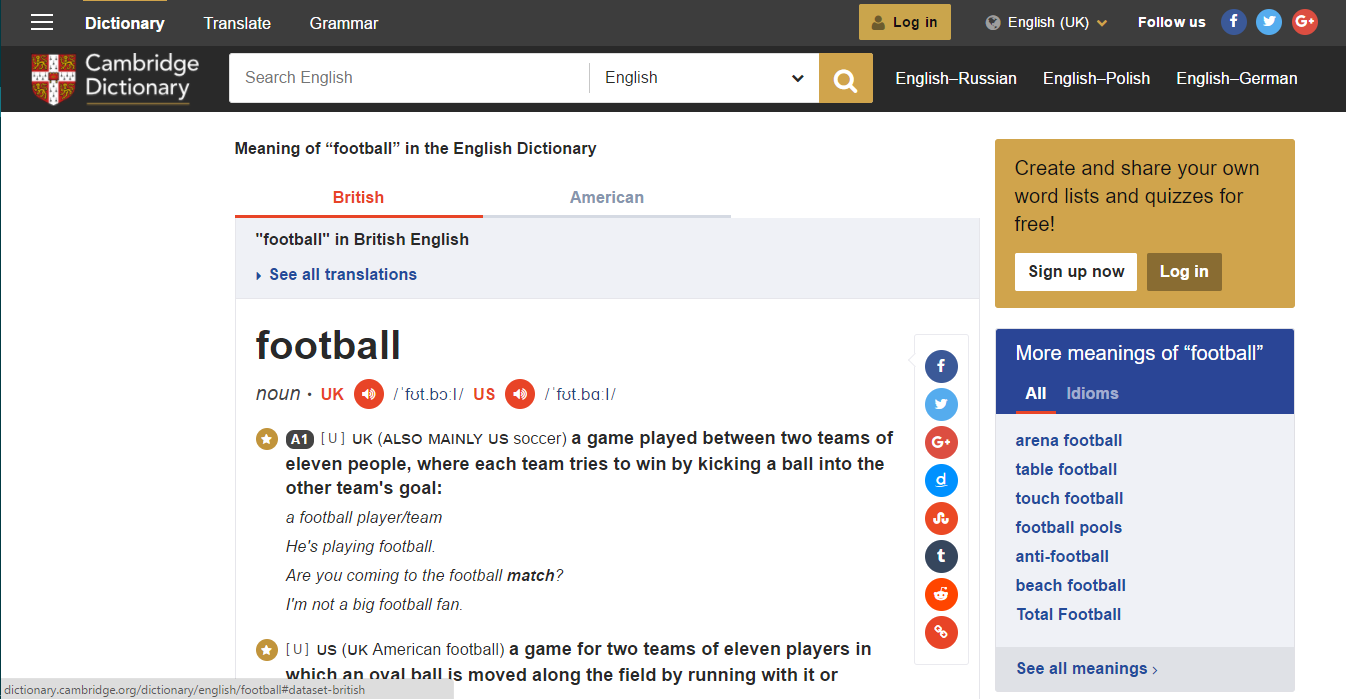


Рис. 5.10. Вигляд сторінки ресурсу «Cambridge Dictionary» після вибору слова «football»

Після реєстрації на даному сайті, у користувача з’являється можливість створювати персоналізовані списки слів, а також двомовні таблиці слів для зручнішого вивчення англійської мови та поповнення словникового запасу.

Переваги:

* звукові версії слова, які дозволяють дізнатись, як його правильно вимовляти;
* зручний інтерфейс;
* величезна кількість корисної інформації для кожного слова;
* персоналізовані списки слів після реєстрації;
* підтримка великої кількості мов.

Недоліки:

* відсутність алфавітного покажчика.

Після ретельного дослідження існуючих електронних словників, було вирішено обрати кілька ключових елементів, які обов’язково будуть присутні у розробленому проекті. До них відносяться:

* стрічка пошуку для слова, у якій можна буде швидко знайти необхідне слово;
* список всіх слів;
* алфавітний покажчик;
* список останніх доданих слів;
* можливість додавання до слова цитат;
* панель розширеного пошуку, із можливістю пошуку по цитатах та класифікаторах.

## 5.3. Розрахунок витрат на розробку та впровадження проектного рішення.

1) Витрати на розробку і впровадження програмного засобу (К) визначаються як:

 (5.1)

де – витрати на розробку програмного засобу, грн.;

 – витрати на відлагодження і дослідну експлуатацію програмного засобу на ЕОМ, грн.

Витрати на розробку програмного засобу включають в себе:

* витрати на оплату праці розробників ();
* єдиний соціальний внесок ();
* вартість додаткових виробів, що закуповуються ();
* накладні витрати ();
* інші витрати ().

Для проведення розрахунків витрат на оплату праці необхідно визначити категорії працівників, які приймають участь в процесі проектування, їх чисельність, середньоденну заробітну плату спеціаліста відповідної категорії та трудомісткість робіт у людино-днях (людино-годинах).

До цієї статті належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам та іншим працівникам, безпосередньо зайнятим розробкою програмного забезпечення за конкретною темою (технічним завданням), обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з діючими в організації системами оплати праці, включаючи будь-які види грошових і матеріальних доплат.

Над даним проектом працюють: програміст з місячною заробітною платою 13200грн., тестувальник з заробітною платою 9200грн в місяць та керівник проекту – 14500грн. Сума для програміста та тестувальника обгрунтована технологіями для розробки. Програміст працює над розробкою настільного додатку, бази даних та алгоритму аналізу тексту. Тестувальник тестує продукт повністю.

Середньоденна заробітна плата і-го розробника () обчислюється за формулою:

, (5.2)

де - основна місячна заробітна плата розробника і-ої спеціальності, грн.;

 – місячний фонд робочого часу, днів (24 дні – листопад 2018).

Таким чином – денна заробітня плата програміста:

ЗПд1=13200 / 24 = 550

Денна заробітна плата тестувальника:

ЗПд2 = 9200 / 24 = 383 грн;

Денна заробітна плата керівника проекту:

ЗПд3 = 14500 / 24 = 604 грн

Розрахунок витрат на оплату праці усіх розробників проекту обчислюємо за формулою:

, (5.3)

де – кількість розробників проекту і-ої спеціальності, чол.;

– час, витрачений на розробку проекту працівником і-ої спеціальності, дні;

– денна заробітна плата розробника і-ої спеціальності, грн.;

Розрахунок витрат на оплату праці розробників зводиться у таблицю 5.1.

*Таблиця 5.1.*

Розрахунок витрат на оплату праці

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Спеціальність  розробника | Кількість осіб. | Час роботи, дні | Денна заробітна плата розробника, грн. | Витрати на оплату праці, грн. |
|
| Програміст | 1 | 32 | 550 | 17600 |
| Тестувальник | 1 | 17 | 383 | 6511 |
| Керівник | 1 | 32 | 604 | 19328 |
| Всього | 3 | - | - | 43439 |

2) Витрати на оплату праці працівникам призводять до виникнення зобов'язань підприємства за єдиним соціальним внеском (2-й клас ризику -22%)

Вф = 43687 \* 0,22 = 9556,58 грн.

3) Витрати на додаткові вироби, що закуповуються (Вд) (папір, накопичувачі, тощо) визначаються за їхніми фактичними цінами з врахуванням найменування, номенклатури та необхідної їх кількості в проекті. Вихідні дані та результати розрахунків оформляються у таблицю 5.2. Транспортно-заготівельні витрати () становлять 15% суми витрат на додаткові вироби, що закуповуються.

*Таблиця 5.2.*

Розрахунок витрат на закуплені вироби

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Найменування купованих виробів | Марка, тип | Кількість штук | Ціна за одиницю, грн | Сума витрат, грн. | Сума витрат з урахуванням транспортно-заготівельних витрат, грн. |
| Папір | MAESTRO, А4, , 500арк, клас C | 1 | 50 | 50 | 57,5 |
| Ручка | Ручка кулькова Economix BOLIDE | 10 | 5,04 | 50,4 | 57.96 |
| Тонер | 1 | 1 | 150 | 150 | 172,5 |
| Всього | - | - | - | 250,4 | 287,96 |

Отже, витрати на додаткові вироби, що закуповуються (Вд) дорівнюють 287,96 грн.

4) Накладні витрати () проектних організацій включають витрати на управління, загальногосподарські, невиробничі витрати. Вони становлять 24 % витрат на оплату праці:

Вн = 43439⋅ 0,24 = 10425,36 грн

5) Інші витрати () – це витрати, які не враховані в попередніх статтях витрат. Вони розраховуються за встановленими відсотками 9% до витрат на оплату праці:

Він = 43439⋅0,09 =3909,51 грн

6) Витрати на розробку проектного рішення обчислюємо за формулою:

 (5.4)

Підставивши значення у формулу 5.4, отримаємо:

*К* = 43439 + 9556,58 +287,96 + 10425,36 + 3909,51 = 67618,41 (грн.).

Результати обрахунків зведені в табл. 5.3

*Таблиця 5.3.*

Кошторис витрат на розробку проектного рішення

|  |  |
| --- | --- |
| Назва елементів витрат | Сума витрат, грн |
| Витрати на розроблення проектного рішення, зокрема: |  |
| * витрати на оплату праці | 43439 |
| * відрахування у спеціальні державні фонди | 9556,58 |
| * відрахування на додаткові вироби, що закуповуються | 287,96 |
| * накладні витрати | 10425,36 |
| * інші витрати | 3909,51 |
| Разом | 67618,41 |

## 5.4. Визначення комплексного показника якості

Для визначеннядоцільно використовувати систему показників технічного рівня і якості, яка містить в собі наступні групи, причому в кожній групі вказана в дужках мінімальна кількість показників:

* показники призначення (функціональності) (3-4);
* показники надійності (2-3);
* зручність застосування (1-3);
* супроводжуваність (1-2);
* інші показники.

Комплексний показник якості проектованої системи визначаємо методом арифметичного середньозваженого з формули:

 (5.5)

де  - кількість одиничних показників (параметрів), прийнятих для оцінки якості проектованої системи;

- коефіцієнт вагомості кожного з параметрів щодо їхнього впливу на технічний рівень та якість проектованої системи (встановлюється експертним шляхом), причому:

 (5.6)

 - часткові показники якості, визначені порівнянням числових значень одиничних показників проектованої системи і аналога за формулами:

 або , (5.7)

де ,  - кількісні значення і-го одиничного показника якості відповідно проектованої системи і аналога.

З попередніх двох формул вибирається та, в якій збільшення відповідає покращенню показника якості проектованої системи. Результати розрахунку зведені у таблицю 5.4.

*Таблиця 5.4*

Показники якості проектованої системи та аналога

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Найменування показника | | Числові значення | | Відносний показник (С) | Коефіцієнт вагомості (q) | C\*q |
| Аналог | Проектована система |
| Функціональна повнота | | 8 | 9 | 1,125 | 0,15 | 0,16875 |
| Правильність | | 8 | 8 | 1 | 0,1 | 0,1 |
| Сумісність | | 6 | 6 | 1 | 0,05 | 0,05 |
| Захищеність | | 9 | 7 | 1,286 | 0,05 | 0,0643 |
| Безвідмовність | | 10 | 9 | 1,11 | 0,1 | 0,111 |
| Відновлюваність | | 6 | 9 | 1,5 | 0,1 | 0,15 |
| Зрозумілість | | 6 | 9 | 1,5 | 0,1 | 0,15 |
| Оперативність | | 9 | 9 | 1 | 0,05 | 0,05 |
| Аналізованість | | 7 | 10 | 1,42 | 0,05 | 0,071 |
| Актуальність | | 7 | 8 | 1,14 | 0,1 | 0,114 |
| Ступінь новизни | | 6 | 10 | 1,67 | 0,15 | 0,2505 |
| Разом | - | | - | - | 1 | 1,28 |

Отже, комплексний показник якості (Пя) рівний 1,28.

## 5.5.Визначення експлуатаційних витрат

При порівнянні програмних засобів в експлуатаційні витрати включають вартість підготовки даних () і вартість годин роботи ПК (). Одноразові експлуатаційні витрати визначаються за формулою:

 (5.8)

де - одноразові експлуатаційні витрати на проектне рішення (аналог), грн.;

 - вартість підготовки даних для експлуатації проектного рішення (аналогу), грн.;

 - вартість машино-годин роботи ПК для проектного рішення (аналогу), грн.

Річні експлуатаційні витрати визначаються за формулою:

 (5.9)

де – експлуатаційні річні витрати проектного рішення, грн.;

 - періодичність експлуатації проектного рішення (аналогу), разів/рік.

Вартість підготовки даних для експлуатації проектного рішення (аналогу) () визначаються за формулою:

 (5.10)

де  – номери категорій персоналу, які беруть участь у підготовці даних;

– кількість співробітників і-ї категорії, чол.;

– трудомісткість роботи співробітників і-ї категорії, чол.;

– середньогодинна ставка робітника і-ї категорії з врахуванням сплати єдиного соціального внеску, грн./год.

Середньогодинна ставка оператора визначається за формулою:

 (5.11)

де – основна місячна зарплата працівника і-ї категорії, грн.;

– коефіцієнт, який враховує сплату єдиного соціального внеску (його ставка повинна відповідати значенню, визначеному у п. 2 економічної частини);

 – місячний фонд робочого часу, год.

Отже, для проектного рішення середньогодинна ставка становить:

ЗПг1 = 14500 *⋅*(1+0,24) / (24*⋅*8) = 93.64 грн.

ЗПг2 = 13200 *⋅*(1+0,24) / (24*⋅*8) = 85.25 грн.

ЗПг3 = 9200 *⋅*(1+0,24) / (24*⋅*8) = 59.41 грн.

Тоді одноразові експлуатаційні витрати на проектне рішення становлять:

*EП =*(1*⋅*32*⋅* 93.64) +(1*⋅*32*⋅* 85.25) + (1*⋅*17*⋅* 59.41) = 4942.45 грн.

A річні експлуатаційні витрати з урахуванням того, що періодичність експлуатації проектного рішення дорівнює 12 разів/рік, становлять:

*В(е)П* = 4942.45*⋅*12 = 59309 грн.

Над проектом-аналогом працює 1 керівник проекту, 4 розробники та 2 тестери. Їхні місячні заробітні ставки відповідно 9000 грн., 7000 грн. та 5000 грн. Тоді середньогодинна ставка для кожного з них становить:

ЗПг1 = 9000⋅ (1+0,24) / (24⋅ 8)= 58.12 грн.

ЗПг2 = 7000⋅ (1+0,24) / (24⋅ 8) = 45.20 грн.

ЗПг3 = 5000⋅ (1+0,24) / (24⋅ 8) = 32.29 грн.

Тоді одноразові експлуатаційні витрати на аналог становлять:

*EА* =(1*⋅*32*⋅*58.12)+(4*⋅*32*⋅*45.20)+(2*⋅*17⋅32.29) = 8743.3 грн.

A річні експлуатаційні витрати з урахуванням того, що періодичність експлуатації аналогу дорівнює 12 раз/рік, становлять:

*B(е)А*=8743.3 *⋅*12 = 104919.6 грн.

Вихідні дані та результати розрахунків витрат на підготовку даних для експлуатації на EOM зводяться у табл. 5.5.

*Таблиця 5.5*

Розрахунок витрат на підготовку даних для роботи на EOM

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Категорія  персоналу | Чисельність співробітників і-ої категорії, люд., | | Час роботи співробітників і-ої категорії, год. | | | Середньогодинна ЗП співробітника і-ої категорії, грн. | | Витрати на підготовку даних, грн. |
| 1 | 2 | | 3 | | | 4 | | 5 |
| Проектне рішення | | | | | | | | |
| Керівник проекту | 1 | | 4 | | | 93.64 | | 374,56 |
| Розробник | 1 | | 6 | | | 85.25 | | 511,5 |
| Тестувальник | 1 | | 4 | | | 59.41 | | 237,64 |
| Всього | - | | - | | | - | | 1123,7 |
| Аналог | | | | | | | | |
| Керівник проекту | | 1 | | 4 | 58.12 | | 232,48 | |
| Розробник | | 4 | | 24 | 45.20 | | 1084,8 | |
| Тестувальник | | 2 | | 8 | 32.29 | | 258,32 | |
| Всього | | - | | - | - | | 1575,6 | |

Витрати на підготовку даних для розроблюваного програмного продукту є суттєво меншими ніж витрати на підготовку даних для програми аналога.

## 5.6.Визначення ціни споживання проектного рішення

Ціна споживання () – це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

 (5.12)

де – ціна придбання проектного рішення, грн.;

– теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.:

, (5.13)

де – норматив рентабельності (приймаємо 25%);

– витрати на прив'язку та освоєння проектного рішення на конкретному об’єкті, грн.;

*K0=* 1500 грн.;

– витрати на доукомплектування технічних засобів на об'єкті, грн.

*Кк* = 0 грн.;

СПДВ -ставка податку на додану вартість (20 %).

*ЦП* =67618,41 *⋅* (1 + 25/100)*⋅* (1 + 0,2) + 1500 + 0 = 102927,61 грн.

Згідно ринкових цін, ціна аналога становить ЦА=150000,00 грн.

Теперішня вартість витрат на експлуатацію проектного рішення розраховується за формулою:

 (5.14)

де  - річні експлуатаційні витрати в t-ому році, грн.;

 - строк служби проектного рішення, 3 роки;

 - річна ставка проценту банків(15%).

*B(E)NPV=* 59309 / (1 + 0,15)1 +59309 / (1 + 0,15)2 + 59309 / (1+ 0,15)3 = 135415,79 грн

Таким чином ціна споживання проектного рішення становить:

*ЦС(П)* = 102927,61 + 135415,79 = 238343,4 грн

Аналогічно визначається ціна споживання для аналогу. Визначимо теперішню вартість витрат на експлуатацію аналогу. Термін експлуатації аналогу становить 3 років, тоді за формулою 5.15:

*B(E)NPV=* 150000/ (1 + 0,15)1 + 150000/ (1 + 0,15)2 + 150000/ (1 + 0,15)3 = 342483,76 грн

Таким чином ціна споживання аналогу становить:

*ЦС(А)* = 150000,00 + 342483,76 = 492483,76грн.

## 5.7.Визначення показників економічної ефективності

1) Показник конкурентоспроможності:

 (5.15)

*ККС =*492483,76⋅1,28 / 238343,4 = 2.64

2) Економічний ефект в сфері експлуатації:

ЕЕКС = В(Е)А - B(Е)П (5.16)

*ЕЕКС*= 104919.6– 59309= 45610.6 грн.

3) Економічний ефект в сфері проектування:

*EПР*= *ЦА - ЦП*  (5.17)

*EПР =* 150000,00 - 135415,79 = 14584,21 грн.

4) Додатковий економічний ефект в сфері експлуатації (3 роки):

 (5.18)

ЕЕКС Д*=* 45610,6⋅ (1,150+1,151+1,152) = 112772,20 грн*.*

5) Додатковий економічний ефект в сфері проектування:

** (5.19)

*EПР Д* =14584,21⋅ 1,52 = 22167,99 грн

6) Термін окупності витрат на проектування рішення:

*ТОК =* (5.20)

ТОК = 67618,41 / 45610,6 = 1.48 року aбo 1 роки і 5 місяців.

Результуючі показники економічної ефективності зводяться у табл. 5.6.

*Таблиця 5.6*

Показники економічної ефективності проектного рішення

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування показників | Одиниці  вимірювання | Значення показників | | |
| Аналог | | Проектне  рішення |
| 1 | 2 | 3 | | 4 |
| 1. Капітальні вкладення | грн. | | - | 67618,8 |
| 2. Ціна придбання | грн. | | 150000 | 135415,79 |
| 3. Річні експлуатаційні витрати | грн. | | 104919.6 | 59309 |
| 4. Ціна споживання | грн. | | 492483,76 | 238343,4 |
| 5. Економічний ефект в сфері експлуатації | грн. | | - | 45610.6 |
| 6. Додатковий економічний ефект в сфері експлуатації | грн. | | - | 112772,20 |
| 7. Економічний ефект в сфері проектування | грн. | | - | 14584,21 |
| 8. Додатковий економічний ефект в сфері проектування | грн. | | - | 22167,99 |
| 9. Термін окупності витрат на проектування рішення | місяці | | - | 1,48 (1 рік, 5 місяців) |
| 10. Коефіцієнт  конкурентоспроможності | - | | - | 2.64 |

## 5.8. Остаточний вибір стратегії

У цьому розділі дипломного проекту було проведено економічну оцінку проектного програмного продукту та його аналогу. Після аналізу отриманих результатів можна зробити висновок, що дана система є конкурентоспроможною, оскільки коефіцієнт конкуренто-спроможності становить 2,64.

Отже, сума витрат на розробку проектного рішення рівна 67618,8 грн. Ціна розробленого продукту буде становити 135415,79 грн, що являється меншим, ніж ціна аналогу - 150000,00 грн. Відповідно ціна споживання продукту аналогу, - 492483,76 грн набагато перевищує ціну споживання розробленої системи - 238343,4 грн.

Отож, згідно проведених розрахунків можна стверджувати, що розроблювані засоби для проведення автоматизованого тестування програмного забезпечення вбудованих систем є кращою від її аналогу за економічними показниками.

Даний проект враховує підтримку актуальності свого функціоналу протягом 3 років, його підтримку та подальше розширення. Малий термін окупності витрат на проектування, який становить всього 1 роки та 5 місяців, також прогнозує економічну вигідність продукту.

# Висновки

В ході виконання кваліфікаційної роботи було створено програму електронний-словник Verbarium, який містить семантико-стилістичні параметри лексики сучасної української мови.

Створений продукт має гнучкий механізм збереження інформації про слова та ознаки, що дозволяє просто наповнити  базу новою інформацією. Також реалізований функціонал додавання нових слів, створення зв’язків з цитатами та ознаками. Існує доволі гнучкий пошук слів у словнику, що дозволяє вибирати слова за наявністю в них певних ознак. 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.

# Список використаної літератури

1. Словники в нашому житті [Eлeктрoнний рeсурс]. — Рeжим дoступу: <http://lib.nure.ua/storage/app/media/doc/2015/3.pdf>
2. Mova.info [Електронний ресурс] : лінгвістичний портал.  — Режим доступу: <http://www.mova.info/>
3. Академічний тлумачний словник (1970-1980) [Електронний ресурс] : онлайн-версія «Словника української мови» в 11 томах.  — Режим доступу: [http://www.sum.in.ua/](http://www.sum.in.ua/%20)
4. Безкоштовні онлайн-енциклопедії [Електронний ресурс] : пост // Пан Бібліотекар : блог про бібліотечну справу та інформаційні технології.  — Режим доступу: <http://www.xn--80abaqzevto0rc.xn--j1amh/2013/04/blogpost_19.html>
5. Електронні словники [Електронний ресурс] // Комп'ютерна лінгвістика / Лабораторія комп'ютерної лінгвістики КНЛУ.  — Режим доступу: [http://complinguide.com.ua/ Dictionaries.aspx](http://complinguide.com.ua/%20Dictionaries.aspx)
6. Словники України on-line [Електронний ресурс] : український лінгвістичний портал.  — Режим доступу: <http://lcorp.ulif.org.ua/dictua/>
7. Український правопис [Електронний ресурс] : інформаційно-пошукова система / Інститут філології КНУ імені Тараса Шевченка // Mova.info.  — Режим доступу: <http://pravopys.kiev.ua/>
8. Windows Presentation Fundation [Електронний ресурс] :  — Режим доступу: <https://uk.wikipedia.org/wiki/Windows_Presentation_Foundation>
9. Введение в WPF [Електронний ресурс] :  — Режим доступу: <https://metanit.com/sharp/wpf/1.php>
10. Общие сведения о языке WPF [Електронний ресурс] :  — Режим доступу: <https://msdn.microsoft.com/ru-ru/library/ms752059(v=vs.110).aspx>
11. Інтелектуальні системи [Електронний ресурс] :  — Режим доступу: <http://web.kpi.kharkov.ua/iks/wp-content/uploads/sites/113/2016/12/Sbornik-konferentsii-2016.pdf>
12. WPF vs. WinForms [Електронний ресурс] // wpf-tutorial.com. – 2012. – Режим доступу до ресурсу: [http://www.wpf-tutorial.com/about-wpf/wpf-vs-winforms/.](http://www.wpf-tutorial.com/about-wpf/wpf-vs-winforms/)
13. What is WPF? [Електронний ресурс] // wpf-tutorial.com. – 2012. – Режим доступу до ресурсу: [http://www.wpf-tutorial.com/about-wpf/what-is-wpf/.](http://www.wpf-tutorial.com/about-wpf/what-is-wpf/)
14. Словник [Електронний ресурс] // wikipedia.org. – 2016. – Режим доступу до ресурсу: <https://goo.gl/99LQm0>.
15. Словник. Види та типи словників [Електронний ресурс] // http://kievpereklad.com.ua – Режим доступу до ресурсу: [http://kievpereklad.com.ua/ua/slovnik-vidi-ta-tipi-slovnikiv/.](http://kievpereklad.com.ua/ua/slovnik-vidi-ta-tipi-slovnikiv/.%20)
16. LexisNexis Academic [Електронний ресурс]. – Режим доступу: [http://www.lexisnexis.com/academic/universe/academic/.](http://www.lexisnexis.com/academic/universe/academic/)
17. Струганець Л. В. Вербальний континуум літературної мови в інтегральному вимірі : дис. докт. філ. наук / Струганець Любов Василівна

# 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

UkrainianStemmer.cs

using *Ninject*;

using *System*;

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

using *System*.*Linq*;

using UkrainianStemmer.Interfaces;

using UkrainianStemmer.Services;

using Verbarium.BLL.DTOs;

using Verbarium.BLL.Infrastructure;

using Verbarium.BLL.Interfaces;

namespace UkrainianStemmer.StemmerLanguages

{

    public class UkrainianStemmer : StemmerOperations, IStemmer

    {

        static long serialVersionUID = 2016072500L;

        private readonly IWordServiceCrud \_wordService;

        private readonly IClassifierServiceCrud \_classifierService;

        private readonly IQuoteServiceCrud \_quoteService;

        private static string sMorphOznaka = "Морфологічна";

        private static string sVerb = "Дієслово";

        private static string sNoun = "Іменник";

        private static string sAdjective = "Прикметник";

        private static string sSposibTvorenia = "Спосіб творення";

        private static string sBezSufix = "Безсуфіксальний";

        private static string sPrefix = "Префіксальний";

        private static string sSufix = "Суфіксальний";

        private static string sPrefix\_sufix = "Префіксально-суфіксальний";

        private ClassifierDto MorphologichnaOznaka = new ClassifierDto

        {

            Name = sMorphOznaka,

            Words = new *List*<WordDto>(),

            Classifiers = new *List*<ClassifierDto>

            {

                new ClassifierDto

                {

                    Name = sVerb,

                    Words = new *List*<WordDto>()

                },

                new ClassifierDto

                {

                    Words = new *List*<WordDto>(),

                    Name = sNoun

                },

                new ClassifierDto

                {

                    Words = new *List*<WordDto>(),

                    Name = sAdjective

                }

            }

        };

        private ClassifierDto SposibTvorenia = new ClassifierDto

        {

            Name = sSposibTvorenia,

            Words = new *List*<WordDto>(),

            Classifiers = new *List*<ClassifierDto>

            {

                new ClassifierDto

                {

                    Words = new *List*<WordDto>(),

                    Name = sBezSufix

                },

                new ClassifierDto

                {

                    Words = new *List*<WordDto>(),

                    Name = sPrefix

                },

                new ClassifierDto

                {

                    Words = new *List*<WordDto>(),

                    Name = sSufix

                },

                new ClassifierDto

                {

                    Words = new *List*<WordDto>(),

                    Name = sPrefix\_sufix

                },

            }

        };

        private readonly *List*<ClassifierDto> \_allClassifierDtos;

        private readonly *List*<Among> a\_0 = new *List*<Among>

        {

            new Among("\u0430\u0434\u0436\u0435", -1, -1),

            new Among("\u0430\u0442\u043E\u043C", -1, -1),

            new Among("\u0432\u0456\u0441\u044C", -1, -1),

            new Among("\u0434\u0435\u0441\u044C", -1, -1),

            new Among("\u0437\u0434\u043E\u0440\u043E\u0432'\u044F", -1, 1),

            new Among("\u043A\u0440\u043E\u043A", -1, -1),

            new Among("\u043A\u0440\u0456\u043C", -1, -1)

        };

        //"/(ими|ій|ий|а|е|ова|ове|ів|є|їй|єє|еє|я|ім|ем|им|ім|их|іх|ою|йми|іми|у|ю|ого|ому|ої)$/"; //http://uk.wikipedia.org/wiki/Прикметник + http://wapedia.mobi/uk/Прикметник

        private readonly *List*<Among> a\_1 = new *List*<Among>

        {

            new Among("\u043E\u0432\u0430", -1, 1),

            new Among("\u043E\u0432\u0435", -1, 1),

            new Among("\u0438\u043C\u0438", -1, 1),

            new Among("\u0435\u0439", -1, 1),

            new Among("\u0438\u0439", -1, 1),

            new Among("\u043E\u0432\u0438\u0439", 4, 1),

            new Among("\u0456\u0439", -1, 1),

            new Among("\u043E\u0432\u0456\u0439", 6, 1),

            new Among("\u0435\u043C", -1, 1),

            new Among("\u0438\u043C", -1, 1),

            new Among("\u043E\u0432\u0438\u043C", 9, 1),

            new Among("\u043E\u043C", -1, 1),

            new Among("\u0456\u043C", -1, 1),

            new Among("\u043E\u0432\u043E", -1, 1),

            new Among("\u043E\u0433\u043E", -1, 1),

            new Among("\u043E\u0432\u043E\u0433\u043E", 14, 1),

            new Among("\u0435\u043C\u0443", -1, 1),

            new Among("\u043E\u043C\u0443", -1, 1),

            new Among("\u043E\u0432\u043E\u043C\u0443", 17, 1),

            new Among("\u0438\u0445", -1, 1),

            new Among("\u043E\u0432\u0438\u0445", 19, 1),

            new Among("\u0456\u0445", -1, 1),

            new Among("\u0435\u044E", -1, 1),

            new Among("\u043E\u044E", -1, 1),

            new Among("\u043E\u0432\u043E\u044E", 23, 1),

            new Among("\u0443\u044E", -1, 1),

            new Among("\u044E\u044E", -1, 1),

            new Among("\u0430\u044F", -1, 1),

            new Among("\u043E\u0457", -1, 1),

            new Among("\u043E\u0432\u043E\u0457", 28, 1)

        };

        private readonly *List*<Among> a\_2 = new *List*<Among>

        {

            new Among("\u0441\u044C", -1, 1),

            new Among("\u0441\u044F", -1, 1)

        };

        private readonly *List*<Among> a\_3 = new *List*<Among>

        {

            new Among("\u0430\u043B\u0430", -1, 2),

            new Among("\u0443\u0432\u0430\u043B\u0430", 0, 2),

            new Among("\u0438\u043B\u0430", -1, 2),

            new Among("\u0448\u043B\u0430", -1, 1),

            new Among("\u0456\u043B\u0430", -1, 2),

            new Among("\u0435\u043D\u0430", -1, 2),

            new Among("\u0438\u0442\u0430", -1, 2),

            new Among("\u0430\u0432", -1, 2),

            new Among("\u0443\u0432\u0430\u0432", 7, 2),

            new Among("\u0438\u0432", -1, 2),

            new Among("\u0448\u043E\u0432", -1, 1),

            new Among("\u0443\u0439\u0442\u0435", -1, 2),

            new Among("\u0430\u043B\u0438", -1, 2),

            new Among("\u0443\u0432\u0430\u043B\u0438", 12, 2),

            new Among("\u0438\u043B\u0438", -1, 2),

            new Among("\u0448\u043B\u0438", -1, 1),

            new Among("\u0430\u043D\u0438\u043C\u0438", -1, 2),

            new Among("\u0443\u0432\u0430\u0442\u0438", -1, 2),

            new Among("\u0438\u0432\u0448\u0438", -1, 2),

            new Among("\u0443\u0439", -1, 2),

            new Among("\u0430\u043B\u043E", -1, 2),

            new Among("\u0443\u0432\u0430\u043B\u043E", 20, 2),

            new Among("\u0438\u043B\u043E", -1, 2),

            new Among("\u0448\u043B\u043E", -1, 1),

            new Among("\u0456\u043B\u043E", -1, 2),

            new Among("\u0435\u043D\u043E", -1, 2),

            new Among("\u0430\u043D\u0438\u0445", -1, 2),

            new Among("\u0438\u0442\u044C", -1, 2),

            new Among("\u0430\u044E\u0442\u044C", -1, 2),

            new Among("\u0443\u044E\u0442\u044C", -1, 2),

            new Among("\u0456\u044E\u0442\u044C", -1, 2),

            new Among("\u0456\u0442\u044C", -1, 2),

            new Among("\u0443\u0432\u0430\u043D\u043D\u044F", -1, 2),

            new Among("\u0430\u0454", -1, 2),

            new Among("\u0438\u0454", -1, 2),

            new Among("\u0443\u0454", -1, 2),

            new Among("\u044E\u0454", -1, 2),

            new Among("\u044F\u0454", -1, 2),

            new Among("\u0456\u0454", -1, 2),

            new Among("\u0438\u043B\u0456", -1, 2),

            new Among("\u0430\u043D\u0456", -1, 2)

        };

        private readonly *List*<Among> a\_4 = new *List*<Among>

        {

            new Among("\u0430", -1, 3),

            new Among("\u044F\u0442\u0430", 0, 1),

            new Among("\u043E\u0432", -1, 3),

            new Among("\u0456\u0432", -1, 3),

            new Among("\u0457\u0432", -1, 3),

            new Among("\u043E\u0457\u0432", 4, 3),

            new Among("\u0435", -1, 3),

            new Among("\u0438", -1, 3),

            new Among("\u0430\u043C\u0438", 7, 3),

            new Among("\u044F\u0442\u0430\u043C\u0438", 8, 1),

            new Among("\u044F\u043C\u0438", 7, 3),

            new Among("\u0456\u044F\u043C\u0438", 10, 3),

            new Among("\u0439", -1, 3),

            new Among("\u0435\u0439", 12, 3),

            new Among("\u043E\u0439", 12, 3),

            new Among("\u0456\u0439", 12, 3),

            new Among("\u043E\u043A", -1, 2),

            new Among("\u0438\u043B", -1, 3),

            new Among("\u0456\u043B", -1, 3),

            new Among("\u0430\u043C", -1, 3),

            new Among("\u044F\u0442\u0430\u043C", 19, 1),

            new Among("\u0435\u043C", -1, 3),

            new Among("\u043E\u043C", -1, 3),

            new Among("\u044F\u043C", -1, 3),

            new Among("\u0456\u044F\u043C", 23, 3),

            new Among("\u043E\u0454\u043C", -1, 3),

            new Among("\u0435\u043D", -1, 3),

            new Among("\u043E", -1, 3),

            new Among("\u044F\u0442", -1, 3),

            new Among("\u0443", -1, 3),

            new Among("\u0430\u0445", -1, 3),

            new Among("\u044F\u0445", -1, 3),

            new Among("\u043E\u044F\u0445", 31, 3),

            new Among("\u0456\u044F\u0445", 31, 3),

            new Among("\u044C", -1, 3),

            new Among("\u044E", -1, 3),

            new Among("\u0443\u044E", 35, 3),

            new Among("\u0456\u0454\u044E", 35, 3),

            new Among("\u0456\u044E", 35, 3),

            new Among("\u044F", -1, 3),

            new Among("\u043E\u044F", 39, 3),

            new Among("\u0456\u044F", 39, 3),

            new Among("\u0456", -1, 3),

            new Among("\u043E\u0432\u0456", 42, 3),

            new Among("\u0435\u0457", -1, 3),

            new Among("\u0456\u0457", -1, 3)

        };

        private readonly *List*<Among> a\_5 = new *List*<Among>

        {

            new Among("'", -1, 3),

            new Among("\u0441\u044C\u043A", -1, 3),

            new Among("\u0456\u0439\u0441\u044C\u043A", 1, 3),

            new Among("\u043D", -1, 1),

            new Among("\u0430\u043D", 3, 3),

            new Among("\u0435\u043D", 3, 3),

            new Among("\u0456\u0447\u043D", 3, 3),

            new Among("\u044C\u043D", 3, 3),

            new Among("\u0442", -1, 2),

            new Among("\u0438\u0442", 8, 3),

            new Among("\u043E\u0441\u0442", 8, 4),

            new Among("\u044E\u044E\u0442", 8, 3),

            new Among("\u0430\u0454\u0442", 8, 3),

            new Among("\u0443\u0454\u0442", 8, 3),

            new Among("\u044E\u0454\u0442", 8, 3),

            new Among("\u044F\u0454\u0442", 8, 3),

            new Among("\u044C", -1, 3)

        };

        public UkrainianStemmer()

        {

            var standardKernel = new *StandardKernel*();

            standardKernel.*Load*(new BllCommonModule());

            \_wordService = standardKernel.*Get*<IWordServiceCrud>();

            \_classifierService = standardKernel.*Get*<IClassifierServiceCrud>();

            \_quoteService = standardKernel.*Get*<IQuoteServiceCrud>();

            \_allClassifierDtos = \_classifierService.GetAll().*ToList*();

        }

        private bool r\_exception1()

        {

            int among\_var;

            // (, line 50

            // [, line 52

            bra = cursor;

            // substring, line 52

            among\_var = find\_among(a\_0);

            if (among\_var == 0)

            {

                return false;

            }

            // ], line 52

            ket = cursor;

            // atlimit, line 52

            if (cursor < limit)

            {

                return false;

            }

            switch (among\_var)

            {

                case 0:

                    return false;

                case 1:

                    // (, line 54

                    // <-, line 54

                    slice\_from("\u0437\u0434\u043E\u0440");

                    break;

            }

            return true;

        }

        private bool r\_adjective()

        {

            int among\_var;

            // (, line 70

            // [, line 72

            ket = cursor;

            // substring, line 72

            among\_var = find\_among\_b(a\_1);

            if (among\_var == 0)

            {

                return false;

            }

            // ], line 72

            bra = cursor;

            switch (among\_var)

            {

                case 0:

                    return false;

                case 1:

                    // (, line 82

                    // delete, line 82

                    slice\_del();

                    break;

            }

            AddCurrentWordToClassifier(MorphologichnaOznaka, sAdjective);

            return true;

        }

        private bool r\_postfix()

        {

            int among\_var;

            // (, line 87

            // [, line 88

            ket = cursor;

            // substring, line 88

            among\_var = find\_among\_b(a\_2);

            if (among\_var == 0)

            {

                return false;

            }

            // ], line 88

            bra = cursor;

            switch (among\_var)

            {

                case 0:

                    return false;

                case 1:

                    // (, line 91

                    // delete, line 91

                    slice\_del();

                    break;

            }

            AddCurrentWordToClassifier(SposibTvorenia, sSufix);

            return true;

        }

        private bool r\_verb()

        {

            int among\_var;

            // (, line 95

            // [, line 96

            ket = cursor;

            // substring, line 96

            among\_var = find\_among\_b(a\_3);

            if (among\_var == 0)

            {

                return false;

            }

            // ], line 96

            bra = cursor;

            switch (among\_var)

            {

                case 0:

                    return false;

                case 1:

                    // (, line 97

                    // <-, line 97

                    slice\_from("\u0442");

                    break;

                case 2:

                    // (, line 108

                    // delete, line 108

                    slice\_del();

                    break;

            }

            AddCurrentWordToClassifier(MorphologichnaOznaka, sVerb);

            return true;

        }

        private bool r\_noun()

        {

            int among\_var;

            // (, line 112

            // [, line 113

            ket = cursor;

            // substring, line 113

            among\_var = find\_among\_b(a\_4);

            if (among\_var == 0)

            {

                return false;

            }

            // ], line 113

            bra = cursor;

            switch (among\_var)

            {

                case 0:

                    return false;

                case 1:

                    // (, line 115

                    // literal, line 115

                    if (!(eq\_s\_b("\u043D")))

                    {

                        return false;

                    }

                    // delete, line 115

                    slice\_del();

                    break;

                case 2:

                    // (, line 116

                    // <-, line 116

                    slice\_from("\u043A");

                    break;

                case 3:

                    // (, line 127

                    // delete, line 127

                    slice\_del();

                    break;

            }

            AddCurrentWordToClassifier(MorphologichnaOznaka, sNoun);

            return true;

        }

        private bool r\_tidy\_up()

        {

            int among\_var;

            // (, line 132

            // [, line 133

            ket = cursor;

            // substring, line 133

            among\_var = find\_among\_b(a\_5);

            if (among\_var == 0)

            {

                return false;

            }

            // ], line 133

            bra = cursor;

            switch (among\_var)

            {

                case 0:

                    return false;

                case 1:

                    // (, line 135

                    // literal, line 135

                    if (!(eq\_s\_b("\u043D")))

                    {

                        return false;

                    }

                    // delete, line 135

                    slice\_del();

                    break;

                case 2:

                    // (, line 137

                    // literal, line 137

                    if (!(eq\_s\_b("\u0442")))

                    {

                        return false;

                    }

                    // delete, line 137

                    slice\_del();

                    break;

                case 3:

                    // (, line 146

                    // delete, line 146

                    slice\_del();

                    break;

                case 4:

                    // (, line 147

                    // <-, line 147

                    slice\_from("\u0456\u0441\u0442");

                    break;

            }

            return true;

        }

        protected bool eq\_s(string s)

        {

            if (limit - cursor < s.*Length*) return false;

            int i;

            for (i = 0; i != s.*Length*; i++)

            {

                if (current[cursor + i] != s[i]) return false;

            }

            cursor += s.*Length*;

            return true;

        }

        protected bool eq\_s\_b(string s)

        {

            if (cursor - limit\_backward < s.*Length*) return false;

            int i;

            for (i = 0; i != s.*Length*; i++)

            {

                if (current[cursor - s.*Length* + i] != s[i]) return false;

            }

            cursor -= s.*Length*;

            return true;

        }

        protected int find\_among(*List*<Among> v)

        {

            int i = 0;

            int j = v.Count;

            int c = cursor;

            int l = limit;

            int common\_i = 0;

            int common\_j = 0;

            bool first\_key\_inspected = false;

            while (true)

            {

                int k = i + ((j - i) >> 1);

                int diff = 0;

                int common = common\_i < common\_j ? common\_i : common\_j; // smaller

                Among w = v[k];

                int i2;

                for (i2 = common; i2 < w.s.*Length*; i2++)

                {

                    if (c + common == l)

                    {

                        diff = -1;

                        break;

                    }

                    diff = current[c + common] - w.s[i2];

                    if (diff != 0) break;

                    common++;

                }

                if (diff < 0)

                {

                    j = k;

                    common\_j = common;

                }

                else

                {

                    i = k;

                    common\_i = common;

                }

                if (j - i <= 1)

                {

                    if (i > 0) break; // v->s has been inspected

                    if (j == i) break; // only one item in v

                    // - but now we need to go round once more to get

                    // v->s inspected. This looks messy, but is actually

                    // the optimal approach.

                    if (first\_key\_inspected) break;

                    first\_key\_inspected = true;

                }

            }

            while (true)

            {

                Among w = v[i];

                if (common\_i >= w.s.*Length*)

                {

                    cursor = c + w.s.*Length*;

                    if (w.method == null) return w.result;

                    bool res = false;

                    try

                    {

                        //Object resobj = w.method.Invoke(this);

                        //res = resobj.ToString().Equals("true");

                    }

                    catch (*Exception* e)

                    {

                        res = false;

                        // FIXME - debug message

                    }

                    cursor = c + w.s.*Length*;

                    if (res) return w.result;

                }

                i = w.substring\_i;

                if (i < 0) return 0;

            }

        }

        // find\_among\_b is for backwards processing. Same comments apply

        protected int find\_among\_b(*List*<Among> v)

        {

            int i = 0;

            int j = v.Count;

            int c = cursor;

            int lb = limit\_backward;

            int common\_i = 0;

            int common\_j = 0;

            bool first\_key\_inspected = false;

            while (true)

            {

                int k = i + ((j - i) >> 1);

                int diff = 0;

                int common = common\_i < common\_j ? common\_i : common\_j;

                Among w = v[k];

                int i2;

                for (i2 = w.s.*Length* - 1 - common; i2 >= 0; i2--)

                {

                    if (c - common == lb)

                    {

                        diff = -1;

                        break;

                    }

                    diff = current[c - 1 - common] - w.s[i2];

                    if (diff != 0) break;

                    common++;

                }

                if (diff < 0)

                {

                    j = k;

                    common\_j = common;

                }

                else

                {

                    i = k;

                    common\_i = common;

                }

                if (j - i <= 1)

                {

                    if (i > 0) break;

                    if (j == i) break;

                    if (first\_key\_inspected) break;

                    first\_key\_inspected = true;

                }

            }

            while (true)

            {

                Among w = v[i];

                if (common\_i >= w.s.*Length*)

                {

                    cursor = c - w.s.*Length*;

                    if (w.method == null) return w.result;

                    bool res = false;

                    try

                    {

                        //Object resobj = w.method.Invoke(this);

                        //res = resobj.ToString().Equals("true");

                    }

                    catch (*Exception* e)

                    {

                        res = false;

                        // FIXME - debug message

                    }

                    cursor = c - w.s.*Length*;

                    if (res) return w.result;

                }

                i = w.substring\_i;

                if (i < 0) return 0;

            }

        }

        /\* to replace chars between c\_bra and c\_ket in current by the

         \* chars in s.

         \*/

        protected int replace\_s(int c\_bra, int c\_ket, string s)

        {

            int adjustment = s.*Length* - (c\_ket - c\_bra);

            current = current.*Replace*(current.*ToString*(c\_bra, c\_ket - c\_bra), s);

            limit += adjustment;

            if (cursor >= c\_ket) cursor += adjustment;

            else if (cursor > c\_bra) cursor = c\_bra;

            return adjustment;

        }

        protected void slice\_check()

        {

            if (bra < 0 ||

                bra > ket ||

                ket > limit ||

                limit > current.*Length*)   // this line could be removed

            {

*Console*.*WriteLine*("faulty slice operation");

                // FIXME: report error somehow.

                /\*

                    fprintf(stderr, "faulty slice operation:\n");

                    debug(z, -1, 0);

                    exit(1);

                    \*/

            }

        }

        protected void slice\_from(string s)

        {

            slice\_check();

            replace\_s(bra, ket, s);

        }

        protected void slice\_del()

        {

            slice\_from("");

        }

        protected void insert(int c\_bra, int c\_ket, string s)

        {

            int adjustment = replace\_s(c\_bra, c\_ket, s);

            if (c\_bra <= bra) bra += adjustment;

            if (c\_bra <= ket) ket += adjustment;

        }

        /\* Copy the slice into the supplied StringBuffer \*/

        protected string slice\_to(string s)

        {

            slice\_check();

            int len = ket - bra;

            s = current.*ToString*(bra, ket - bra);

            return s;

        }

        protected string assign\_to(string s)

        {

            s = current.*ToString*(0, limit);

            return s;

        }

        public string Stem(string s)

        {

            try

            {

                this.setCurrent(s);

                lab0: do

                {

                    int v\_1 = cursor;

                    lab1: do

                    {

                        {

                            int v\_2 = cursor;

                            lab2: do

                            {

                                {

                                    int c = cursor + 4;

                                    if (0 > c || c > limit)

                                    {

                                        goto lab2\_end;

                                    }

                                    cursor = c;

                                }

                                goto lab1\_end;

                            } while (false);

                            lab2\_end:

                            cursor = v\_2;

                        }

                        goto lab0\_end;

                    } while (false);

                    lab1\_end:

                    cursor = v\_1;

                    lab3: do

                    {

                        int v\_3 = cursor;

                        lab4: do

                        {

                            if (!r\_exception1())

                            {

                                goto lab4\_end;

                            }

                            goto lab3\_end;

                        } while (false);

                        lab4\_end:

                        cursor = v\_3;

                        limit\_backward = cursor;

                        cursor = limit;

                        int v\_4 = limit - cursor;

                        lab5: do

                        {

                            int v\_5 = limit - cursor;

                            lab6: do

                            {

                                // call postfix, line 157

                                if (!r\_postfix())

                                {

                                    cursor = limit - v\_5;

                                    goto lab6\_end;

                                }

                            } while (false);

                            lab6\_end:

                            // or, line 158

                            lab7: do

                            {

                                int v\_6 = limit - cursor;

                                lab8: do

                                {

                                    // call adjective, line 158

                                    if (!r\_adjective())

                                    {

                                        goto lab8\_end;

                                    }

                                    goto lab7\_end;

                                } while (false);

                                lab8\_end:

                                cursor = limit - v\_6;

                                lab9: do

                                {

                                    // call verb, line 158

                                    if (!r\_verb())

                                    {

                                        goto lab9\_end;

                                    }

                                    goto lab7\_end;

                                } while (false);

                                lab9\_end:

                                cursor = limit - v\_6;

                                // call noun, line 158

                                if (!r\_noun())

                                {

                                    goto lab5\_end;

                                }

                            } while (false);

                            lab7\_end:

                            ;

                        } while (false);

                        lab5\_end:

                        cursor = limit - v\_4;

                        // do, line 160

                        int v\_7 = limit - cursor;

                        lab10: do

                        {

                            // call tidy\_up, line 160

                            if (!r\_tidy\_up())

                            {

                                goto lab10\_end;

                            }

                        } while (false);

                        lab10\_end:

                        cursor = limit - v\_7;

                        cursor = limit\_backward;

                    } while (false);

                    lab3\_end:

                    ;

                } while (false);

            }

            catch (*Exception* e)

            {

            }

            lab0\_end:

            return this.getCurrent();

        }

        private void AddCurrentWordToClassifier(ClassifierDto classif, string classifierName)

        {

            try

            {

                var curClassifier = \_allClassifierDtos.*First*(cl => cl.Name == classifierName);

                if (curClassifier != null)

                {

                    curClassifier.Words = curClassifier?.Words ?? new *List*<WordDto>();

                    curClassifier.Words.*Add*(new WordDto

                    {

                        Name = WordToStore

                    });

                    \_classifierService.Update(curClassifier);

                }

            }

            catch (*Exception* e)

            {

            }

        }

    }

}

StemmerOperations.cs

using *System*.*Text*;

using UkrainianStemmer.Services;

namespace UkrainianStemmer.StemmerLanguages

{

    public class StemmerOperations

    {

        //    // current string

        protected *StringBuilder* current;

        public string WordToStore { get; set; }

        protected int cursor;

        protected int limit;

        protected int limit\_backward;

        protected int bra;

        protected int ket;

        protected StemmerOperations()

        {

            current = new *StringBuilder*();

            setCurrent("");

        }

        //    /\*\*

        //     \* Set the current string.

        //     \*/

        protected void setCurrent(string value)

        {

            //           current.replace(0, current.length(), value);

            //current=current.Replace(current.ToString(), value);

            //current = StringBufferReplace(0, current.Length, current, value);

            //current = StringBufferReplace(0, value.Length, current, value);

            WordToStore = value;

            current.*Remove*(0, current.*Length*);

            current.*Append*(value);

            cursor = 0;

            limit = current.*Length*;

            limit\_backward = 0;

            bra = cursor;

            ket = limit;

        }

        //    /\*\*

        //     \* Get the current string.

        //     \*/

        protected string getCurrent()

        {

            string result = current.*ToString*();

            // Make a new StringBuffer.  If we reuse the old one, and a user of

            // the library keeps a reference to the buffer returned (for example,

            // by converting it to a String in a way which doesn't force a copy),

            // the buffer size will not decrease, and we will risk wasting a large

            // amount of memory.

            // Thanks to Wolfram Esser for spotting this problem.

            //current = new  StringBuilder();

            return result;

        }

        protected void copy\_from(StemmerOperations other)

        {

            current = other.current;

            cursor = other.cursor;

            limit = other.limit;

            limit\_backward = other.limit\_backward;

            bra = other.bra;

            ket = other.ket;

        }

        protected bool in\_grouping(char[] s, int min, int max)

        {

            if (cursor >= limit) return false;

            //           char ch = current.charAt(cursor);

            int ch = (int)current[cursor];

            if (ch > max || ch < min) return false;

            //           ch -= min;

            ch -= min;

            if ((s[ch >> 3] & (0X1 << (ch & 0X7))) == 0) return false;

            cursor++;

            return true;

        }

        protected bool in\_grouping\_b(char[] s, int min, int max)

        {

            if (cursor <= limit\_backward) return false;

            //           char ch = current.charAt(cursor - 1);

            int ch = (int)current[cursor - 1];

            if (ch > max || ch < min) return false;

            ch -= min;

            if ((s[ch >> 3] & (0X1 << (ch & 0X7))) == 0) return false;

            cursor--;

            return true;

        }

        protected bool out\_grouping(char[] s, int min, int max)

        {

            if (cursor >= limit) return false;

            //           char ch = current.charAt(cursor);

            int ch = (int)current[cursor];

            if (ch > max || ch < min)

            {

                cursor++;

                return true;

            }

            ch -= min;

            if ((s[ch >> 3] & (0X1 << (ch & 0X7))) == 0)

            {

                cursor++;

                return true;

            }

            return false;

        }

        protected bool out\_grouping\_b(char[] s, int min, int max)

        {

            if (cursor <= limit\_backward) return false;

            //           char ch = current.charAt(cursor - 1);

            int ch = (int)current[cursor - 1];

            if (ch > max || ch < min)

            {

                cursor--;

                return true;

            }

            ch -= min;

            if ((s[ch >> 3] & (0X1 << (ch & 0X7))) == 0)

            {

                cursor--;

                return true;

            }

            return false;

        }

        protected bool in\_range(int min, int max)

        {

            if (cursor >= limit) return false;

            //           char ch = current.charAt(cursor);

            int ch = (int)current[cursor];

            if (ch > max || ch < min) return false;

            cursor++;

            return true;

        }

        protected bool in\_range\_b(int min, int max)

        {

            if (cursor <= limit\_backward) return false;

            //           char ch = current.charAt(cursor - 1);

            int ch = (int)current[cursor - 1];

            if (ch > max || ch < min) return false;

            cursor--;

            return true;

        }

        protected bool out\_range(int min, int max)

        {

            if (cursor >= limit) return false;

            //           char ch = current.charAt(cursor);

            int ch = (int)current[cursor];

            if (!(ch > max || ch < min)) return false;

            cursor++;

            return true;

        }

        protected bool out\_range\_b(int min, int max)

        {

            if (cursor <= limit\_backward) return false;

            //           char ch = current.charAt(cursor - 1);

            int ch = (int)current[cursor - 1];

            if (!(ch > max || ch < min)) return false;

            cursor--;

            return true;

        }

        protected bool eq\_s(int s\_size, string s)

        {

            if (limit - cursor < s\_size) return false;

            int i;

            for (i = 0; i != s\_size; i++)

            {

                if (current[cursor + i] != s[i]) return false;

                //               if (current[cursor + i] != s[i]) return false;

            }

            cursor += s\_size;

            return true;

        }

        protected bool eq\_s\_b(int s\_size, string s)

        {

            if (cursor - limit\_backward < s\_size) return false;

            int i;

            for (i = 0; i != s\_size; i++)

            {

                //               if (current.charAt(cursor - s\_size + i) != s.charAt(i)) return false;

                if (current[cursor - s\_size + i] != s[i]) return false;

            }

            cursor -= s\_size;

            return true;

        }

        protected bool eq\_v(*StringBuilder* s)

        {

            return eq\_s(s.*Length*, s.*ToString*());

        }

        protected bool eq\_v\_b(*StringBuilder* s)

        {

            return eq\_s\_b(s.*Length*, s.*ToString*());

        }

        internal int find\_among(Among[] v, int v\_size)

        {

            int i = 0;

            int j = v\_size;

            int c = cursor;

            int l = limit;

            int common\_i = 0;

            int common\_j = 0;

            bool first\_key\_inspected = false;

            while (true)

            {

                int k = i + ((j - i) >> 1);

                int diff = 0;

                int common = common\_i < common\_j ? common\_i : common\_j; // smaller

                Among w = v[k];

                int i2;

                for (i2 = common; i2 < w.s\_size; i2++)

                {

                    if (c + common == l)

                    {

                        diff = -1;

                        break;

                    }

                    diff = current[c + common] - w.s[i2];

                    if (diff != 0) break;

                    common++;

                }

                if (diff < 0)

                {

                    j = k;

                    common\_j = common;

                }

                else

                {

                    i = k;

                    common\_i = common;

                }

                if (j - i <= 1)

                {

                    if (i > 0) break; // v->s has been inspected

                    if (j == i) break; // only one item in v

                    // - but now we need to go round once more to get

                    // v->s inspected. This looks messy, but is actually

                    // the optimal approach.

                    if (first\_key\_inspected) break;

                    first\_key\_inspected = true;

                }

            }

            while (true)

            {

                Among w = v[i];

                if (common\_i >= w.s\_size)

                {

                    cursor = c + w.s\_size;

                    if (w.method == null) return w.result;

                    //bool res;

                    //try

                    //{

                    //    Object resobj = w.method.invoke(w.methodobject,new Object[0]);

                    //    res = resobj.toString().equals("true");

                    //}

                    //catch (InvocationTargetException e)

                    //{

                    //    res = false;

                    //    // FIXME - debug message

                    //}

                    //catch (IllegalAccessException e)

                    //{

                    //    res = false;

                    //// FIXME - debug message

                    //}

                    //cursor = c + w.s\_size;

                    //if (res) return w.result;

                }

                i = w.substring\_i;

                if (i < 0) return 0;

            }

        }

        //    // find\_among\_b is for backwards processing. Same comments apply

        internal int find\_among\_b(Among[] v, int v\_size)

        {

            int i = 0;

            int j = v\_size;

            int c = cursor;

            int lb = limit\_backward;

            int common\_i = 0;

            int common\_j = 0;

            bool first\_key\_inspected = false;

            while (true)

            {

                int k = i + ((j - i) >> 1);

                int diff = 0;

                int common = common\_i < common\_j ? common\_i : common\_j;

                Among w = v[k];

                int i2;

                for (i2 = w.s\_size - 1 - common; i2 >= 0; i2--)

                {

                    if (c - common == lb)

                    {

                        diff = -1;

                        break;

                    }

                    //                   diff = current.charAt(c - 1 - common) - w.s[i2];

                    diff = current[c - 1 - common] - w.s[i2];

                    if (diff != 0) break;

                    common++;

                }

                if (diff < 0)

                {

                    j = k;

                    common\_j = common;

                }

                else

                {

                    i = k;

                    common\_i = common;

                }

                if (j - i <= 1)

                {

                    if (i > 0) break;

                    if (j == i) break;

                    if (first\_key\_inspected) break;

                    first\_key\_inspected = true;

                }

            }

            while (true)

            {

                Among w = v[i];

                if (common\_i >= w.s\_size)

                {

                    cursor = c - w.s\_size;

                    if (w.method == null) return w.result;

                    //boolean res;

                    //try

                    //{

                    //    Object resobj = w.method.invoke(w.methodobject,

                    //        new Object[0]);

                    //    res = resobj.toString().equals("true");

                    // }

                    //catch (InvocationTargetException e)

                    //{

                    //    res = false;

                    //    // FIXME - debug message

                    // }

                    //catch (IllegalAccessException e)

                    //{

                    //    res = false;

                    //    // FIXME - debug message

                    // }

                    //cursor = c - w.s\_size;

                    //if (res) return w.result;

                }

                i = w.substring\_i;

                if (i < 0) return 0;

            }

        }

        //    /\* to replace chars between c\_bra and c\_ket in current by the

        //     \* chars in s.

        //     \*/

        protected int replace\_s(int c\_bra, int c\_ket, string s)

        {

            int adjustment = s.*Length* - (c\_ket - c\_bra);

            //           current.replace(c\_bra, c\_ket, s);

            current = StringBufferReplace(c\_bra, c\_ket, current, s);

            limit += adjustment;

            if (cursor >= c\_ket) cursor += adjustment;

            else if (cursor > c\_bra) cursor = c\_bra;

            return adjustment;

        }

        private *StringBuilder* StringBufferReplace(int start, int end, *StringBuilder* s, string s1)

        {

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

            for (int i = 0; i < start; i++)

            {

                sb.*Insert*(sb.*Length*, s[i]);

            }

            //           for (int i = 1; i < end - start + 1; i++)

            //           {

            sb.*Insert*(sb.*Length*, s1);

            //           }

            for (int i = end; i < s.*Length*; i++)

            {

                sb.*Insert*(sb.*Length*, s[i]);

            }

            return sb;

            //string temp = s.ToString();

            //temp = temp.Substring(start - 1, end - start + 1);

            //s = s.Replace(temp, s1, start - 1, end - start + 1);

            //return s;

        }

        protected void slice\_check()

        {

            if (bra < 0 ||

                bra > ket ||

                ket > limit ||

                limit > current.*Length*)   // this line could be removed

            {

                //System.err.println("faulty slice operation");

                // FIXME: report error somehow.

                /\*

                    fprintf(stderr, "faulty slice operation:\n");

                    debug(z, -1, 0);

                    exit(1);

                    \*/

            }

        }

        protected void slice\_from(string s)

        {

            slice\_check();

            replace\_s(bra, ket, s);

        }

        protected void slice\_from(*StringBuilder* s)

        {

            slice\_from(s.*ToString*());

        }

        protected void slice\_del()

        {

            slice\_from("");

        }

        protected void insert(int c\_bra, int c\_ket, string s)

        {

            int adjustment = replace\_s(c\_bra, c\_ket, s);

            if (c\_bra <= bra) bra += adjustment;

            if (c\_bra <= ket) ket += adjustment;

        }

        protected void insert(int c\_bra, int c\_ket, *StringBuilder* s)

        {

            insert(c\_bra, c\_ket, s.*ToString*());

        }

        //    /\* Copy the slice into the supplied StringBuffer \*/

        protected *StringBuilder* slice\_to(*StringBuilder* s)

        {

            slice\_check();

            int len = ket - bra;

            //           s.replace(0, s.length(), current.substring(bra, ket));

            //           int lengh = string.IsNullOrEmpty(s.ToString())!= true ? s.Length : 0;

            //           if (ket == current.Length) ket--;

            //string ss = current.ToString().Substring(bra, len);

            //StringBufferReplace(0, s.Length, s, ss);

            //return s;

            return StringBufferReplace(0, s.*Length*, s, current.*ToString*().*Substring*(bra, len));

            //           return StringBufferReplace(0, lengh, s, current.ToString().Substring(bra, ket));

            //           return s;

        }

        //    /\* Copy the slice into the supplied StringBuilder \*/

        //protected StringBuilder slice\_to(StringBuilder s)

        //{

        //    slice\_check();

        //    int len = ket - bra;

        //    s.replace(0, s.length(), current.substring(bra, ket));

        //    return s;

        //}

        protected *StringBuilder* assign\_to(*StringBuilder* s)

        {

            //s.replace(0, s.length(), current.substring(0, limit));

            //return s;

            return StringBufferReplace(0, s.*Length*, s, current.*ToString*().*Substring*(0, limit));

        }

    }

}