РЕФЕРАТ

Розрахунково-пояснювальна записка складається з 9 розділів:

* вступ – в даному розділі описується сутність розробки, її актуальність. Складається з 2 сторінок;
* призначення, постановка задачі та огляд програмних аналогів і літератури – у цьому розділі описано призначення, постановка задачі, проведений огляд аналогів і літератури, а також аналіз існуючих систем. Складається з 30 сторінок;
* зовнішнє та логічне проектування – у цьому розділі проведений огляд вхідних і вихідних даних, формалізація задачі, розробка структур файлів і структури правил переходу. Складається з 5 сторінок;
* внутрішнє проектування – у цьому розділі приводиться опис об’єктно-орієнтованого проектування, проектування інтерфейсу користувача, ескізи форм, проектування динаміки системи, вибір мови програмування. Складається з 21 сторінки;
* тестування та налагодження – включає в себе вибір об’єкту та методів тестування, результати тестування та налагодження програми. Складається з 5 сторінок;
* безпека праці користувача при роботі на ЕОМ. Складається з 12 сторінок;
* висновки. Складається з 2 сторінок;
* список літератури – включає в себе бібліографічний список використаної літератури. Складається з 1 сторінки;
* додатки – містить технічне завдання і робочий проект.

Кількість таблиць: 23.

Кількість рисунків: 56.

ЗМІСТ

[1 Огляд проблеми 9](#_Toc1895409652)

[2 зовнішнє та логічне проектування 24](#_Toc999437375)

[2.1 Зовнішнє проектування 24](#_Toc189248429)

[2.1.1 Опис функціональних характеристик 24](#_Toc1980695222)

[2.1.2 Вхідні дані 25](#_Toc352118505)

*[Тут можно расписать про SPARQL со скришнотами (это же тоже входящие данные!)](#_Toc934016377)* [25](#_Toc934016377)

[3 ТЕСтування та налагодження програми 61](#_Toc987652236)

[3.1 Вибір стратегії тестування 61](#_Toc3838806)

[Для досягнення максимальної якості тестування під кожен метод або модуль програми, що тестується необхідно обрати найбільш вдалий метод або набір методів, що забезпечать необхідний результат. При цьому необхідно підібрати найкраще співвідношення між часом, що буде витрачено на тестування, та якістю тестування. Набір тестів повинен мати мінімальну збитковість, але при цьому максимально охоплювати функціональність системи [16]. 61](#_Toc154121645)

[4 Охорона праці та безпека у надзвичайних ситуаціях 84](#_Toc138230143)

[5.1 Аналіз шкідливих та небезпечних виробничих факторів 84](#_Toc1017232940)

[5.2 Організаційні і технічні заходи щодо захисту користувачів від шкідливих і небезпечних факторів. 87](#_Toc1539149625)

[5.3 Норми пожежної безпеки на робочому місці користувача 88](#_Toc1151805341)

[5.4 Вимоги безпеки праці 91](#_Toc646382045)

[5.5 Дії при аварійних ситуаціях 93](#_Toc468597703)

[5.5.1 Допомога при уражені електричним струмом 93](#_Toc398086492)

[висновки 96](#_Toc1657727114)

[Після закінчення розробки системи, була написана уся необхідна програмна документація. Наявність якісного керівництва користувача дозволить методистам учбового відділу, що будуть використовувати систему, швидко вивчити можливості програми та почати її використовувати. Задокументований опис та текст програми спростить подальше супроводження системи. 97](#_Toc59699604)

[ЛІТЕРАТУРА 98](#_Toc546868341)

[(old) ЛІТЕРАТУРА 99](#_Toc1170554059)

ЗМІСТ (old)

[Вступ 7](#_Toc421172491)

[1 Огляд проблеми 8](#_Toc421172492)

[1.1 Призначення та область застосування 8](#_Toc421172493)

[1.2 Постановка задачі 8](#_Toc421172494)

[1.3 Огляд програмних аналогів 9](#_Toc421172495)

[1.3.1 Особливості автоматизованої системи «Розклад занять» 9](#_Toc421172496)

[1.3.2 Особливості системи «Галактика. Управление ВУЗом» 10](#_Toc421172497)

[1.3.3 Особливості автоматизованої системи «Навантаження ВНЗ» 10](#_Toc421172499)

[1.3.4 Особливості АІСУ «Навчальне навантаження» 12](#_Toc421172500)

[1.4 Огляд літературних джерел 13](#_Toc421172500)

[2 Зовнішнє та логічне проектування 17](#_Toc421172500)

[2.1 Зовнішнє проектування 17](#_Toc421172501)

[2.1.1 Опис функціональних характеристик 17](#_Toc421172502)

[2.1.2 Вхідні дані 18](#_Toc421172503)

[2.1.3 Вихідні дані 28](#_Toc421172504)

[2.2 Проектування бази даних 29](#_Toc421172505)

[2.2.1 Специфікація вимог до бази даних 29](#_Toc421172506)

[2.2.2 Визначення типів сутностей та зв’язків між ними 29](#_Toc421172507)

[2.2.3 Логічне проектування бази даних](#_Toc421172508) 57

[2.2.4 Схема бази даних 59](#_Toc421172514)

[2.3 Вибір мови програмування](#_Toc421172515) [64](#_Toc421172515)

[2.3.1 Опис платформи .NET Framework 64](#_Toc421172515)

[2.3.2 Опис мови розробки програмного продукту – C# 68](#_Toc421172515)

[2.4 Об’єктно-орієнтоване проектування 69](#_Toc421172516)

[2.5 Розробка інтерфейсу користувача 78](#_Toc421172516)

[3 Тестування та налагодження програми 80](#_Toc421172500)

[3.1 Вибір стратегії тестування 80](#_Toc421172500)

[3.2 Тестування функціональності 80](#_Toc421172500)

[3.2.1 Тестування методу ReadDocFromExcel класу CurriculumDocService 80](#_Toc421172500)

[3.2.2 Тестування методу FormationDocs класу WorkingCurriculumDocService 96](#_Toc421172500)

[3.2.3 Тестування методу Formation класу LoadDepartmentService 98](#_Toc421172500)

[3.2.4 Результати тестування 99](#_Toc421172500)

[3.3 Налагодження програми 101](#_Toc421172500)

[4 Охорона праці та безпека у надзвичайних ситуаціях 103](#_Toc421172500)

[4.1](#_Toc421172500) [Аналіз шкідливих та небезпечних виробничих факторів 104](#_Toc421172500)

[4.2](#_Toc421172500) [Організаційні та технічні заходи по усуненню виявлених шкідливих та  
 небезпечних виробничих факторів 106](#_Toc421172500)

[4.2.1 Вимоги щодо забезпечення пожежної безпеки 107](#_Toc421172500)

[4.2.2 Вимоги до електробезпеки 108](#_Toc421172500)

[4.2.3 Вимоги до організації робочого місця 109](#_Toc421172500)

[4.3 Правила безпечного виконання робіт методистами навчального відділу  
 при роботі за комп’ютером 110](#_Toc421172500)

[4.4 Дії у разі настання надзвичайної ситуації 111](#_Toc421172500)

[4.4.1 Дії у разі виявлення пожежі 112](#_Toc421172500)

[Висновки 114](#_Toc421172500)

[Література 116](#_Toc421172500)

[Додатки 118](#_Toc421172500)

ВСТУП

Програмний засіб «Система доступу до енциклопедичних знань на природній мові» призначений для отримання енциклопедичних знань у режимі діалогу за допомогою природньої мови.

Причиною виникнення необхідності розробки є відсутність програми, яка б виконувала дану задачу на росіській мові з точністю, достатньою для ефективного повсякденного користування. Традиційно дана задача вирішувалася за допомогою звичайних пошукових систем та формулювання питання у вигляді ключових слів.

Головною метою розробки є спрощення складності та оптимізація процесу пошуку інформації із енциклопедичних джерел. Розроблюване програмне забезпечення дозволить використовувати енциклопедичну інформацію з навчальною або довідковою метою. У ролі користувача може виступати будь-яка людина, що має доступ до інтернету. Інтерфейс взаємодії з системою буде реалізований у вигляді веб-додатку через систему текстових або голосових запитів та відповідей. Це стане у нагоді як звичайним користувачам, так і користувачам з обмеженими можливостями або проблемами зору. До того ж це буде сприяти збільшенню інтересу до знань за рахунок використання перспективних технологій.

1 Огляд проблеми

1.1 Призначення та область застосування

Система доступу до енциклопедичних знань на природній мові орієнтована на широке коло користувачів та призначена спростити процес пошуку нової інформації енциклопедичного характеру.

Функціональним призначенням системи є надання користувачу можливості за текстовим або голосовим запитом отримати інформацію, що міститься в енциклопедичних системах.

Експлуатаційним призначенням програмного комплексу є надання швидкого доступу до енциклопедичних знань у навчанні та роботі; надання альтернативного мовного інтерфейсу для людей з проблемами зору; збільшення інтересу до знань за рахунок використання перспективних технологій.

1.2 Постановка задачі

Необхідно проаналізувати існуючі відкриті бази знань, принципи їх формування, можливості інтеграції з ними та запропонувати алгоритм пошуку найбільш релевантної відповіді.

Необхідно розробити програмний продукт для автоматичного пошуку відповіді на питання енциклопедичного характеру, що поставлене російською мовою. До функціональних характеристик програмного продукту висунуті наступні вимоги. Програма повинна:

* мати можливість вводу запиту на природній мові у форматі текстового або голосового повідомлення;
* надавати супровідні матеріали в якості довідки до знайденої відповіді (короткий текст та картинка);
* збирати інформацію, надану користувачем, щодо правильності знайденої відповіді за бінарною шкалою (“так” або “ні”);
* відображати приклад вдалого запиту для початкової орієнтації користувача у системі.

1.3 Огляд програмних аналогів

Досить часто системи відповідей на запитання інтегровані у пошукові системи для того, щоб доповнити набір релевантних посилань повністю готовими відповідями на запитання. Характерною рисою таких аналогів є те, що такі системи є комерційними та закритими, а тому можуть не мати у вільному доступі достатньої кількості інформації щодо деталей їх реалізації. З іншої сторони, існують дослідницькі проекти, які ставлять на меті дослідити питання систем відповідей на запитання та надають розгорнуту інформацію щодо їх функціонування, сильних сторін та недоліків.

1.3.1 Особливості фрейму відповідей пошукової системи «Google»

“Google” є беззаперечним технологічним лідером і найпопулярнішою в світі пошуковою системою з часткою ринку 62.4% (на 2012 рік) [1]. Окрім стандартних результатів пошукової видачі у вигляді веб-посилань, використовуються і фрейми готових відповідей (рис. 1.1) [2].

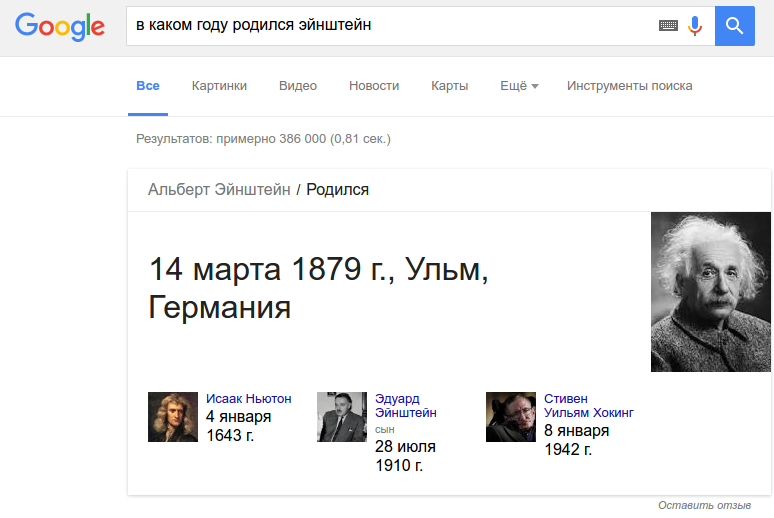


Рисунок 1.1 − Фрейм відповідей пошукової системи Google на питання “В каком году родился Эйнштейн”

Варто звернути увагу, що на питання “В каком году родился Эйнштейн” система відповідає не тільки роком народження, а взагалі всією інформацію, що міститься під знайденою властивістю “Родился”, а саме: дата народження, місто та країна.

Відповіддю на запитання також може бути і список із атомарних значень, як наприклад питання “Кто был научным руководителем Эйнштейна” (рис. 1.2) містить імена двох людей, що були науковими керівниками Ейнштейна.

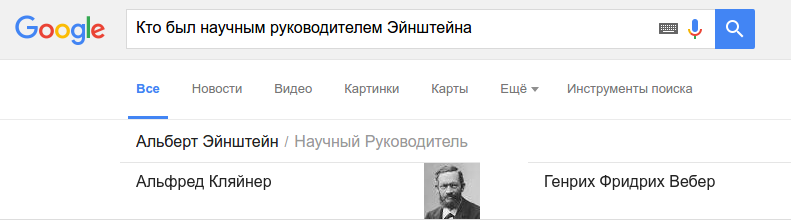


Рисунок 1.2 − Фрейм відповідей пошукової системи Google на питання “Кто был научным руководителем Эйнштейна”

Проте не на всі питання система Google може знайти відповідь. Наприклад, питання “Какой почтовый код Днепропетровска” залишається без фрейму відповіді (рис. 1.3).

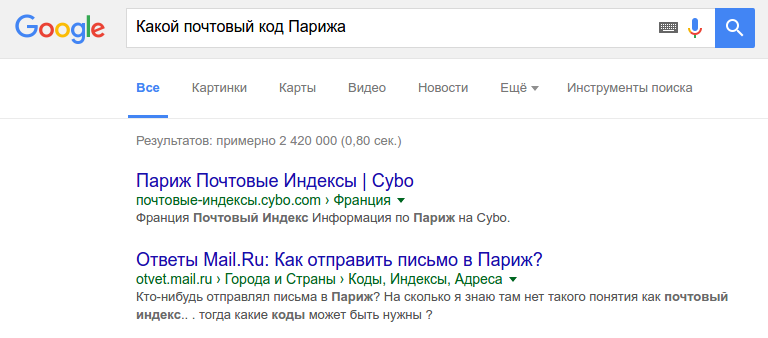


Рисунок 1.3 − Фрейм відповідей пошукової системи Google на питання “Какой почтовый код Днепропетровска”

В цілому, можна відмітити, що для більшості стандартних питань фрейм відповідей пошукової системи Google відображає правильні відповіді.

1.3.2 Особливості фрейму відповідей пошукової системи «Yandex»

“Yandex” є найпопулярнішою пошуковою системою у пострадянських країнах з часткою ринку в 57.5% (жовтень 2015 року) [3]. У пошуковій видачі ця пошукова система також використовує фрейми готових відповідей (рис. 1.4) [4].

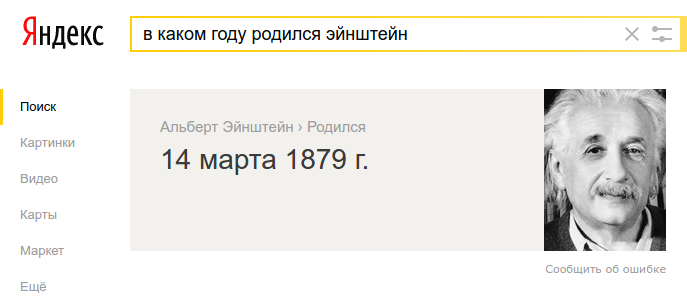


Рисунок 1.4 − Фрейм відповідей пошукової системи Yandex на питання “В каком году родился Эйнштейн”

Yandex містить більш детальний опис цього фрейму відповіді, що називається “Колдунщик” у розділі “Допомога” на сайті [5]. “Колдунщик” - це елементи пошукової видачі, які відповідають на запитання прямо на сторінці з результатами пошуку. Це може бути інформація з Вікіпедії, картинка, переклад слова, розклад електричок тощо.

Підтримуються наступні типи фреймів [5]:

* авто;
* адреси;
* афіша;
* вакансії;
* відео;
* вікіпедія;
* час;
* картинки;
* карти;
* конвертер;
* котирування;
* маркет;
* музичний плеєр;
* новини;
* відключення води;
* погода;
* психологічна допомога;
* розклади;
* вірші;
* телепрограма;
* IP-адреса користувача.

Енциклопедичний характер інформації містять тільки фрейми типів “вікіпедія” та “адреси”.

Звичайно, система Yandex не здатна знайти відповідь на всі питання. Наприклад, питання “Кто был научным руководителем Эйнштейна”, на яке Google успішно знайшов відповідь, залишається без фрейму відповіді у випадку системи Yandex (рис. 1.5).

Не зважаючи на те, що пошукова система Yandex більше орієнтована на російську мову, аніж Google, можна відмітити, що фрейм відповідей пошукової системи Yandex працює менш точно, аніж фрейм відповідей Google.

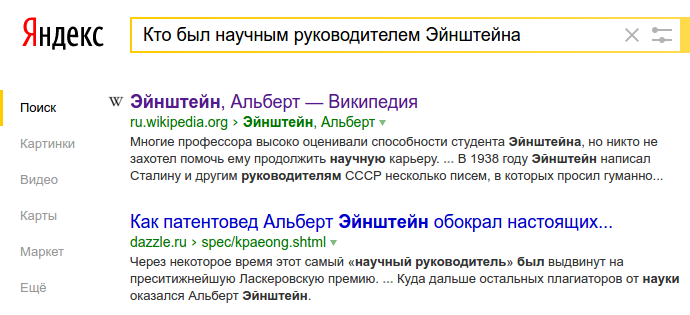


Рисунок 1.5 − Результат пошукової системи Yandex на питання “Кто был научным руководителем Эйнштейна”

1.3.3 Особливості системи відповідей «START: Natural Language Question Answering System»

Ця система розроблена у лабораторії Computer Science and Artificial Intelligence Laboratory (CSAIL) Масачусетського технологічного інституту і є частковим аналогом, оскільки відповідає на питання, задані тільки англійською мовою [6].

Варто звернути увагу, що на питання “In which year was Einstein born?” система відповідає не роком, а датою народження (рис. 1.6).

У питанні, відповіддю на яке може бути список із атомарних значень, як наприклад питання “Who was doctoral advisor of Einstein?” (рис. 1.7) містить імена не двох людей, що були науковими керівниками Ейнштейна, а тільки одного, проте головного. Це відрізняється від відповіді системи Google, яка надає весь список наукових керівників.

Також у системі START реалізований механізм знаходження неоднозначності у питанні. Наприклад у питанні “What is postal code of Paris?” слово “Paris” може позначати близько 10 різних міст (рис. 1.8). В такому разі система намагається дізнатися у користувача, яке саме місто він мав на увазі.

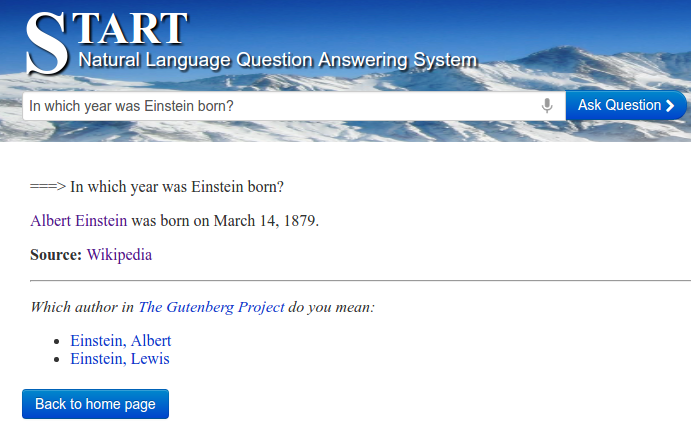


Рисунок 1.6 − Результат системи START на питання “In which year was Einstein born?”

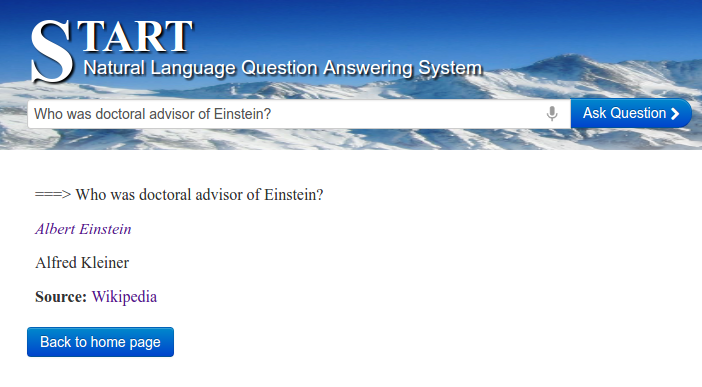


Рисунок 1.7 − Результат системи START на питання “Who was doctoral advisor of Einstein?”

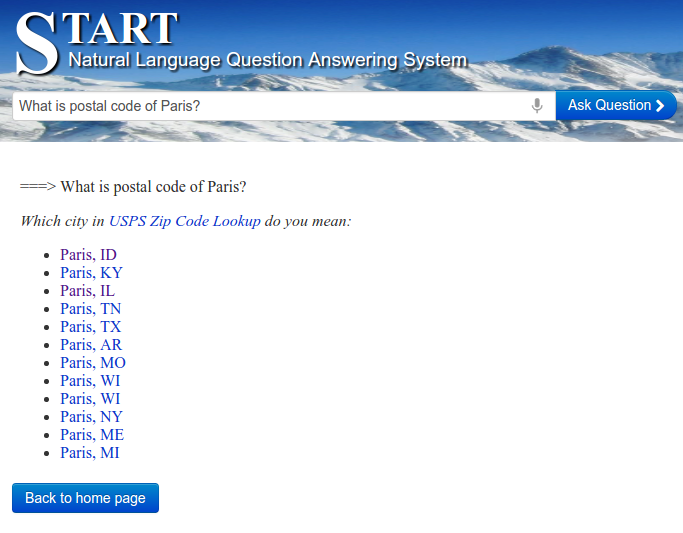


Рисунок 1.8 − Результат системи START на питання “What is postal code of Paris?”

В цілому, після серії питань та відповідей можна зробити висновок, що система START працює краще аналогічних систем Google та Yandex. Певно, це пов’язано з тим, що система START вирішує тільки одну задачу, навідміну від комплексних рішень для пошуку Google та Yandex. До того ж, START працює з англійською мовою, на якій існує набагато більше енциклопедичних матеріалів, що можуть використовуватися системою.

1.4 Огляд літературних джерел

Під час виконання аналізу літератури для розробки продукту, були розглянуті наступні питання: зберігання енциклопедичних даних, відкриті бази знань, проектування серверної та клієнтської частини.

Перш, ніж почати детальний огляд літератури щодо аналізу систем обробки даних, необхідно дати визначення основним поняттям, які відносяться до предметної області, що розглядається.

Семантична павутина (англ. Semantic web) — нова концепція розвитку Всесвітньої павутини і мережі Інтернет, яка створена і впроваджується Консорціумом Всесвітньої павутини (англ. World Wide Web Consortium, W3C). Інші назви — семантичний веб, семантична мережа. Хоча поняття семантична мережа, яке виникло раніше, породило поняття семантична павутина, їх слід відокремлювати. Концепція полягає у впровадженні спільних, стандартних форматів данних у Мережі. Для заохочення впровадження семантичного форматування сторінок, пропонується змінювати структуру вже існуючих, не структурованих чи частково-структурованіх сторінок у «мережу данних». Створення семантичниної Мережі полягає у застосуванні середовища опису ресурсів [7].

RDF — Розроблена консорціумом W3C технологія семантичної павутини, яка включає в себе середовище опису ресурсів (англ. Resource Description Framework, RDF), визначає загальну архітектуру метаданих і призначена для забезпечення сумісності метаданих за допомогою спільної семантики, структури та синтаксису. Технологія семантичної мережі передбачає розширення можливостей інтернету завдяки механізмам надання інформації чітко визначеного значення, яке дозволяє ефективно використовувати її у спільній роботі як комп'ютерів, так і людей [8].

SPARQL (рекурсивний акронім від англ. SPARQL Protocol and RDF Query Language) — мова запитів до даних, представлених по моделі RDF, а також протокол для передачі цих запитів і відповідей на них. SPARQL є рекомендацією консорціуму W3C і одною з технологій семантичної павутини [9].

Big Data (великі дані) – це набори даних такого об'єму, що традиційні інструменти не здатні здійснювати їх охоплення, управління та обробку за помірний час. Важливо також відзначити те, що під терміном Big Data у різних контекстах можуть мати на увазі дані великого об'єму, технології їх обробки, проекти, компанії, які активно використовують дану технологію [3].

До основних переваг використання технології Big Data можна віднести:

* отримання якісно нових знань за рахунок комплексного аналізу усієї інформації у єдиному аналітичному сховищі.
* розширення функціональності існуючих інформаційних систем підтримки бізнесу.
* збільшення ефективності використання апаратних ресурсів серверів;
* забезпечення мінімальної вартості використання всіх видів інформації за рахунок можливості використання ПЗ з відкритим кодом і хмарних технологій зберігання інформації.

Аналіз даних – область математики та інформатики, що займається побудовою і дослідженням найбільш загальних математичних методів і обчислювальних алгоритмів вилучення знань з експериментальних (в широкому сенсі) даних; процес дослідження, фільтрації, перетворення і моделювання даних з метою вилучення корисної інформації та прийняття рішень. Аналіз даних має безліч аспектів і підходів, охоплює різні методи в різних областях науки і діяльності.

Інтелектуальний аналіз даних – це особливий метод аналізу даних, який фокусується на моделюванні і відкритті даних, а не на їх опісаніі.Бізнес-аналітика охоплює аналіз даних, який покладається на агрегацію. У статистичному сенсі деякі поділяють аналіз даних наопісательную статистику, дослідницький аналіз даних і перевірку статистичних гіпотез. Дослідницький аналіз даних займається відкриттям нових характеристик даних, а перевірка статистичних гіпотез на підтвердження або спростування існуючих гіпотез. Прогнозний аналізфокусіруется на застосуванні статистичних або структурних моделей для передбачення або класифікації, а аналіз тексту застосовує статистичні, лінгвістичні і структурні методи для вилучення і класифікації інформації з текстових джерел належать до неструктурованих даними. Все це різновиди аналізу даних [4].

1.4.1 Зберігання енциклопедичних даних

Насьогодні комп'ютери беруть досить обмежену участь у формуванні й обробці інформації в мережі Інтернет. Функції комп'ютерів в основному зводяться до збереження, відображення і пошуку інформації. У той же час створення інформації, її оцінка, класифікація й актуалізація — усе це як і раніше виконує людина. Як включити комп'ютер у ці процеси? Якщо комп'ютер поки не можна навчити розуміти людську мову, то потрібно використовувати мову, що була б зрозумілою комп'ютеру. Тобто, в ідеальному варіанті, вся інформація в Інтернеті повинна розміщуватись двома мовами: людською мовою для людини і комп'ютерною мовою для розуміння комп'ютера. Семантична павутина — це концепція мережі, у якій кожен ресурс людською мовою був би доповнений описом, зрозумілим комп'ютеру.

Для створення зрозумілого комп'ютеру опису ресурсу в семантичній павутині використовується формат RDF (англ. Resource Description Framework), що заснований на синтаксисі XML і використовує ідентифікатори URI для позначення ресурсів. RDF був затверджений як стандарт W3C у лютому 2004 року. RDF — це система опису мережних ресурсів, зрозуміла комп'ютеру. Формат RDF призначений для збереження метаданих (метадані — це дані про дані). Відповідно до концепції семантичної павутини, опис у форматі RDF повинен прикріплюватися до кожного мережного ресурсу. Документи RDF повинні оброблятися комп'ютером автоматично, RDF не призначений для читання і використання людиною. На сьогодні формат RDF вже сформувався й одержав широке поширення, він служить каркасом для створення семантичної павутини. RDFS (англ. RDF Schema) — це важлива надбудова над RDF, що дозволяє створювати класи і властивості (як в об'єктно-орієнтованому програмуванні).

Наступним важливим напрямком концепції семантичної павутини є мова OWL (англ. Web Ontology Language, вимовляється [‘оул]), що стала Рекомендацією W3C у лютому 2004 року. Ця мова побудована на форматах RDF і RDFS, вона призначена для обробки інформації в мережі. Мова OWL має 3 ступені деталізації, що є новим словом у комп'ютерних технологіях. Вона також легко масштабується й узгоджується з найсучаснішими мережними стандартами. У 2008 році було прийнято новий стандарт OWL 2. Теоретичною основою OWL є Описова логіка.

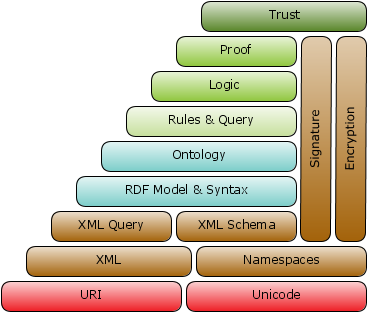


Рисунок 1.9 − Стек понять семантичної павутини

Представлення SPARQL-точок доступу (SPARQL endpoint) є рекомендованою практикою при публікації даних у всесвітній павутині. Загальна схема SPARQL-запиту виглядає наступним чином:

PREFIX foo: <http://example.com/resources/>

# префіксні оголошення

FROM ...

# джерела запиту

SELECT ...

# пункт результату

WHERE {...}

# критерії запиту

ORDER BY ...

# модифікатори запиту

Префіксні оголошення слугують для скорочення універсальних ідентифікаторів ресурсу. Джерела запиту визначають, які RDF графи запитуються. Пункт результату повертає набір даних (вибірку), які задовольняють умові. Критерії запиту визначає, що запитувати в базовому наборі даних. Модифікатори запиту обмежують, упорядковують, і інакше перетворюють результати запиту.

SPAQRL дозволяє користувачам писати глобально однозначні запити. Наприклад, наступний запит повертає імена і адреси кожної людини у світі:

PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name ?email

WHERE {

?person a foaf:Person.

?person foaf:name ?name.

?person foaf:mbox ?email.

}

Приведені параметри використовуються для опису людини, включеної до FOAF. Це ілюструє бачення Семантичної павутини як єдиної великої бази даних. Кожний ідентифікатор в SPARQL, URI, глобально однозначний, на відміну від «e-mail» або «email», які звичайно використовуються в SQL.

Цей запит може бути розподілений на декілька кінцевих точок SPARQL, різних комп'ютерів, і сбір результатів здійснюється процедурою, відомою як федеративний пошук.

SPARQL-endpoint або точка доступу — це служба сумісна з SPARQL. SPARQL-endpoint дозволяє користувачу запросити базу знань (вводити його запити). Запит обробляється і повертає результат в різних форматах. Таким чином, SPARQL точки доступу задуманий як сервіс, який забезпечує нормальний інтерфейс до бази знань SPARQL-endpoint.

Розрізняють два види точок доступу: загального призначення і локального.

Точки доступу загального призначення можуть виконувати запити по будь-яким вказаним RDF-документам, які знаходяться в Мережі. А локальні точки доступу можуть отримувати данні тільки від одного ресурсу.

Користуючись стандартом SPARQL можна інтегрувати різні бази знань. Один з найповніших списків SPARQL точок доступу представлений за посиланням [10].

1.4.2 Проектування веб-серверу

Веб-сервер – сервер, що приймає HTTP-запити від клієнтів, зазвичай веб-браузерів, і видає їм HTTP-відповіді, як правило, разом з HTML-сторінкою, зображенням, файлом, медіа-потоком або іншими даними.

Веб-сервером називають як програмне забезпечення, яке виконує функції веб-сервера, так і безпосередньо комп'ютер, на якому це програмне забезпечення працює.

Клієнт, яким зазвичай є веб-браузер, передає веб-серверу запити на отримання ресурсів, позначених URL-адресами. Ресурси – це HTML-сторінки, зображення, файли, медіа-потоки або інші дані, які необхідні клієнту. У відповідь веб-сервер передає клієнту запитані дані. Цей обмін відбувається по протоколу HTTP.

Веб-сервери можуть мати різні додаткові функції, наприклад:

* автоматизація роботи веб-сторінок;
* ведення журналу звернень користувачів до ресурсів;
* аутентифікація і авторизація користувачів;
* підтримка динамічно генеруються сторінок;
* підтримка HTTPS для захищених з'єднань з клієнтами.

Проектування корпоративного веб додатки, як і будь-якого іншого застосування, варто почати з визначення початковий цілей і області вирішуваних завдань. Створити реєстр зацікавлених осіб.

На наступному етапі необхідно зібрати вимоги до додатка, яке необхідно розробити. Уточнити цілі і область вирішуваних завдань і побудувати ієрархічну структуру робіт. Розглянемо окремо завдання побудови ієрархічної структури робіт. Кожний web-додаток можна представити в наступному вигляді:

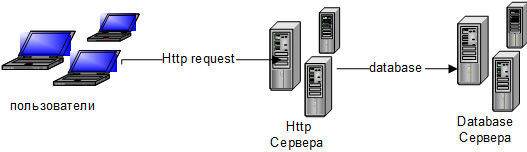


Рисунок 1.10 – Типовий приклад схеми роботи веб-сервера

Іншими словами, кожне web-додаток відправляє http запити на web-сервер для отримання корисних даних. Програма під керуванням web-сервера використовує ту чи іншу модель для зберігання даних. У сучасному світі найчастіше використовуються бази даних, SQL або NoSQL. Формально кожна web-додаток можна розбити на 3 взаємно незалежні частини:

* модуль, який виповнюється WEB-браузером. Ця програма може бути написано на будь-якій мові, який підтримує браузер. Найчастіше використовується мова JavaScript, як найбільш підтримуваний і має велику бібліотечну підтримку. Це дуже важливо, тому що дозволяє істотно економити бюджети проектів.
* модуль, що виконується на стороні сервера під керуванням web-сервера. Ця програма може бути написано на будь-якій мові, інтерпретацію якого підтримує обраний Вами web-сервер. Останнім часом, часто, в якості мови програмування вибирається мова Java. Ця мова також має серйозну бібліотечну підтримку.
* база даних. У цій області так само існує досить широкий вибір. Є промислові бази даних, такі як Oracle, DB2, PostgreSQL. Є легкі бази даних, такі як MySQL. База даних вибирається грунтуючись на цілях і області вирішуваних завдань.

1.4.3 Проектування веб-інтерфейсів

Веб-інтерфейс – це сукупність засобів, за допомогою яких користувач взаємодіє з веб-сайтом або веб-додатком через браузер. Веб-інтерфейси отримали широке поширення у зв'язку із зростанням популярності всесвітньої павутини і відповідно з широкого та швидкого розповсюдження веб-браузерів[1].

Завдяки веб-сайтам, які прийшли на заміну стаціонарно встановленим програмам, зникла потреба окремо встановлювати програмне забезпечення на комп’ютер користувача, замість цього потрібно встановити лише веб-браузер і вказати адресу сервера, на якому знаходиться потрібний користувачу сайт. Це дозволяє заощадити час та пам’ять комп'ютера.

Але у веб-додатків є і свої обмеження. Перш за все, вони працюють у контексті веб-переглядача та не мають доступу до файлової системи користувача. Відсутність можливості внести зміну у систему пов’язані з забезпеченням безпеки персональних даних користувача. Вдруге, вони обмежені у використанні пам’яті та ресурсах процесора. Саме з цих причин багато з існуючих програм досі залишаються у вигляді додатків, що встановлюються стаціонарно. Більшість з цих програм, або потребує доступу до файлової системи (різні текстові редактори, антивіруси, тощо), або висувають занадто високі вимоги до використання оперативної пам’яті та ресурсів процесора (комп’ютерні ігри, графічні та відео редактори, тощо). Але з часом навіть таке програмне забезпечення “перебирається” у більш зручний та швидкий веб формат. Пошуковий гігант Google навіть представив користувачам спеціальну операційну систему під назвою Chrome OS, яка не надає користувачу нічого крім можливості користуватися браузером. Серед переваг представленої операційної системи є швидкість роботи та зручність у користуванні.

Оскільки жодний веб-інтерфейс не мав би сенсу без веб-браузера, то і розробку кожного інтерфейсу потрібно розглядати у рамках взаємодії між клієнтом (браузером) та сервером. У основі цієї взаємодії лежить стандарт HTTP.

HTTP – протокол передачі даних, що використовується в комп’ютерних мережах. Назва скорочена від Hyper Text Transfer Protocol, протокол передачі гіпер-текстових документів. HTTP належить до протоколів моделі OSI 7-го прикладного рівня.

Основним призначенням протоколу HTTP є передача веб-сторінок (текстових файлів з розміткою HTML), хоча за допомогою нього успішно передаються і інші файли, які пов’язані з веб-сторінками (зображення і додатки), так і не пов’язані з ними[2].

Використовуючи протокол HTTP, браузер посилає запит за указаною адресою, отримуючи у відповідь дані для відображення. Зазвичай ці дані складаються з трьох частин:

HTML – мова, якою описується розмітка веб-сторінки. За допомогою тегів, ця мова повідомляє браузеру, які елементи будуть зображені на сторінці та у якому порядку;

CSS – описує зовнішній вид сторінки, задає стилі для елементів, які були описані у HTML;

Javascript – мова програмування, яка описує сценарії та поведінку веб-сторінки. Саме завдяки появленню мови Javascript, сторінки перестали бути статичними носіями інформації, а перетворились у динамічні програми, задовільняючи, час від часу, зростаючим потребам користувачів.

Після того як браузер отримає відповідь з серверу, він почне процес відображення сторінки сайту та виконання сценаріїв JavaScript. Але такий формат отримання даних не підходить для роботи динамічного програмного інтерфейсу, тому що щоб запросити нові дані з серверу, браузеру необхідно посилати ще один запит та починати процес відображення сторінки спочатку. Такий підхід, незважаючи на свою розповсюдженість, має великий недолік – інтерфейс працює повільно та недостатньо інтерактивно, тому що кожен раз щоб оновити інформацію, користувач змушений чекати доки вся сторінка оновиться.

Тому для вирішення проблеми з оновленням сторінки, розробники веб-браузерів додали до основного функціоналу можливість відправляти асинхронні запити (AJAX) до серверу без перезавантаження сторінки засобами JavaScipt. Це рішення викликало справжню революцію у проектуванні веб-інтерфейсів та запровадило новий принцип роботи сайту – Single page application (SPA).

SPA – з англійської, односторінковий додаток, цей термін було обрано для програмних інтерфейсів, які одноразово надходили до браузера, а інші дані, потрібні під час виконання, запитували асинхронно за допомогою AJAX. Цей підхід вирішив проблему низької інтерактивності веб-інтерфейсів та дозволив створювати більш зручні та динамічні веб-сторінки.

2 зовнішнє та логічне проектування

2.1 Зовнішнє проектування

2.1.1 Опис функціональних характеристик

Програмний комплекс призначений для доступу до енциклопедичних знань на природній мові. Також надає необхідну інформацію для складання розкладу, а саме перелік: викладачів, дисциплін, аудиторій, груп, вимоги викладачів, тощо.

Програма повинна:

* мати можливість вводу запиту на природній мові у форматі текстового або голосового повідомлення;
* надавати супровідні матеріали в якості довідки до знайденої відповіді (короткий текст та картинка);
* збирати інформацію, надану користувачем, щодо правильності знайденої відповіді за бінарною шкалою (“так” або “ні”);
* відображати приклад вдалого запиту для початкової орієнтації користувача у системі.

2.1.2 Вхідні дані

Вхідними даними є:

* запит у форматі тексту, що є назвою певної сутності або має структуру питального речення російською мовою, яке починається із питальної конструкції;
* аналогічний запит у форматі аудіозапису;
* зворотний зв’язок у вигляді оцінки правильності відповіді системи за бінарною шкалою (“так” або “ні”);
* інформація зі сторонніх енциклопедичних систем, а саме: короткий текстовий опис та картинка.

2.1.3 Вихідні дані

Результатом роботи програми є наступні вихідні дані:

* відображення голосового запиту у текстовому вигляді;
* відповідь на запит у вигляді тексту;
* рисунок, який доповнює відповідь на запит;
* озвучення тексту відповіді на запит;
* статистика результатів оцінювання якості відповідей, що містить інформацію про відсоток правильних відповідей та загальну кількість оцінок;
* запит пошуку сутностей до енциклопедичних систем;
* запит до енциклопедичних систем у форматі SPARQL.

2.1.4 Формалізація задачі

Формалізація задачі на рівні зовнішнього проектування представлена у вигляді діаграми варіантів прецеднтів (варіантів використання програми) (рис. 2.1).

Діаграма варіантів використання користувачем (2)

Рисунок 2.1 – Діаграма варіантів використання для актора «Користувач»

Користувач представлений у вигляді одного актора (Actor), взаємодія з системою представлена за допомогою варіантів використання (Use Case). У програмі «Система доступу до енциклопедичних знань на природній мові» у якості актора виступатиме кінцевий користувач системи. Варіанти використання надають опис можливостей, які система надає актору.

Актор «Користувач» може виконувати наступні дії:

* задання питання голосом;
* задання питання текстом;
* задання питання за запропонованим прикладом;
* оцінка правильності відповіді.

На діаграмах використан тип відношення між варіантами використання та актором “асоціація”, що відображається лінією зі стрілкою між актором і варіантом використання.

2.2 Проектування бази даних

2.2.1 Специфікація вимог до бази даних

Програма використовує спільну базу даних з системою складання розкладу занять і може імпортувати її дані, тому структура має бути ідентична.

2.2.2 Визначення типів сутностей та зв’язків між ними

Типи сутностей бази даних, що використовуються у системі наведені в табл. 2.21.

2.2.3 Логічне проектування бази даних

Етап логічного проектування бази даних передбачає наступні дії [9]:

1. перетворення концептуальної моделі даних в логічну:
2. видалення зв’язків типу M:N;
3. видалення складних зв’язків;
4. видалення рекурсивних зв’язків;
5. видалення зв’язків, що мають атрибути;
6. видалення множинних атрибутів;
7. перевірка зв’язків типу 1:1;
8. видалення надлишкових зв’язків.
9. визначення набору відношень;
10. перевірка моделі за допомогою правил нормалізації;
11. створення і перевірка логічної моделі даних.

Видалення зв’язків типу M:N **–** проводиться видалення зв’язків, коли багатьом записам таблиці А відповідає багато записів таблиці Б, які є в концептуальній моделі.

В концептуальній моделі даних такі зв’язки відсутні.

Видалення складних зв’язків **–** проводиться видалення не бінарних зв’язків, які є в концептуальній моделі.

В концептуальній моделі даних складні зв’язки відсутні.

Видалення рекурсивних зв’язків **–** проводиться видалення зв’язків між окремими екземплярами сутності одного і того ж типу.

В концептуальній моделі даних рекурсивні зв’язки відсутні.

Видалення множинних атрибутів **–** множинним атрибутом називається атрибут, який має декілька значень для однієї сутності.

В концептуальній моделі даних множинні атрибути відсутні.

В концептуальній моделі даних зв’язки типу 1:1 відсутні.

Нормалізація – це розбивка таблиці на відношення, що не будуть мати аномалії відновлення, включення і видалення даних [9].

Проведемо перевірку розробленої логічної моделі даних на відповідність її до форми НФБК.

НФ1 – відношення знаходиться в НФ1, якщо в ньому на перетині кожного рядка і кожного стовпця міститься тільки одне значення.

У розроблених відношеннях відсутні такі комірки в таблицях, щоб мали більше одного значення на перетині рядка і стовпчику.

НФ2 – відношення знаходиться в НФ2, якщо воно знаходиться в НФ1 і кожен атрибут якого, що не входить до складу первинного ключа, характеризується повною функціональною залежністю від цього первинного ключа.

У розроблених сутностях первинні ключі складаються із одного атрибуту, тому відношення відповідають НФ2.

НФ3 – відношення знаходиться в НФ3, якщо воно знаходиться в НФ1 і НФ2, і не має не вхідних у первинний ключ атрибутів, які б знаходилися в транзитивній функціональній залежності від первинного ключа.

Кожна розроблена сутність має одну функціональну залежність, тому транзитивної залежності між атрибутами сутності не може бути.

Нормальна форма Бойса – Кодда (НФБК) враховує функціональні залежності, у яких беруть участь усі потенційні ключі відносини, а не тільки його первинний ключ.

Для відношення з єдиним потенційним ключем його НФ3 і НФБК є еквівалентними.

Відношення знаходиться в НФБК тоді і тільки тоді, коли кожен його детермінант є потенційним ключем.

Кожна сутність має один потенціальний ключ, який є первинним ключем, тому, в даному випадку, НФ3 і НФБК є еквівалентними.

2.2.4 Схема бази даних

Кожна таблиця належить до групування об’єктів всередині бази даних, котру називають схемою бази даних. Схема бази даних – це іменований контейнер (простір імен) котрий можна використовувати для того щоб згрупувати таблиці і інші об’єкти бази даних.

Для таблиць схема бази даних дозволяє таблицям з однією назвою належати різним схемам.

Схема бази даних (рис. 2.58 – 2.61) відображає таблиці і зв’язки між ними.

Так як кількість відношень бази даних є достатньо великою, то схема представлена чотирма пакетами відношень, для більш зручного аналізу.

2.3 Вибір мови програмування

Обираючи мову програмування необхідно проаналізувати задачі, що ставляться перед розробником та можливі проблеми, що можуть виникнути під час розробки.

Перш за все для реалізації поставленої задачі потрібно звернути увагу на мови програмування, що дозволяють проводити ефективну обробку тексту. Під ефективністю мається на увазу наявність необхідних бібліотек для типових задач, наприклад:

* виділення лексем з тексту;
* морфологічна нормалізація слів;
* визначення частин мови слів;
* виділення іменованих сутностей.

Ще однією задачою є розробка веб-сервера це, мабуть, одна з найбільш поширинеш задач у сучасному програмуванні, та з розширенням та популірізацією мережі інтернет, потреба у програмному забезпечені цього типу тільки зростає. Для реалізації задач які виконуються веб-серверами існує багато технологій, кожна з яких має свою спеціалізацію та призначена для різних типів задач. Перелік найбільш пошириних з них:

* PHP;
* C#;
* Java;
* Scala;
* Python.

В якості мови програмування було обрано мову Python версії 3.5.1, що є сучасною і стабільною версією. Перш за все тому, що Python має бібліотеки NLTK та PyMorphy2.

В якості середовища для розробки було обрано PyCharm 2016 Community Edition. Програмний продукт є безкоштовним та має широкі можливості для розробки, рефакторінгу, налагодження та профілювання програмних засобів.

3.3.1 Огляд мови програмування PHP

PHP – скриптова мова загального призначення, інтенсивно застосовується для розробки веб-додатків В даний час підтримується переважною більшістю хостинг-провайдерів і є одним з лідерів серед мов, що застосовуються для створення динамічних веб-сайтів. Мова і його інтерпретатор розробляються групою ентузіастів в рамках проекту з відкритим кодом. Проект поширюється під власною ліцензією, несумісною з GNU GPL.

В області веб-програмування, зокрема серверної частини, PHP – один з популярних сценарних мов.

Популярність в області побудови веб-сайтів визначається наявністю великого набору вбудованих засобів для розробки веб-додатків [10]. Основні з них:

автоматичне вилучення POST і GET-параметрів, а також змінних оточення веб-сервера в зумовлені масиви;

взаємодія з великою кількістю різних систем управління базами даних (MySQL, MySQLi, SQLite, PostgreSQL, Oracle (OCI8), Oracle, Microsoft SQL Server, Sybase, ODBC, mSQL, IBM DB2 і т.п.);

автоматизована відправка HTTP-заголовків;

робота з HTTP-авторизацією;

робота з cookies і сесіями;

робота з локальними і віддаленими файлами, сокетами;

обробка файлів, що завантажуються на сервер.

Синтаксис PHP подібний синтаксису мови Сі. Деякі елементи, такі як асоціативні масиви і цикл foreach, запозичені з Perl.

Для роботи програми не потрібно описувати будь-які змінні, використовувані модулі і т. п. Будь-яка програма може починатися безпосередньо з оператора PHP.

Найпростіша програма Hello world на PHP представлена на рис. 3.1.

Рисунок 3.1 – Приклад простої программи на мові PHP

Імена змінних починаються з символу $, тип змінної оголошувати не потрібно. Імена змінних і констант чутливі до регістру символів. Імена класів, методів класів і функцій до регістру символів не чутливі. Змінні обробляються в рядках, ув'язнених в подвійні лапки, і heredoc-рядках (рядках, створених за допомогою оператора <<<). Змінні в рядках, укладених в одинарні лапки, не обробляються.

PHP розглядає перехід на новий рядок як пробіл, так само як HTML і інші мови з вільним форматом. Інструкції поділяються за допомогою крапки з комою (;), за винятком деяких випадків, після оголошення конструкції if /else і циклів.

Змінні в функцію можна передавати як за значенням, так і за посиланням (використовується знак &). PHP підтримує три типи коментарів: в стилі мови С (обмежені / \* \* /), C++ (починаються з // і йдуть до кінця рядка).

PHP є мовою програмування з динамічною типізацією, що не вимагає вказівки типу при оголошенні змінних, так само як і самого оголошення змінних. Перетворення між скалярними типами часто здійснюються неявно без додаткових зусиль (втім, PHP надає широкі можливості і для явного перетворення типів).

До скалярним типів даних відносяться:

цілий тип (integer);

дійсний тип даних (float, double);

логічний тип (boolean);

строковий тип (string);

спеціальний тип NULL.

До нескалярним типів відносяться:

«Ресурс» (resource);

масив (array);

об'єкт (object).

До псевдотіпов відносяться:

mixed будь-який тип

number число (integer або float)

callback (string або анонімна функція)

void відсутність параметрів

Діапазон цілих чисел (integer) в PHP залежить від платформи (зазвичай, це діапазон 32-бітних знакових цілих чисел, тобто, від -2 147 483 648 до 2 147 483 647). Числа можна задавати в десятковій, вісімковій та шістнадцятковій системах числення. Діапазон дійсних чисел (double) також залежить від платформи (для 32-бітної архітектури діапазон дозволяє оперувати числами від ± 1.7 × 10-308 до ± 1.7 × 10 + 308).

PHP надає розробникам логічний тип (boolean), здатний приймати тільки два значення TRUE ( «істина») і FALSE ( «брехня»). При перетворенні в логічний тип число 0, порожній рядок, нуль в рядку «0», NULL і порожній масив вважаються рівними FALSE. Всі інші символи будуть автоматично перетворюються в TRUE.

Спеціальний тип NULL призначений для змінних без певного значення. Єдиним значенням даного типу є константа NULL. Тип NULL приймають неініціалізовані змінні, змінні початкові константою NULL, а також змінні, вилучені за допомогою конструкції unset ().

Посилання на зовнішні ресурси мають тип «ресурс» (resource). Змінні даного типу, як правило, представляють собою дескриптор, що дозволяє управляти зовнішніми об'єктами, такими як файли, динамічні зображення, результуючі таблиці бази даних і т. п.

Масиви (array) підтримують числові і рядкові ключі і є гетерогенними. Масиви можуть містити значення будь-яких типів, включаючи інші масиви. Порядок елементів і їх ключів зберігається. Не зовсім коректно називати php-масиви масивами, насправді це, швидше за все, упорядкований хеш. Можливо несподіване поведінку при використанні циклу for з лічильником замість foreach. Так, наприклад, при сортуванні масиву з чисельними індексами функціями зі стандартної бібліотеки, упорядковано і ключі теж [19].

3.3.2 Огляд мови програмування С#

C# – об'єктно-орієнтована мова програмування. Розроблено в 1998-2001 роках групою інженерів під керівництвом Андерса Хейлсберг в компанії Microsoft як мова розробки додатків для платформи Microsoft .NET Framework і згодом був стандартизований як ECMA-334 і ISO / IEC 23270.

C# відноситься до сім'ї мов з C-подібним синтаксисом, з них його синтаксис найбільш близький до C ++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі в форматі XML.

Перейнявши багато від своїх попередників – мов C++, Pascal, Модула, Smalltalk і, особливо, Java – С#, спираючись на практику їх використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, C# на відміну від C++ не підтримує множинне успадкування класів (між тим допускається множинне спадкування інтерфейсів).

C# розроблявся як мова програмування прикладного рівня для CLR і, як такий, залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#, яка відображає BCL. Присутність або відсутність тих чи інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльований в відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід очікувати і в подальшому (проте, ця закономірність була порушена з виходом C # 3.0, що представляє собою розширення мови, що не спираються на розширення платформи .NET). CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування. Наприклад, Прибирання сміття не реалізована в самому C #, а проводиться CLR для програм, написаних на C# точно так же, як це робиться для програм на VB.NET, J# і ін. Перевагою подібного способу виконання програм є повна незалежність байт-коду від операційної системи і устаткування, що дозволяє виконувати C #-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина.

На рисунку 3.2 представлений код класичної програми «Hello world» на C # для консольного застосування.

Рисунок 3.2 – Приклад простої программи на мові C#

3.3.3 Огляд мови програмування Java

Java – об'єктно-орієнтована мова програмування, розроблений компанією Sun Microsystems (в подальшому придбаної компанією Oracle). Програми Java зазвичай транслюються в спеціальний байт-код, тому вони можуть працювати на будь-який віртуальної Java-машині незалежно від комп'ютерної архітектури. Дата офіційного випуску – 23 травня 1995 року.

Програми на Java транслюються в байт-код, що виконується віртуальною машиною Java (JVM) – програмою, обробній байтовий код і передавальної інструкції обладнанню як інтерпретатор.

Перевагою подібного способу виконання програм є повна незалежність байт-коду від операційної системи і устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина. Іншою важливою особливістю технології Java є гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання [21].

Часто до недоліків концепції віртуальної машини відносять зниження продуктивності. Ряд удосконалень кілька збільшив швидкість виконання програм на Java:

застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми (JIT-технологія) з можливістю збереження версій класу в машинному коді;

широке використання переносних орієнтованого коду (native-код) в стандартних бібліотеках;

апаратні засоби, що забезпечують прискорену обробку байт-коду (наприклад, технологія Jazelle, підтримувана деякими процесорами фірми ARM).

Lля семи різних завдань час виконання на Java становить в середньому в півтора-два рази більше, ніж для C / C ++, в деяких випадках Java швидше, а в окремих випадках в 7 разів повільніше. З іншого боку, для більшості з них споживання пам'яті Java-машиною було в 10-30 разів більше, ніж програмою на C / C ++. Також примітно дослідження, проведене компанією Google, згідно з яким відзначається істотно нижча продуктивність і більше споживання пам'яті в тестових прикладах на Java в порівнянні з аналогічними програмами на C ++.

Ідеї, закладені в концепцію і різні реалізації середовища віртуальної машини Java, надихнули безліч ентузіастів на розширення переліку мов, які могли б бути використані для створення програм, що виконуються на віртуальній машині . Ці ідеї знайшли також вираз в специфікації загальномовна інфраструктури CLI, закладеної в основу платформи .NET компанією Microsoft.

Основні можливості Java:

автоматичне керування пам'яттю;

розширені можливості обробки виняткових ситуацій;

багатий набір засобів фільтрації вводу-виводу;

набір стандартних колекцій: масив, список, стек і т. п .;

наявність простих засобів створення мережевих додатків (у тому числі з використанням протоколу RMI);

наявність класів, що дозволяють виконувати HTTP-запити і обробляти відповіді;

вбудовані в мову засоби створення багатопоточних додатків, які потім були перенести на багато мов (наприклад, python);

уніфікований доступ до баз даних;

даних на основі Java Data Objects (англ.) і Java Persistence API;

підтримка узагальнень (починаючи з версії 1.5);

підтримка лямбда, замикань, вбудовані можливості функціонального програмування (з 1.8).

3.3.4 Огляд мови програмування Scala

Scala – мультипарадигмальності мову програмування, спроектований коротким і тіпобезопасним для простого і швидкого створення компонентного програмного забезпечення, що поєднує можливості функціонального та об'єктно-орієнтованого програмування.

Перші версії мови створені в 2003 році колективом лабораторії методів програмування Федеральної політехнічної школи Лозанни під керівництвом Мартіна Одерської, мова реалізований для платформ Java і .Net. На думку Джеймса Стречі (англ. James Strachan), творця мови програмування Groovy, Scala може стати наступником мови Java.

У Scala використовується чиста об'єктно-орієнтована модель, схожа на застосовувану в Smalltalk: кожне значення – це об'єкт, і кожна операція – це відправка повідомлення. Наприклад, додавання x + y інтерпретується як x. + (Y), тобто як виклик методу + з аргументом y і x в якості об'єкта-приймача. Розглянемо ще один приклад: 1 + 2. Цей вислів інтерпретується як (1). + (2). Дужки навколо чисел обов'язкові, тому що лексичний аналізатор Scala розбиває вираз на лексеми за принципом найдовшого можливого зіставлення. Таким чином, вираз 1. + (2) розіб'ється на лексеми 1., + І 2, тому що лексема 1. довше лексеми 1 і перший аргумент складання буде інтерпретований, як тип Double замість Int.

Кожна функція в мові Scala – це значення. Мова надає легкий синтаксис для визначення анонімних і каррінгових функцій. Кожна конструкція повертає значення. Зіставлення зі зразком природно розширюється до обробки XML c допомогою регулярних виразів. Кожна компонентна система з потужними конструкціями абстракції і композиції стикається з проблемою, коли справа доходить до інтеграції підсистем, розроблених різними командами в різний час. Проблема полягає в тому, що інтерфейс компонентів, розроблених тією чи іншою групою, часто не підходить клієнтам, які мають намір використовувати цей компонент.

Scala представляє нову концепцію вирішення проблеми зовнішньої розширюваності – види (views). Вони дозволяють розширювати клас новими членами і trait-ами. Види в Scala переводять в об'єктно-орієнтоване уявлення використовувані в Haskell класи типів. На відміну від класів типів, область видимості видів можна контролювати, причому в різних частинах програми можуть співіснувати паралельні види [22].

Ключові аспекти мови:

Scala-програми багато в чому схожі на Java-програми, і можуть вільно взаємодіяти з Java-кодом;

Scala включає одноманітну об'єктну модель – в тому сенсі, що будь-яке значення є об'єктом, а будь-яка операція – викликом методу;

Scala – це також функціональна мова в тому сенсі, що функції – це повноправні значення;

у Scala включені потужні і одноманітні концепції абстракцій як для типів, так і для значень;

вона містить гнучкі симетричні конструкції домішок (mixin) для композиції класів і trait-ів;

вона дозволяє виробляти декомпозицію об'єктів шляхом порівняння зі зразком;

зразки і вирази були узагальнені для підтримки природної обробки XML-документів;

в цілому, ці конструкції дозволяють легко висловлювати самостійні компоненти, які використовують бібліотеки Scala, не користуючись спеціальними мовними конструкціями;

Scala допускає зовнішні розширення компонентів з використанням видів (views);

наявність шаблонів (generics) і шаблонів вищих порядків (generics of a higher kind);

є підтримка структурних та екзистенційних типів.

3.3.4 Огляд мови програмування Python

Python – високорівнева мова програмування загального призначення, орієнтований на підвищення продуктивності розробника і читання коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує кілька парадигм програмування, в тому числі структурний, об'єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване. Основні архітектурні риси – динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточних обчислень і зручні високорівневі структури даних. Код в Python організовується у функції та класи, які можуть об'єднуватися в модулі (вони в свою чергу можуть бути об'єднані в пакети).

Еталонної реалізацією Python є інтерпретатор CPython, що підтримує більшість активно використовуваних платформ. Він поширюється під вільною ліцензією Python Software Foundation License, що дозволяє використовувати його без обмежень у будь-яких додатках, включаючи пропрієтарні. Є реалізації інтерпретаторів для JVM (з можливістю компіляції), MSIL (з можливістю компіляції), LLVM та інших. Проект PyPy пропонує реалізацію Python на самому Python, що зменшує витрати на зміни мови та постановку експериментів над новими можливостями.

Python – активно розвивається мова програмування, нові версії (з додаванням / зміною мовних властивостей) виходять приблизно раз в два з половиною роки. Внаслідок цього і деяких інших причин на Python відсутні стандарт ANSI, ISO або інші офіційні стандарти, їх роль виконує Cpython.

Мова володіє чітким і послідовним синтаксисом, продуманої модульність і масштабованість, завдяки чому вихідний код написаних на Python програм легко читаємо. При передачі аргументів у функції Python використовує виклик по соіспользованію.

Набір операторів досить традиційний:

умовний оператор if (якщо). Альтернативний блок після else (інакше). Якщо умов і альтернатив кілька, можна використовувати elif (скор. Від else if);

оператори циклу while (поки) і for (для). Всередині циклу можливе застосування break і continue для переривання циклу і переходу відразу до наступної ітерації відповідно;

оператор визначення класу class;

оператор визначення функції, методу чи генератора def. Всередині можливе застосування return (повернення) для повернення з функції або методу, а в разі генератора – yield (давати);

оператор обробки винятків try-except-else або try-finally (починаючи з версії 2.5, можна використовувати finally, except і else в одному блоці);

оператор pass нічого не робить. Використовується для порожніх блоків коду.

Однією з цікавих синтаксичних особливостей мови є виділення блоків коду за допомогою відступів (прогалин чи табуляцій), тому в Python відсутні операторні дужки begin / end, як у мові Паскаль, або фігурні дужки, як в Сі. Такий «трюк» дозволяє скоротити кількість рядків і символів в програмі і привчає до «хорошого» стилю програмування. З іншого боку, поведінка і навіть коректність програми може залежати від початкових прогалин в тексті. Деяким така поведінка може здатися неінтуітівнимі і незручним.

Вираз є повноправним оператором в Python. Склад, синтаксис, асоціативність і пріоритет операцій досить звичні для мов програмування і покликані мінімізувати вживання дужок.

2.4 Об’єктно-орієнтоване проектування

Для розробки програмного продукту був обраний метод об'єктно-орієнтовного проектування [14].

Підставами до вибору даної парадигми є:

1. швидкість модифікації та розширення програмного коду;
2. наявність якісного середовища проектування;
3. широкі можливості з повторного використання коду;
4. зручність при розробці архітектури системи;
5. наявність загальноприйнятих стратегій щодо вирішення типових проблем.

Для створення об’єктно-орієнтованої моделі системи використовуються діаграми класів UML [15].

Під час об’єктно-орієнтованого проектування для вирішення деяких типових проблем були використані шаблони проектування.

Для роботи з даними використовується технологія ORM (object-relational mapping – відображення даних на реальні об’єкти) Entity Framework.

Entity Framework представляє собою об’єктно-орієнтовану технологію на базі .NET Framework для роботи з даними. Якщо традиційні засоби ADO.NET дозволяють створювати підключення, команди та інші об’єкти для взаємодії з базами даних, то Entity Framework представляє собою більш високий рівень абстракції, котрий дозволяє абстрагуватися від самої бази даних і працювати незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, ключами, то на концептуальному рівні, котрий пропонує Entity Framework, ми працюємо з об’єктами.

З виходом версії 4.0 в 2010 році Entity Framework став рекомендованною технологією для доступу до даних, були введені можливості взаємодії з базою даних – підходи Database First, Model First, Code First. В даній розробці використовується підхід Code First.

Центральною концепцією Entity Framework є поняття сутності. Сутність представляє набір даних асоційованих з конкретним об’єктом. Тому дана технологія передбачає роботу не з таблицями, а з об’єктами. Для вибірки з бази даних використовуються запроси LINQ.

Таким чином через класи, об’явлені в програмі можна взаємодіяти з таблицями бази даних.

Для розробки була обрана трирівнева архітектура, яка ділить проект на три рівні:

1. Presentation layer (рівень представлення даних) – рівень з яким безпосередньо взаємодіє користувач. Включає компоненти користувацького інтерфейсу, механізм прийому даних від користувача;
2. Business layer (рівень бізнес логіки) – містить набір компонентів, які відповідають за обробку отриманих від рівня представлення даних, реалізує всю логіку програми, взаємодіє з рівнем доступу до даних і передає йому результат обробки;
3. Data access layer (рівень доступу до даних) – зберігає моделі, які описують використовувані сутності, також тут містяться специфічні класи для роботи з різними технологіями доступу до даних, наприклад клас контексту Entity Framework. Тут також зберігаються репозиторії, через які рівень бізнес логіки взаємодіє з базою даних.

Для роботи з даними був обраний шаблон «Репозиторій» – який дозволяє абстрагуватися від конкретних підключень до даних, з якими працює програма і є проміжною ланкою між класами, які безпосередньо взаємодіють з даними і рештою програми. Таким чином «Репозиторій» додає гнучкості програмі при роботі з різними типами підключень. Через класи репозиторіїв рівень бізнес логіки взаємодіє з базою даних.

Для того щоб спростити роботу з великою кількістю репозиторіїв був обраний шаблон «Одиниця роботи». Даний шаблон дає доступ до репозиторіїв через окремі методи і визначає загальний контекст для всіх репозиторіїв.

Крім того на рівні представлення даних використовується шаблон MVP – який є похідним від MVC. Відмінність заключається в тому що, представлення не зв’язано з модел’ю. Складається з: Model – Presenter – View. Презентер відповідає за переміщення даних, а також за оновлення представлень. «Спілкування» моделей з представленнями відбувається через презентер.

Діаграми UML проекту представлені на рис. 2.62 – 2.76.

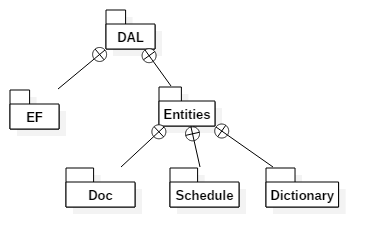


Рисунок 2.62 – Діаграма пакетів рівня доступу до даних

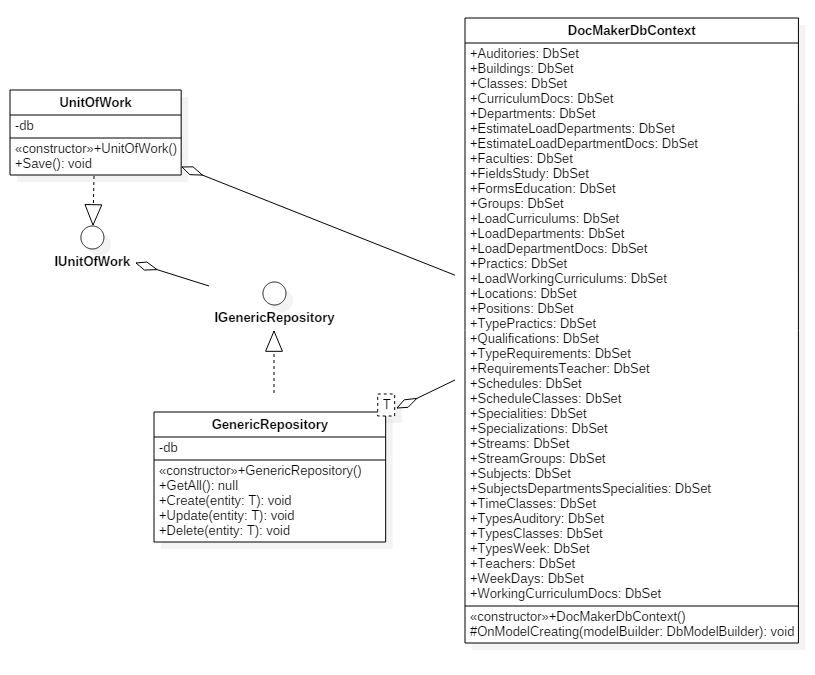


Рисунок 2.63 – Діаграма класів пакету EF

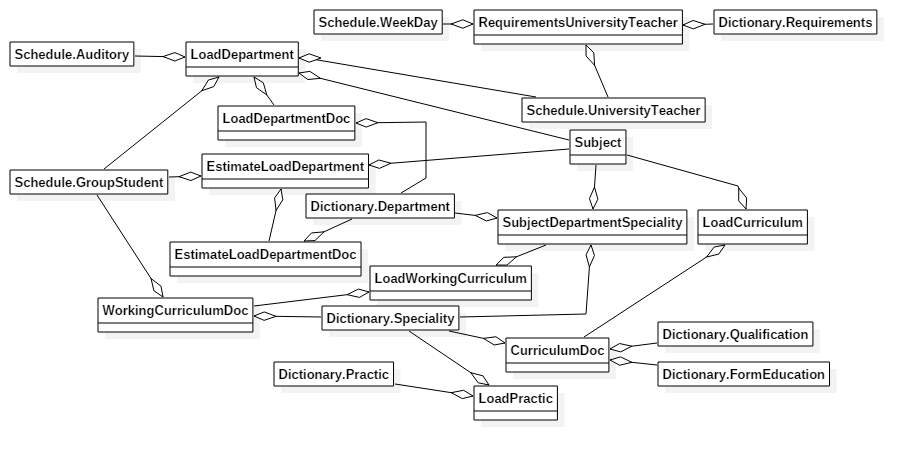


Рисунок 2.64 – Діаграма класів пакету Doc

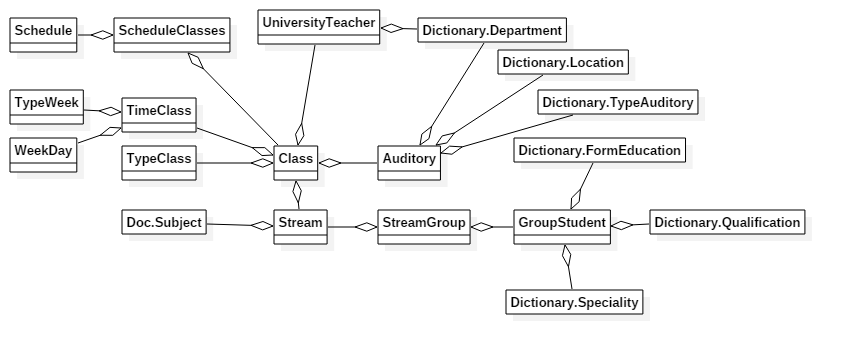


Рисунок 2.65 – Діаграма класів пакету Schedule

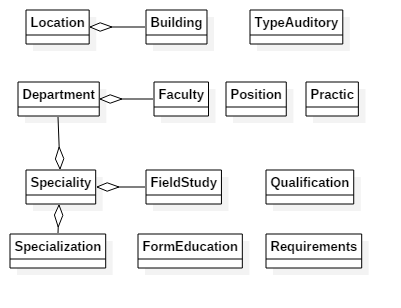


Рисунок 2.66 – Діаграма класів пакету Dictionary

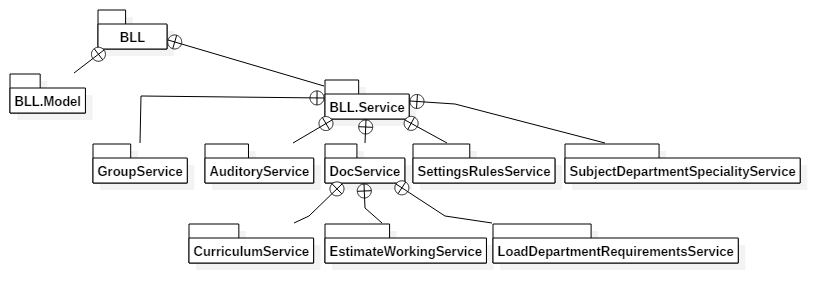


Рисунок 2.67 – Діаграма пакетів рівня бізнес логіки

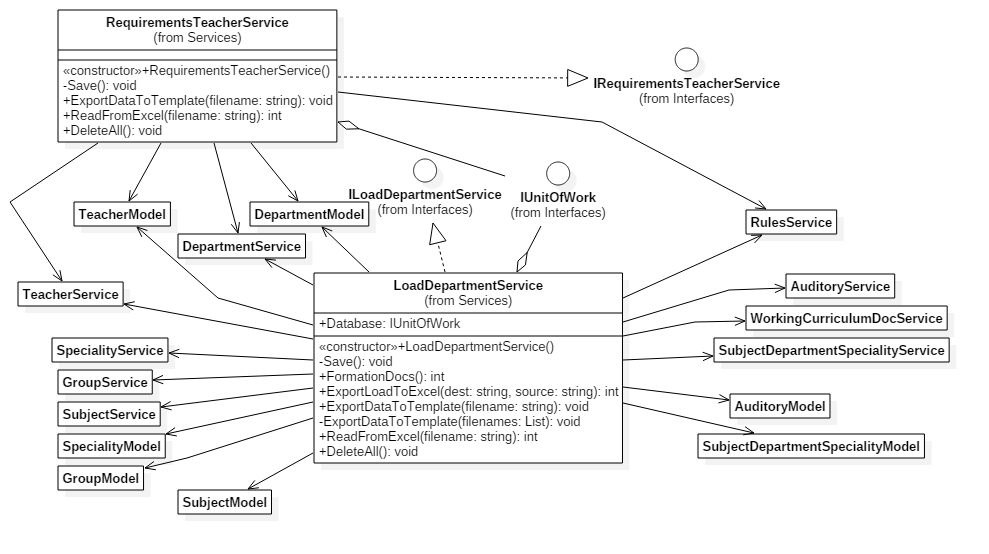


Рисунок 2.68 – Діаграма класів пакету LoadDepartmentRequirementsService

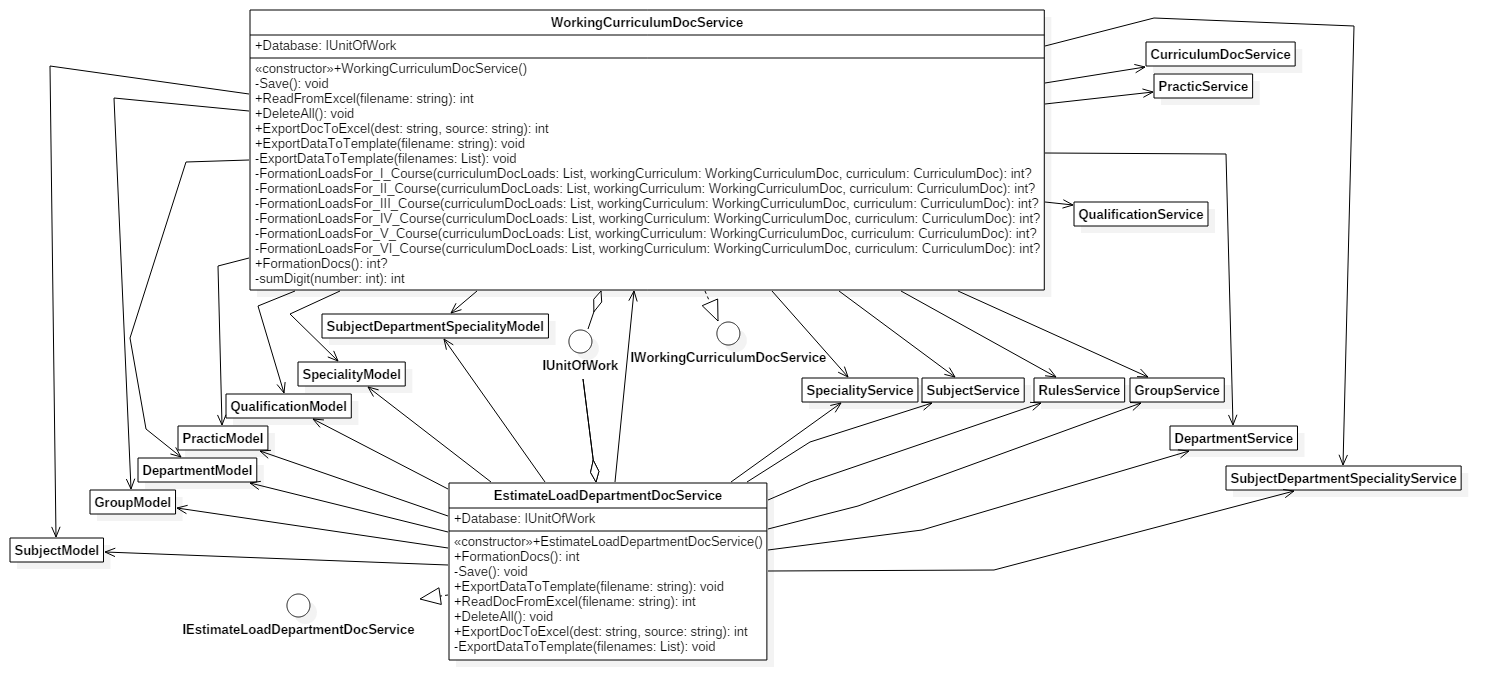


Рисунок 2.69 – Діаграма класів пакету EstimateWorkingService

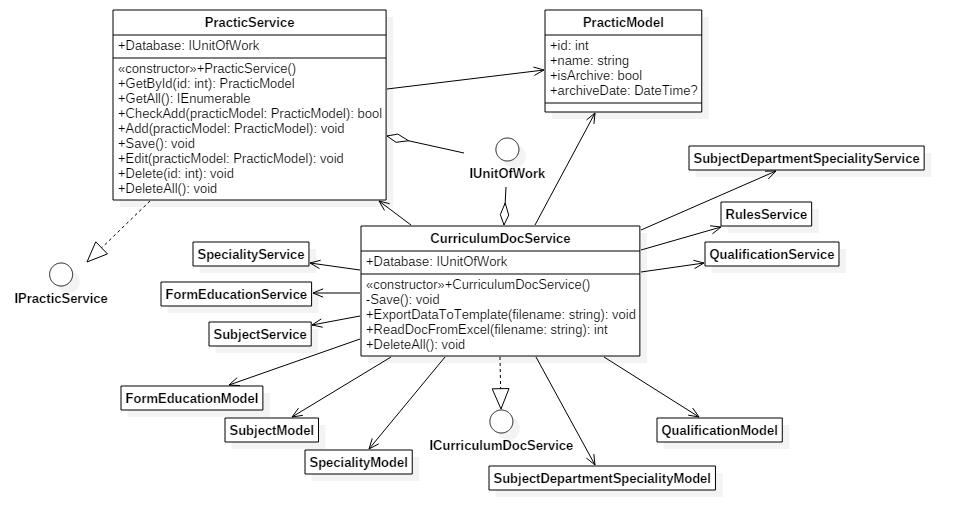


Рисунок 2.70 – Діаграма класів пакету CurriculumService

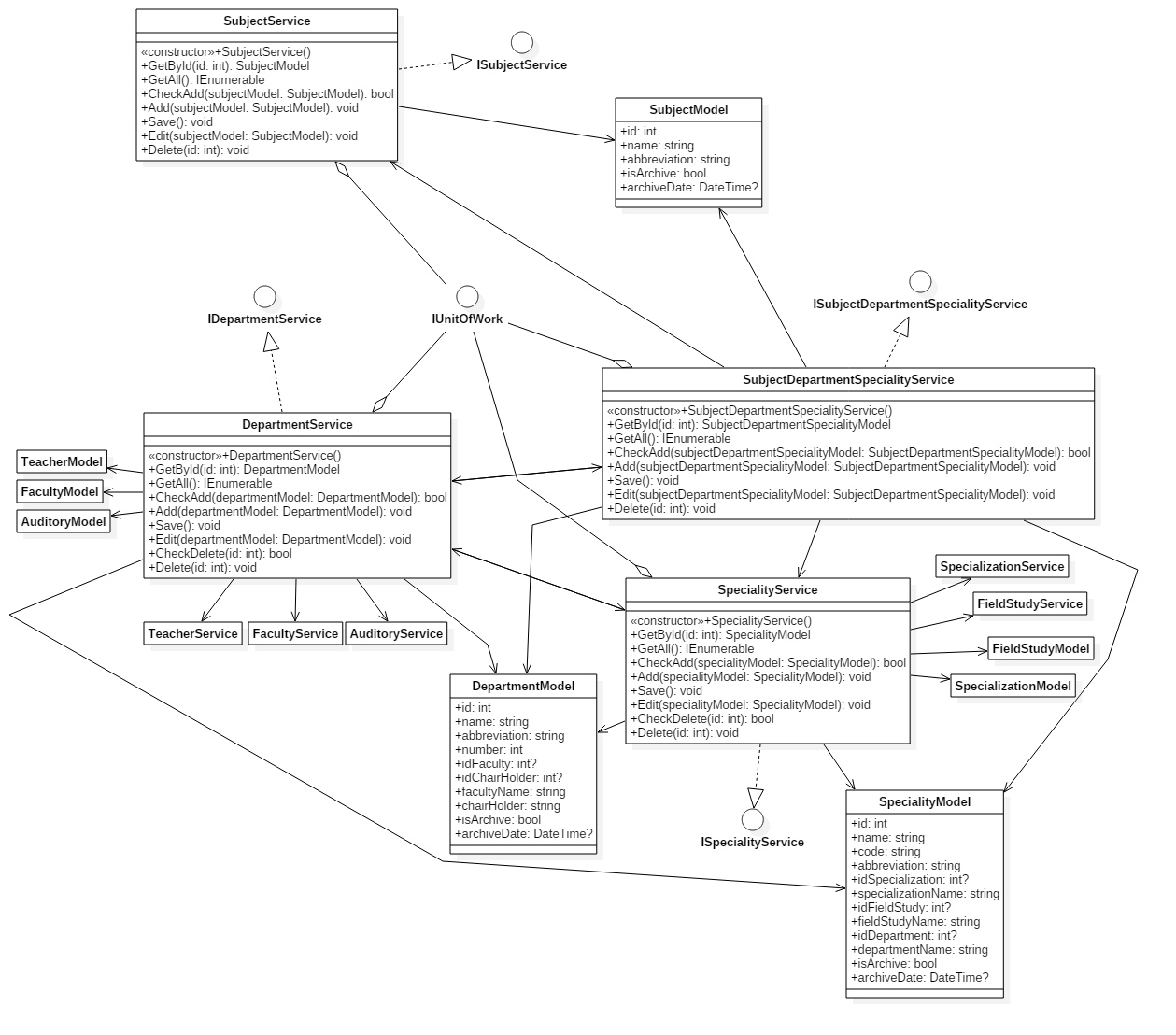


Рисунок 2.71 – Діаграма класів пакету SubjectDepartmentSpecialityService

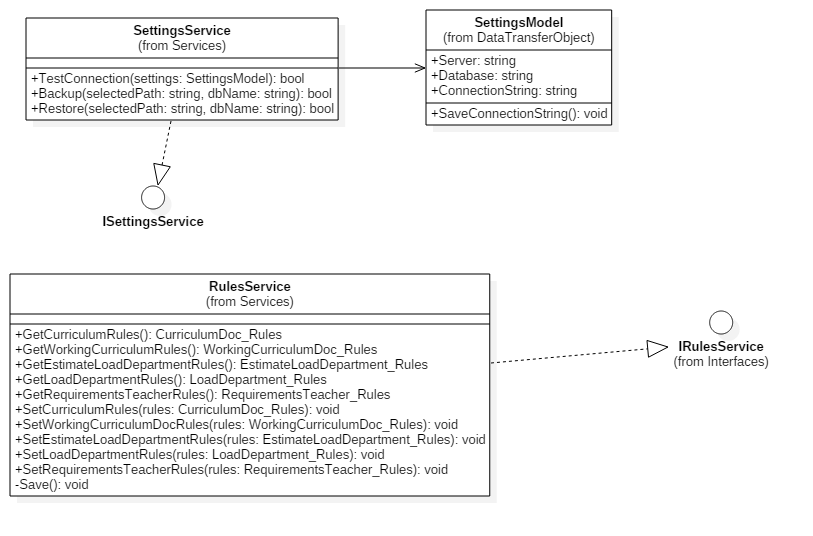


Рисунок 2.72 – Діаграма класів пакету SettingsRulesService

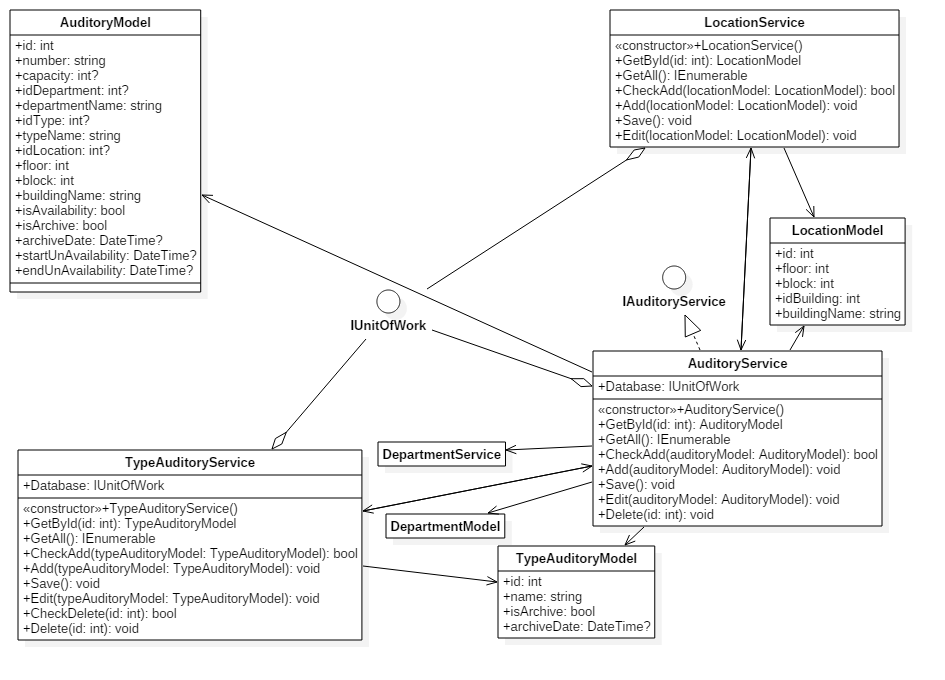


Рисунок 2.73 – Діаграма класів пакету AuditoryService

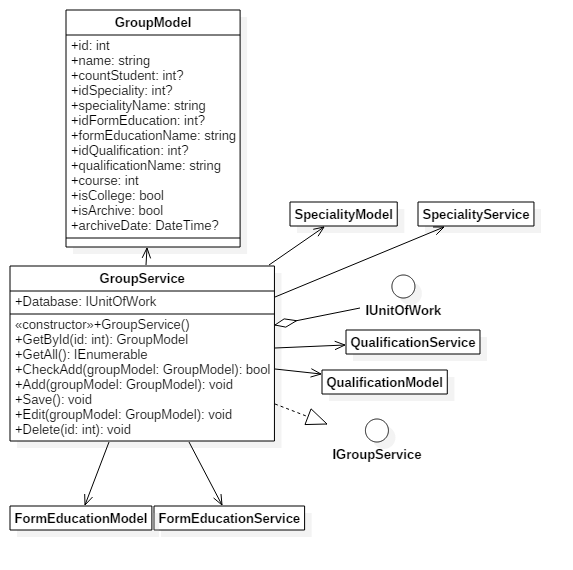


Рисунок 2.74 – Діаграма класів пакету GroupService



Рисунок 2.75 – Діаграма класів пакету BLL.Model

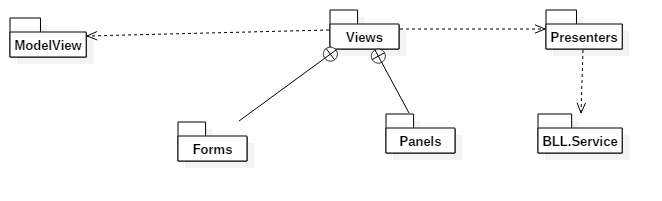


Рисунок 2.76 – Діаграма пакетів рівня представлення даних

2.5 Розробка інтерфейсу користувача

Етап розробки інтерфейсу користувача є дуже важливим етапом в створені даної системи. Це зумовлено тим, що працівникам учбового відділу необхідно оперувати великою кількістю інформації (велика кількість аудиторій, занять, тощо). Тому інтерфейс системи має забезпечувати швидкий пошук інформації, її зручне подання та максимальну кількість інформації, що одночасно відображається користувачу.

Окрім цього інтерфейс програми має бути зручним та інтуїтивно зрозумілим. Ніякі дії користувача не повинні призвести до помилок в системі. У разі виникнення непередбачуваних станів, користувач повинен отримати вичерпне повідомлення про помилку, можливі причині її виникнення та рекомендовані подальші дії.

Колір є одним з важливих атрибутів інтерфейсу і досить суб’єктивним, тому користувач має змогу самостійно його обрати.

Так як методисту доведеться оперувати досить великою кількістю даних, були додані можливості фільтру інформації за допомогою пошуку по будь-якому з полів, що дозволить залишати лише необхідні на даний момент дані.

Всі елементи інтерфейсу для зручності навігації були згруповані в меню з плиток, які відрізняються не лише заголовками, а й зображеннями (рис. 2.77).



Рисунок 2.77 – Головна форма з меню

3 ТЕСтування та налагодження програми

3.1 Вибір стратегії тестування

Для досягнення максимальної якості тестування під кожен метод або модуль програми, що тестується необхідно обрати найбільш вдалий метод або набір методів, що забезпечать необхідний результат. При цьому необхідно підібрати найкраще співвідношення між часом, що буде витрачено на тестування, та якістю тестування. Набір тестів повинен мати мінімальну збитковість, але при цьому максимально охоплювати функціональність системи [16].

Під час тестування програми буде використано декілька різних методів тестування. Умовно ці методи можна поділити на 2 типи: методи білого ящика та методи чорного ящика. Різниця між цими групами методів полягає в тому, що при використанні методів чорного ящика, тестування відбувається без доступу до коду програми, в наявності тестувальника є тільки ті можливості, що матиме користувач програми. Під час використання методів білого ящика, тестувальник використовує код програми для досягнення необхідного результату.

3.2 Тестування функціональності

3.2.1 Тестування методу ReadDocFromExcel класу CurriculumDocService

3.2.1.1 Описання методу

Метод зчитує дані з .xls файлу та завантажує їх в базу даних.

Вхідні дані: файл навчального плану в форматі .xls; правила читання файлів навчальних планів; база даних, а саме таблиці:

* CurriculmDoc;
* LoadCurriculumDoc;
* Qualification;
* FormEducation;
* Speciality;
* Subject;
* SubjectDepartmentSpeciality;
* Practic;
* LoadPractic.

Вихідні дані: результат зчитування та завантаження (-1 – не успішно; 0 – успішно; 1 – такі дані вже є), база даних.

3.2.1.2 Текст методу

public int ReadDocFromExcel(string filename)

{

int result = 0, index = 0;

string codeSpeciality = String.Empty, nameSpeciality = String.Empty;

string nameSubject = String.Empty;

bool isHalfSemester = false;

CurriculumDoc newCurriculumDoc = new CurriculumDoc();

LoadCurriculum curriculumLoad = new LoadCurriculum();

FileInfo fi = new FileInfo(filename);

using (ExcelPackage excel = new ExcelPackage(fi))

{

try

{

RulesService service = new RulesService();

CurriculumDoc\_Rules rules = service.GetCurriculumRules();

ExcelWorksheet sheet = excel.Workbook.Worksheets[rules.titleMainList];

var yearReceptionExcel = sheet.Cells[rules.yearReception\_Cell].Value;

var qualificationNameExcel = sheet.Cells[rules.qualification\_Cell].Value;

var academicTermExcel = sheet.Cells[rules.termEducation\_Cell].Value;

var formEducationNameExcel = sheet.Cells[rules.formEducation\_Cell].Value;

var specialityNameExcel = sheet.Cells[rules.speciality\_Cell].Value;

if (specialityNameExcel == null || formEducationNameExcel == null || qualificationNameExcel == null || yearReceptionExcel == null || academicTermExcel == null) **//1**

{

throw new Exception();

}

while (specialityNameExcel.ToString()[index] != ' ') **//2**

{

codeSpeciality += specialityNameExcel.ToString()[index];

index++;

}

for (index = index + 1; index < specialityNameExcel.ToString().Length; index++)

{

if (Char.IsLetter(specialityNameExcel.ToString()[index]) == true || specialityNameExcel.ToString()[index] == ' ') **//3**

{

nameSpeciality += specialityNameExcel.ToString()[index];

}

}

var qualification = Database.Qualifications.GetAll().Where(\_ => \_.isArchive == false && \_.name.ToLower() == qualificationNameExcel.ToString().ToLower()).FirstOrDefault();

var formEducation = Database.FormsEducation.GetAll().Where(\_ => \_.isArchive == false && \_.name.ToLower() == formEducationNameExcel.ToString().ToLower()).FirstOrDefault();

var speciality = Database.Specialities.GetAll().Where(\_ => \_.isArchive == false && \_.name.ToLower() == nameSpeciality.ToLower()).FirstOrDefault();

if (qualification == null) **//4**

{

Qualification qualificationModel = new Qualification

{

name = qualificationNameExcel.ToString()

};

Database.Qualifications.Create(qualificationModel);

Save();

qualification = qualificationModel;

}

if (formEducation == null) **//5**

{

FormEducation formEducationModel = new FormEducation

{

name = formEducationNameExcel.ToString()

};

Database.FormsEducation.Create(formEducationModel);

Save();

formEducation = formEducationModel;

}

if (speciality == null) **//6**

{

Speciality specialityModel = new Speciality

{

code = codeSpeciality,

name = nameSpeciality

};

Database.Specialities.Create(specialityModel);

Save();

speciality = specialityModel;

}

var curriculumDoc = Database.CurriculumDocs.GetAll().Where(\_ => \_.academicTerm == Convert.ToInt32(academicTermExcel) && \_.idFormEducation == formEducation.id && \_.idQualification == qualification.id && \_.idSpeciality == speciality.id && \_.yearReception == Convert.ToInt32(yearReceptionExcel));

if (curriculumDoc.Count() < 1) **//7**

{

newCurriculumDoc.academicTerm = Convert.ToInt32(academicTermExcel);

newCurriculumDoc.idFormEducation = formEducation.id;

newCurriculumDoc.idQualification = qualification.id;

newCurriculumDoc.idSpeciality = speciality.id;

newCurriculumDoc.yearReception = Convert.ToInt32(yearReceptionExcel);

Database.CurriculumDocs.Create(newCurriculumDoc);

Save();

sheet = excel.Workbook.Worksheets[rules.titleLearningProcessList];

for (int j = rules.startReading\_Row; j < rules.endReading\_Row + 1; j++)

{

if (sheet.Cells[rules.discipline\_Column + j].Value != null && sheet.Cells[rules.individualWork\_Column + j].Value != null && (sheet.Cells[rules.exam\_Column + j].Value != null || sheet.Cells[rules.test\_Column + j].Value != null))**//8**

{

var subjectNameExcel = sheet.Cells[rules.discipline\_Column + j].Value;

var exam = sheet.Cells[rules.exam\_Column + j].Value;

var test = sheet.Cells[rules.test\_Column + j].Value;

var courseWork = sheet.Cells[rules.courseWork\_Column + j].Value;

var lecture = sheet.Cells[rules.lecture\_Column + j].Value;

var labWork = sheet.Cells[rules.labWork\_Column + j].Value;

var practicWork = sheet.Cells[rules.practicWork\_Column + j].Value;

var individualTask = sheet.Cells[rules.individualTask\_Column + j].Value;

var individualWork = sheet.Cells[rules.individualWork\_Column + j].Value;

var numberSubject = sheet.Cells[rules.number\_Column + j].Value;

var totalHour = sheet.Cells[rules.totalHour\_Column + j].Value;

var semester1 = sheet.Cells[rules.semester1\_Column + j].Value;

var semester2 = sheet.Cells[rules.semester2\_Column + j].Value;

var semester3 = sheet.Cells[rules.semester3\_Column + j].Value;

var semester4 = sheet.Cells[rules.semester4\_Column + j].Value;

var semester5 = sheet.Cells[rules.semester5\_Column + j].Value;

var semester6 = sheet.Cells[rules.semester6\_Column + j].Value;

var semester7 = sheet.Cells[rules.semester7\_Column + j].Value;

var semester8 = sheet.Cells[rules.semester8\_Column + j].Value;

var semester9 = sheet.Cells[rules.semester9\_Column + j].Value;

var semester10 = sheet.Cells[rules.semester10\_Column + j].Value;

var semester11 = sheet.Cells[rules.semester11\_Column + j].Value;

var semester12 = sheet.Cells[rules.semester12\_Column + j].Value;

if (subjectNameExcel.ToString()[subjectNameExcel.ToString().Length - 1] == ')' && subjectNameExcel.ToString()[subjectNameExcel.ToString().Length - 2] == '8') **//9**

{

isHalfSemester = true;

nameSubject = subjectNameExcel.ToString().Substring(0, (subjectNameExcel.ToString().Length - 4));

}

else if (subjectNameExcel.ToString()[subjectNameExcel.ToString().Length - 1] == specialSubject) **//10**

{

nameSubject = subjectNameExcel.ToString().Remove(subjectNameExcel.ToString().Length - 1);

if (lecture != null && lecture.ToString()[lecture.ToString().Length - 1] == specialSubject) **//11**

{

lecture = lecture.ToString().Remove(lecture.ToString().Length - 1);

}

if (labWork != null && labWork.ToString()[labWork.ToString().Length - 1] == specialSubject) **//12**

{

labWork = labWork.ToString().Remove(labWork.ToString().Length - 1);

}

if (practicWork != null && practicWork.ToString()[practicWork.ToString().Length - 1] == specialSubject) **//13**

{

practicWork = practicWork.ToString().Remove(practicWork.ToString().Length - 1);

}

if (individualWork != null && individualWork.ToString()[individualWork.ToString().Length - 1] == specialSubject) **//14**

{

individualWork = individualWork.ToString().Remove(individualWork.ToString().Length - 1);

}

if (totalHour != null && totalHour.ToString()[totalHour.ToString().Length - 1] == specialSubject) **//15**

{

totalHour = totalHour.ToString().Remove(totalHour.ToString().Length - 1);

}

if (semester1 != null && semester1.ToString()[semester1.ToString().Length - 1] == specialSubject) **//16**

{

semester1 = semester1.ToString().Remove(semester1.ToString().Length - 1);

}

if (semester2 != null && semester2.ToString()[semester2.ToString().Length - 1] == specialSubject) **//17**

{

semester2 = semester2.ToString().Remove(semester2.ToString().Length - 1);

}

if (semester3 != null && semester3.ToString()[semester3.ToString().Length - 1] == specialSubject) **//18**

{

semester3 = semester3.ToString().Remove(semester3.ToString().Length - 1);

}

if (semester4 != null && semester4.ToString()[semester4.ToString().Length - 1] == specialSubject) **//19**

{

semester4 = semester4.ToString().Remove(semester4.ToString().Length - 1);

}

if (semester5 != null && semester5.ToString()[semester5.ToString().Length - 1] == specialSubject) **//20**

{

semester5 = semester5.ToString().Remove(semester5.ToString().Length - 1);

}

if (semester6 != null && semester6.ToString()[semester6.ToString().Length - 1] == specialSubject) **//21**

{

semester6 = semester6.ToString().Remove(semester6.ToString().Length - 1);

}

if (semester7 != null && semester7.ToString()[semester7.ToString().Length - 1] == specialSubject) **//22**

{

semester7 = semester7.ToString().Remove(semester7.ToString().Length - 1);

}

if (semester8 != null && semester8.ToString()[semester8.ToString().Length - 1] == specialSubject) **//23**

{

semester8 = semester8.ToString().Remove(semester8.ToString().Length - 1);

}

if (semester9 != null && semester9.ToString()[semester9.ToString().Length - 1] == specialSubject) **//24**

{

semester9 = semester9.ToString().Remove(semester9.ToString().Length - 1);

}

if (semester10 != null && semester10.ToString()[semester10.ToString().Length - 1] == specialSubject) **//25**

{

semester10 = semester10.ToString().Remove(semester10.ToString().Length - 1);

}

if (semester11 != null && semester11.ToString()[semester11.ToString().Length - 1] == specialSubject) **//26**

{

semester11 = semester11.ToString().Remove(semester11.ToString().Length - 1);

}

if (semester12 != null && semester12.ToString()[semester12.ToString().Length - 1] == specialSubject) **//27**

{

semester12 = semester12.ToString().Remove(semester12.ToString().Length - 1);

}

}

else**//28**

{

nameSubject = subjectNameExcel.ToString();

}

var subject = Database.Subjects.GetAll().Where(\_ => \_.isArchive == false && \_.name.ToLower() == nameSubject.ToLower()).FirstOrDefault();

if (subject == null && nameSubject.ToLower() != "військова підготовка".ToLower() && nameSubject.ToLower() != "вибіркова дисципліна".ToLower())**//29**

{

if (lecture == null && labWork == null && practicWork == null) **//30**

{

var typePractics = Database.Practics.GetAll().Where(\_ => \_.isArchive == false && \_.name.ToLower() == nameSubject.ToLower()).FirstOrDefault();

if (typePractics == null) **//31**

{

Practic practicModel = new Practic();

practicModel.name = nameSubject;

Database.Practics.Create(practicModel);

Save();

typePractics = practicModel;

}

LoadPractic loadPractic = new LoadPractic

{

countHour = Convert.ToInt32(individualWork),

idPractic = typePractics.id,

idSpeciality = speciality.id,

semester = Convert.ToInt32(test),

yearReception = Convert.ToInt32(yearReceptionExcel)

};

Database.PracticLoads.Create(loadPractic);

Save();

continue;

}

else**//32**

{

Subject subjectModel = new Subject

{

name = nameSubject

};

Database.Subjects.Create(subjectModel);

Save();

subject = subjectModel;

}

}

if (subject != null) **//33**

{

var subjectsDepartmentSpeciality = Database.DepartmentSpecialitySubjects.GetAll().Where(\_ => \_.isArchive == false && \_.idSubject == subject.id && \_.idSpeciality == speciality.id);

if (subjectsDepartmentSpeciality.Count() < 1) **//34**

{

SubjectDepartmentSpeciality subjectDepartmentSpeciality = new SubjectDepartmentSpeciality

{

idSpeciality = speciality.id,

idSubject = subject.id

};

Database.DepartmentSpecialitySubjects.Create(subjectDepartmentSpeciality);

Save();

}

curriculumLoad.countHourIndividualWork = Convert.ToInt32(individualWork);

curriculumLoad.countHourLabWork = Convert.ToInt32(labWork);

curriculumLoad.countHourLecture = Convert.ToInt32(lecture);

curriculumLoad.countHourPracticWork = Convert.ToInt32(practicWork);

curriculumLoad.idCurriculumDoc = curriculumDoc.First().id;

curriculumLoad.idSubject = subject.id;

curriculumLoad.individualTask = Convert.ToString(individualTask);

curriculumLoad.totalHour = Convert.ToInt32(totalHour);

if (exam != null) **//35**

{

curriculumLoad.isExam = true;

}

else**//36**

{

curriculumLoad.isExam = false;

}

if (test != null) **//37**

{

curriculumLoad.isTest = true;

}

else**//38**

{

curriculumLoad.isTest = false;

}

curriculumLoad.isHalfSemester = isHalfSemester;

curriculumLoad.numberSubject = Convert.ToInt32(numberSubject);

curriculumLoad.semesterI = Convert.ToInt32(semester1);

curriculumLoad.semesterII = Convert.ToInt32(semester2);

curriculumLoad.semesterIII = Convert.ToInt32(semester3);

curriculumLoad.semesterIV = Convert.ToInt32(semester4);

curriculumLoad.semesterV = Convert.ToInt32(semester5);

curriculumLoad.semesterVI = Convert.ToInt32(semester6);

curriculumLoad.semesterVII = Convert.ToInt32(semester7);

curriculumLoad.semesterVIII = Convert.ToInt32(semester8);

curriculumLoad.semesterIX = Convert.ToInt32(semester9);

curriculumLoad.semesterX = Convert.ToInt32(semester10);

curriculumLoad.semesterXI = Convert.ToInt32(semester11);

curriculumLoad.semesterXII = Convert.ToInt32(semester12);

Database.CurriculumDocLoads.Create(curriculumLoad);

Save();

}

}

}

}

else**//39**

{

result = 1;

}

}

catch**//40**

{

result = -1;

DeleteAll();

Database.PracticLoads.DeleteAll();

Save();

}

return result;

}

}

3.2.1.3 Вибір методу тестування

Тестування даного методу було вирішено проводити одним із методів білого ящика, оскільки дані методи дозволять досягти кращої якості тестування, ніж методи чорного ящика. Серед методів білого ящика було обрано метод покриття операторів. Його використання забезпечить найкраще відношення якості тестування до витраченого часу. Метод дозволить перевірити досяжність та правильність роботи кожного оператора. Для того, щоб кожен оператор був виконаний, набір тестів має пройти всі гілки коду. Вони помічені коментарями з числами у тексті методу.

3.2.1.4 Формування набору тестів

Вхідні дані тесту 1: файл навчального плану (табл. 3.1) та рис. 3.1, правила читання (табл. 3.2), таблиці бази даних порожні.

Таблиця 3.1 – Склад файлу навчального плану, титульний лист

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рік прийому | Кваліфікаційний рівень | Термін навчання | Форма навчання | Спеціальність |
| 2015 | бакалавр | 4 | денна |  |

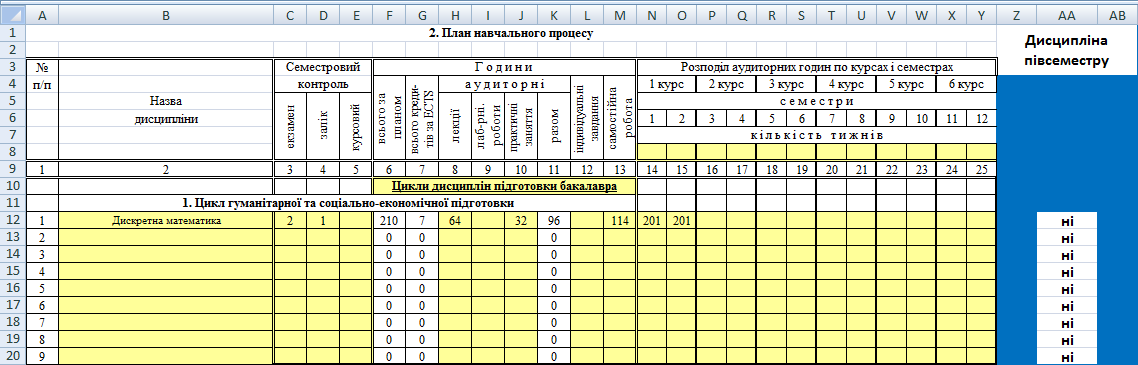


Рисунок 3.1 – Вхідні дані для тесту 1, лист навчальний процес файлу навчальний план

Таблиця 3.2 – Правила читання файлів навчальних планів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рік прийому | Кваліфікаційний рівень | Термін навчання | Форма навчання | Спеціальність |
| AS1 | AT3 | AP4 | AB11 | AB17 |

Вихідні дані тесту 1: результат зчитування – -1; таблиці бази даних, які використовуються залишилися порожніми.

Коментар: файл, що передається не містить спеціальності, а отже він не відповідає шаблону і результат виконання -1, а таблиці не змінилися. Тест проходить правильно.

Вхідні дані тесту 2: файл навчального плану (табл. 3.3) та рис. 3.2, правила читання (табл. 3.4), таблиці бази даних порожні.

Таблиця 3.3 – Склад файлу навчального плану, титульний лист

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рік прийому | Кваліфікаційний рівень | Термін навчання | Форма навчання | Спеціальність |
| 2015 | бакалавр | 4 | денна | 7.05010301 програмне забезпечення систем |

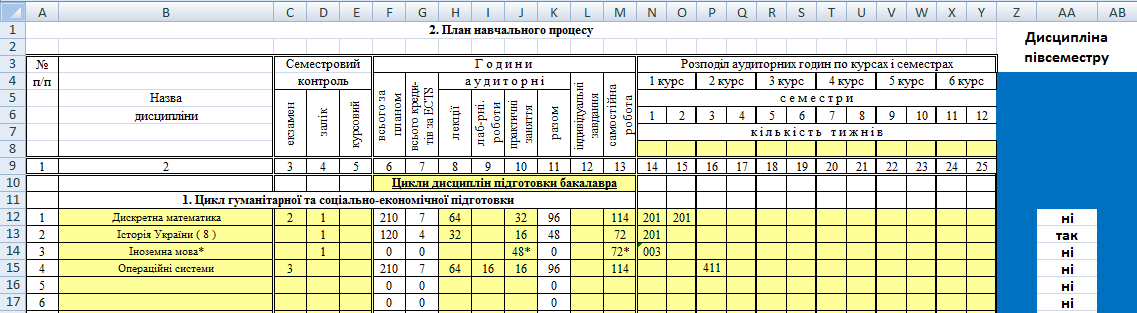


Рисунок 3.2 – Вхідні дані для тесту 2, лист навчальний процес файлу навчальний план

Таблиця 3.4 – Правила читання файлів навчальних планів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рік прийому | Кваліфікаційний рівень | Термін навчання | Форма навчання | Спеціальність |
| AS1 | AT3 | AP4 | AB11 | AB17 |

Вихідні дані тесту 2: результат зчитування – 0; таблиці бази даних   
(табл. 3.5 – 3.10) інші залишилися порожніми.

Коментар: база даних порожня, файл, що передається відповідає шаблону. Містить: дисципліни на півсеместру; спеціальні дисципліни, які помічаються \*; дисципліни тільки з екзаменом; дисципліни тільки з заліком; дисципліни з заліком і екзаменом. Тест проходить правильно.

Таблиця 3.5 – Склад таблиці «Документ навчального плану»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рік прийому | Кваліфікаційний рівень | Термін навчання | Форма навчання | Спеціальність |
| 2015 | бакалавр | 4 | денна | програмне забезпечення систем |

Таблиця 3.6 – Склад таблиці «Кваліфікаційний рівень»

|  |  |  |
| --- | --- | --- |
| Назва | Архів | Дата архіву |
| бакалавр | ні | - |

Таблиця 3.7 – Склад таблиці «Форма навчання»

|  |  |  |
| --- | --- | --- |
| Назва | Архів | Дата архіву |
| денна | ні | - |

Таблиця 3.8 – Склад таблиці «Спеціальність»

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Назва | Коротка назва | Код | Спеціалізація | Галузь знань | Кафедра | Архів | Дата архіву |
| програмне забезпечення систем | - | 7.05010301 | - | - | - | ні | - |

Таблиця 3.9 – Склад таблиці «Дисципліна»

|  |  |  |  |
| --- | --- | --- | --- |
| Назва | Коротка назва | Архів | Дата архіву |
| Дискретна математика | - | ні | - |
| Історія України | - | ні | - |
| Іноземна мова | - | ні | - |
| Операційні системи | - | ні | - |

Таблиця 3.10 – Склад таблиці «Дисципліна\_Кафедра\_Спеціальність»

|  |  |  |
| --- | --- | --- |
| Дисципліна | Кафедра | Спеціальність |
| Дискретна математика | - | програмне забезпечення систем |
| Історія України |  | програмне забезпечення систем |
| Операційні системи |  | програмне забезпечення систем |
| Іноземна мова |  | програмне забезпечення систем |

Склад таблиці «Навантаження документу навчальний план»:

**Запис1:**

Дисципліна: Дискретна математика; Дисципліна на півсеместру: ні;

Номер дисципліни: 1; Кількість годин на лекції: 64;

Кількість годин на лабораторні: 0; Кількість годин на практичні: 32;

Екзамен: 2; Залік: 1; Індивідуальне завдання: -;

Кількість годин на самостійну роботу: 114; Всього годин: 210;

Семестр1: 201; Семестр2: 201; Семестр3: 0; Семестр4: 0; Семестр5: 0;

Семестр6: 0; Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0;

Семестр11: 0; Семестр12: 0.

**Запис2:**

Дисципліна: Історія України; Дисципліна на півсеместру: так;

Номер дисципліни: 2; Кількість годин на лекції: 32;

Кількість годин на лабораторні: 0; Кількість годин на практичні: 16;

Екзамен: -; Залік: 1; Індивідуальне завдання: -;

Кількість годин на самостійну роботу: 72; Всього годин: 120;

Семестр1: 201; Семестр2: 0; Семестр3: 0; Семестр4: 0; Семестр5: 0;

Семестр6: 0; Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0;

Семестр11: 0; Семестр12: 0.

**Запис3:**

Дисципліна: Іноземна мова; Дисципліна на півсеместру: ні;

Номер дисципліни: 3; Кількість годин на лекції: 0;

Кількість годин на лабораторні: 0; Кількість годин на практичні: 48;

Екзамен: -; Залік: 1; Індивідуальне завдання: -;

Кількість годин на самостійну роботу: 72; Всього годин: 0; Семестр1: 003;

Семестр2: 0; Семестр3: 0; Семестр4: 0; Семестр5: 0; Семестр6: 0;

Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0; Семестр11: 0;

Семестр12: 0.

**Запис4:**

Дисципліна: Операційні системи; Дисципліна на півсеместру: ні;

Номер дисципліни: 4; Кількість годин на лекції: 64;

Кількість годин на лабораторні: 16; Кількість годин на практичні: 16;

Екзамен: 3; Залік: - ; Індивідуальне завдання: - ;

Кількість годин на самостійну роботу: 114; Всього годин: 210;

Семестр1: 0; Семестр2: 0; Семестр3: 411; Семестр4: 0; Семестр5: 0;

Семестр6: 0; Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0;

Семестр11: 0; Семестр12: 0.

Вхідні дані тесту 3: файл навчального плану (табл. 3.11) та рис. 3.3, правила читання (табл. 3.12), таблиці бази даних порожні, окрім таблиці «Документ навчальний план» (табл. 3.13) і таблиці «Навантаження документу навчального плану».

Таблиця 3.11 – Склад файлу навчального плану, титульний лист

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рік прийому | Кваліфікаційний рівень | Термін навчання | Форма навчання | Спеціальність |
| 2015 | бакалавр | 4 | денна | 7.05010301 програмне забезпечення систем |

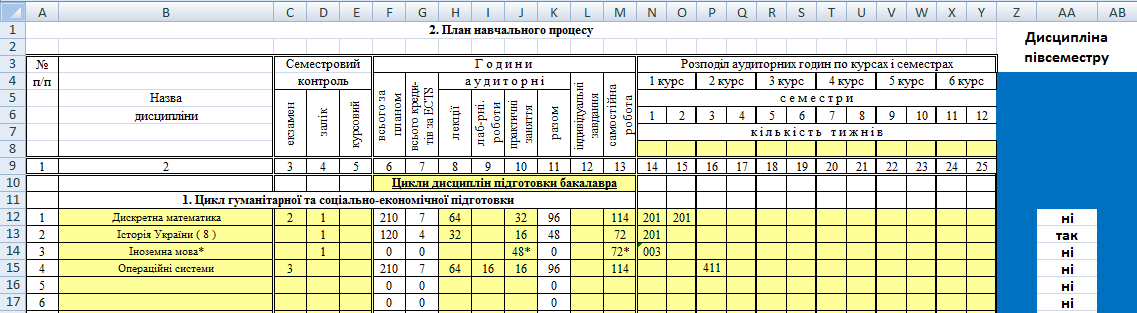


Рисунок 3.3 – Вхідні дані для тесту 3, лист навчальний процес файлу навчальний план

Таблиця 3.12 – Правила читання файлів навчальних планів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рік прийому | Кваліфікаційний рівень | Термін навчання | Форма навчання | Спеціальність |
| AS1 | AT3 | AP4 | AB11 | AB17 |

Таблиця 3.13 – Склад таблиці «Документ навчального плану»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рік прийому | Кваліфікаційний рівень | Термін навчання | Форма навчання | Спеціальність |
| 2015 | бакалавр | 4 | денна | програмне забезпечення систем |

Склад таблиці «Навантаження документу навчальний план»:

**Запис1:**

Дисципліна: Дискретна математика; Дисципліна на півсеместру: ні;

Номер дисципліни: 1; Кількість годин на лекції: 64;

Кількість годин на лабораторні: 0; Кількість годин на практичні: 32;

Екзамен: 2; Залік: 1; Індивідуальне завдання: -;

Кількість годин на самостійну роботу: 114; Всього годин: 210;

Семестр1: 201; Семестр2: 201; Семестр3: 0; Семестр4: 0; Семестр5: 0;

Семестр6: 0; Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0;

Семестр11: 0; Семестр12: 0.

**Запис2:**

Дисципліна: Історія України; Дисципліна на півсеместру: так;

Номер дисципліни: 2; Кількість годин на лекції: 32;

Кількість годин на лабораторні: 0; Кількість годин на практичні: 16;

Екзамен: -; Залік: 1; Індивідуальне завдання: -;

Кількість годин на самостійну роботу: 72; Всього годин: 120;

Семестр1: 201; Семестр2: 0; Семестр3: 0; Семестр4: 0; Семестр5: 0;

Семестр6: 0; Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0;

Семестр11: 0; Семестр12: 0.

**Запис3:**

Дисципліна: Іноземна мова; Дисципліна на півсеместру: ні;

Номер дисципліни: 3; Кількість годин на лекції: 0;

Кількість годин на лабораторні: 0; Кількість годин на практичні: 48;

Екзамен: -; Залік: 1; Індивідуальне завдання: -;

Кількість годин на самостійну роботу: 72; Всього годин: 0; Семестр1: 003;

Семестр2: 0; Семестр3: 0; Семестр4: 0; Семестр5: 0; Семестр6: 0;

Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0; Семестр11: 0;

Семестр12: 0.

**Запис4:**

Дисципліна: Операційні системи; Дисципліна на півсеместру: ні;

Номер дисципліни: 4; Кількість годин на лекції: 64;

Кількість годин на лабораторні: 16; Кількість годин на практичні: 16;

Екзамен: 3; Залік: - ; Індивідуальне завдання: - ;

Кількість годин на самостійну роботу: 114; Всього годин: 210;

Семестр1: 0; Семестр2: 0; Семестр3: 411; Семестр4: 0; Семестр5: 0;

Семестр6: 0; Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0;

Семестр11: 0; Семестр12: 0.

Вихідні дані: результат зчитування – 1; таблиці бази даних (табл. 3.14 – 3.17).

Коментар: повторне завантаження файлу, результат 1 отже така інформація вже була завантажена. Тест проходить правильно.

Таблиця 3.14 – Склад таблиці «Документ навчального плану»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рік прийому | Кваліфікаційний рівень | Термін навчання | Форма навчання | Спеціальність |
| 2015 | бакалавр | 4 | денна | програмне забезпечення систем |

Таблиця 3.15 – Склад таблиці «Кваліфікаційний рівень»

|  |  |  |
| --- | --- | --- |
| Назва | Архів | Дата архіву |
| бакалавр | ні | - |

Таблиця 3.16 – Склад таблиці «Форма навчання»

|  |  |  |
| --- | --- | --- |
| Назва | Архів | Дата архіву |
| денна | ні | - |

Таблиця 3.17 – Склад таблиці «Спеціальність»

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Назва | Коротка назва | Код | Спеціалізація | Галузь знань | Кафедра | Архів | Дата архіву |
| програмне забезпечення систем | - | 7.05010301 | - | - | - | ні | - |

Склад таблиці «Навантаження документу навчальний план»:

**Запис1:**

Дисципліна: Дискретна математика; Дисципліна на півсеместру: ні;

Номер дисципліни: 1; Кількість годин на лекції: 64;

Кількість годин на лабораторні: 0; Кількість годин на практичні: 32;

Екзамен: 2; Залік: 1; Індивідуальне завдання: -;

Кількість годин на самостійну роботу: 114; Всього годин: 210;

Семестр1: 201; Семестр2: 201; Семестр3: 0; Семестр4: 0; Семестр5: 0;

Семестр6: 0; Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0;

Семестр11: 0; Семестр12: 0.

**Запис2:**

Дисципліна: Історія України; Дисципліна на півсеместру: так;

Номер дисципліни: 2; Кількість годин на лекції: 32;

Кількість годин на лабораторні: 0; Кількість годин на практичні: 16;

Екзамен: -; Залік: 1; Індивідуальне завдання: -;

Кількість годин на самостійну роботу: 72; Всього годин: 120;

Семестр1: 201; Семестр2: 0; Семестр3: 0; Семестр4: 0; Семестр5: 0;

Семестр6: 0; Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0;

Семестр11: 0; Семестр12: 0.

**Запис3:**

Дисципліна: Іноземна мова; Дисципліна на півсеместру: ні;

Номер дисципліни: 3; Кількість годин на лекції: 0;

Кількість годин на лабораторні: 0; Кількість годин на практичні: 48;

Екзамен: -; Залік: 1; Індивідуальне завдання: -;

Кількість годин на самостійну роботу: 72; Всього годин: 0; Семестр1: 003;

Семестр2: 0; Семестр3: 0; Семестр4: 0; Семестр5: 0; Семестр6: 0;

Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0; Семестр11: 0;

Семестр12: 0.

**Запис4:**

Дисципліна: Операційні системи; Дисципліна на півсеместру: ні;

Номер дисципліни: 4; Кількість годин на лекції: 64;

Кількість годин на лабораторні: 16; Кількість годин на практичні: 16;

Екзамен: 3; Залік: - ; Індивідуальне завдання: - ;

Кількість годин на самостійну роботу: 114; Всього годин: 210;

Семестр1: 0; Семестр2: 0; Семестр3: 411; Семестр4: 0; Семестр5: 0;

Семестр6: 0; Семестр7: 0; Семестр8: 0; Семестр9: 0; Семестр10: 0;

Семестр11: 0; Семестр12: 0.

3.2.2 Тестування методу FormationDocs класу WorkingCurriculumDocService

3.2.2.1 Опис методу

Метод призначений для формування робочих навчальних планів. Вхідні дані: вміст бази даних, а саме таблиці:

* CurriculmDoc;
* Qualification;
* FormEducation;
* Speciality;
* WorkingCurriculumDoc;
* Group.

Вихідні дані: результат формування, база даних.

3.2.2.2 Вибір методу тестування

Оскільки користувач ніяк не може вплинути на перебіг формування, єдиним методом тестування залишається метод припущення про помилку.

3.2.2.3 Формування набору тестів

Припускається, що при формуванні робочих навчальних планів, зустрінеться такий навчальний план в якому рік прийому буде більше нинішнього або значно менше.

Вхідні дані тесту 1: таблиці бази даних (табл. 3.17 – 3.20) інші таблиці порожні.

Таблиця 3.17 – Склад таблиці «Документ навчального плану»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рік прийому | Кваліфікаційний рівень | Термін навчання | Форма навчання | Спеціальність |
| 3015 | бакалавр | 4 | денна | програмне забезпечення систем |

Таблиця 3.18 – Склад таблиці «Кваліфікаційний рівень»

|  |  |  |
| --- | --- | --- |
| Назва | Архів | Дата архіву |
| бакалавр | ні | - |

Таблиця 3.19 – Склад таблиці «Форма навчання»

|  |  |  |
| --- | --- | --- |
| Назва | Архів | Дата архіву |
| денна | ні | - |

Таблиця 3.20 – Склад таблиці «Спеціальність»

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Назва | Коротка назва | Код | Спеціалізація | Галузь знань | Кафедра | Архів | Дата архіву |
| програмне забезпечення систем | - | 7.05010301 | - | - | - | ні | - |

Вихідні дані тесту 1: результат формування – 0; таблиці бази даних не змінилися.

Коментар: тест проходить успішно, припущення – невірне.

Вхідні дані тесту 2: таблиці бази даних (табл. 3.21 – 3.24) інші таблиці порожні.

Таблиця 3.21 – Склад таблиці «Документ навчального плану»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рік прийому | Кваліфікаційний рівень | Термін навчання | Форма навчання | Спеціальність |
| 1015 | бакалавр | 4 | денна | програмне забезпечення систем |

Таблиця 3.22 – Склад таблиці «Кваліфікаційний рівень»

|  |  |  |
| --- | --- | --- |
| Назва | Архів | Дата архіву |
| бакалавр | ні | - |

Таблиця 3.23 – Склад таблиці «Форма навчання»

|  |  |  |
| --- | --- | --- |
| Назва | Архів | Дата архіву |
| денна | ні | - |

Таблиця 3.24 – Склад таблиці «Спеціальність»

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Назва | Коротка назва | Код | Спеціалізація | Галузь знань | Кафедра | Архів | Дата архіву |
| програмне забезпечення систем | - | 7.05010301 | - | - | - | ні | - |

Вихідні дані: результат формування – 0; таблиці бази даних не змінилися.

Коментар: Тест проходить успішно, припущення – невірне.

3.2.3 Тестування методу Formation класу LoadDepartmentService

3.2.3.1 Опис методу

Метод призначений для формування документів «Розподіл навантаження кафедри». Вхідні дані: вміст бази даних, а саме таблиці:

* WorkingCurriculmDoc;
* LoadWorkingCurriculum;
* SubjectDepartmentSpeciality;
* Department;
* LoadDepartmentDoc;
* LoadDepartmentDocLoads.

Вихідні дані: результат формування, база даних.

3.2.3.2 Вибір методу тестування

Так як і в попередньому методі оскільки користувач ніяк не може вплинути на перебіг формування, єдиним методом тестування залишається метод припущення про помилку.

3.2.3.3 Формування набору тестів

Помилка може виникнути лише, якщо некоректні дані в робочих навчальних планах, або якщо їх нема. Тому припускається, що при формуванні документів «Розподіл навантаження кафедри» не буде знайдено робочих навчальних планів.

**Тест1:**

Вхідні дані: таблиці порожні.

Вихідні дані: результат формування – 1, таблиці бази даних залишились порожніми.

Коментар: так як робочі навчальні плани не були знайдені, то результат формування 1, а вміст бази даних не змінився.

3.2.4 Результати тестування

3.2.4.1 Результати тестування методу ReadDocFromExcel класу CurriculumDocService

В ході виконання набору тестів були покриті всі оператори (табл. 3.25).

Таблиця 3.25 – Результати тестування методу ReadDocFromExcel класу CurriculumDocService

|  |  |  |  |
| --- | --- | --- | --- |
| Гілка Тест | 1 | 2 | 3 |
| 1 | + |  |  |
| 2 |  | + | + |
| 3 |  | + | + |
| 4 |  | + | + |
| 5 |  | + | + |
| 6 |  | + | + |
| 7 |  | + |  |
| 8 |  | + |  |
| 9 |  | + |  |
| 10 |  | + |  |
| 11 |  | + |  |
| 12 |  | + |  |

Продовження таблиці 3.25

|  |  |  |  |
| --- | --- | --- | --- |
| Гілка Тест | 1 | 2 | 3 |
| 13 |  | + |  |
| 14 |  | + |  |
| 15 |  | + |  |
| 16 |  | + |  |
| 17 |  | + |  |
| 18 |  | + |  |
| 19 |  | + |  |
| 20 |  | + |  |
| 21 |  | + |  |
| 22 |  | + |  |
| 23 |  | + |  |
| 24 |  | + |  |
| 25 |  | + |  |
| 26 |  | + |  |
| 27 |  | + |  |
| 28 |  | + |  |
| 29 |  | + |  |
| 30 |  | + |  |
| 31 |  | + |  |
| 32 |  | + |  |
| 33 |  | + |  |
| 34 |  | + |  |
| 35 |  | + |  |
| 36 |  | + |  |
| 37 |  | + |  |
| 38 |  | + |  |
| 39 |  |  | + |
| 40 | + |  |  |

3.2.4.2 Результати тестування методу FormationDocs класу WorkingCurriculumDocService

Після автоматичного формування документів робочих навчальних планів аномалій і аномальних даних не виявлено, отже тест завершився вдало.

3.2.4.3 Результати тестування методу Formation класу LoadDepartmentService

Після автоматичного формування документів «Розподіл навантаження кафедри» база залишилась порожньою, отже тест завершився вдало.

3.3 Налагодження програми

Під час копіювання папки з усім вмістимим, було викинуте невідоме виключення. Необхідно провести налагодження програми, для усунення даної помилки [24].

Текст методу, що відлагоджується:

private void CopyDirectory(string from, string to)

{

Directory.CreateDirectory(to);

foreach (string item in Directory.GetFiles(from))

{

File.Copy(item, to + "\\" + Path.GetFileName(item));

}

foreach (string item in Directory.GetDirectories(from))

{

CopyDirectory(item, to + "\\" + Path.GetFileName(item));

}

}

Спершу необхідно знайти оператор, на якому викидається виключення. Для цього в меню підменю «Виключення» меню «Відладка» включимо наступні опції у розділі «Перехоплювати виключення не перехоплені кодом користувача»:«Common Language Runtime Exceptions», «Managed Debugging Assistants», «Native Runtime Checks», «Win32 Exceptions». Після цього повторивши дії, що призводять до виникнення помилки, налагодження зупиниться у місці помилки.

У результаті виконання цих дій було виявлено, що виключення викидається на рядку Directory.CreateDirectory(to).

Є два варіанти можливої причини:

* така папка вже існує;
* некоректний шлях.

Для перевірки першого припущення поставимо точку зупинки на методі, в якому викидається виключення (рис. 3.4). Повторимо дії, що призводять до виникнення помилки. Робота програми зупинилася без викиду виключень у цій точці. Після виконання ще одного кроку програми з’явилася помилка. Це доводить вірність перевіреного припущення.

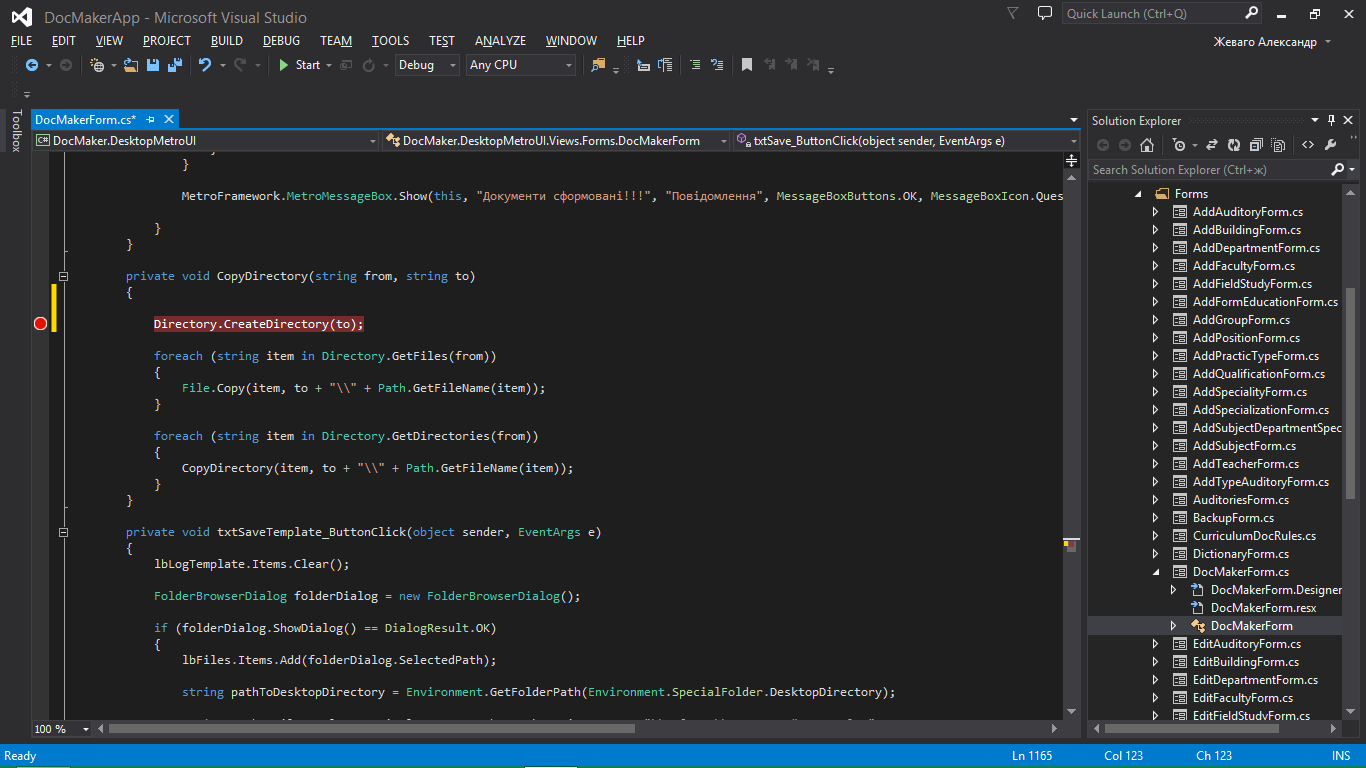


Рисунок 3.4 – Додана точка зупинки

Коли помилка локалізована необхідно визначитись з методом її виправлення. Є 2 найпростіші варіанти вирішення проблеми. Можна «заглушити» переривання порожнім блоком «try-catch». Таке рішення не є вдалим, оскільки механізм перехоплення помилок є доволі затратним по часу і його використання у рекурсії погіршить швидкодію системи. Крім цього можна поставити додаткову умову перевірки на існування папки. Другий варіант є більш прийнятним, його і приймемо.

В результаті код буде виглядати наступним чином:

private void CopyDirectory(string from, string to)

{

if (!Directory.Exists(to))

{

Directory.CreateDirectory(to);

}

foreach (string item in Directory.GetFiles(from))

{

File.Copy(item, to + "\\" + Path.GetFileName(item));

}

foreach (string item in Directory.GetDirectories(from))

{

CopyDirectory(item, to + "\\" + Path.GetFileName(item));

}

}

Після внесення необхідних змін до коду прибираємо точку зупинки та повторюємо дії, що призводили до помилки. Помилка не виникла, можна зробити висновок, що вона була виправлена.

4 Охорона праці та безпека у надзвичайних ситуаціях

«Система доступу до енциклопедичних знань на природній мові» є програмним продуктом, який дозволить використовувати енциклопедичну інформацію з навчальною або довідковою метою. Інтерфейс взаємодії з системою реалізований через систему текстових або голосових запитів та відповідей. Це стане у нагоді як звичайним користувачам, так і користувачам з обмеженими можливостями. Під час роботи з даною системою, користувач буде працювати з персональним комп’ютером, клавіатурою, монітором, або ноутбуком, тому слід зазначити правила користування електронно-обчислювальними машинами та відео-дисплейними терміналами.

5.1 Аналіз шкідливих та небезпечних виробничих факторів

Небезпечні виробничі чинники – це виробничі чинники, вплив яких на користувача в певних умовах призводить до травм, гострого отруєння або іншого раптового різкого погіршення здоров'я або до смерті [8].

До різкого погіршення здоров'я можна віднести отруєння, опромінення, удар електрострумом, тепловий удар та ін.

До небезпечних факторів відносяться електронебезпека та пожежонебезпека.

Шкідливі виробничі фактори – фактори, тривалий вплив яких на працюючого у визначених умовах приведе до захворювання, зниження працездатності i (або) негативного впливу на здоров'я нащадків [8]. У залежності від рівня і тривалості впливу шкідливі фактори можуть класифікуватися і як небезпечні.

Згідно санітарно-гігієнічним вимогам [9] умови праці робітника, який працює з автономною частиною «Системи доступу до енциклопедичних знань на природній мові» на ЕОМ, повинні відповідати I або II класу.

Робота на ЕОМ пов'язана з наступними шкідливими факторами:

* недолік освітлення природним світлом;
* відблиски на екрані монітора;
* електромагнітне поле.

Недостатнє освітлення приводить до швидкої стомленості очей, що у свою чергу, призводить до зниження продуктивності праці і росту кількості прийнятих помилкових рішень.

Відблиски на екрані електронно-променевого монітора, що виникають при неправильному освітленні, приводить до погіршення зору, а у випадку тривалого впливу даного небезпечного фактору, може призвести до повної втрати зору. З метою зниження рівня впливу на робітника даного шкідливого фактору, варто дотримуватись вимог [10] або застосовувати рідкокристалічні монітори, які в силу своєї конструкції і використовуваних матеріалів мають менший коефіцієнт відбиття світла, ніж електронно-променевий монітор, тому відблисків на них практично не буває.

Рівні електромагнітного випромінювання і магнітних полів на робочому місці повинні відповідати вимогам [11].

5.1.1 Небезпечні чинники пов’язані з пожежами в будівлях університету

Пожежі становлять особливу небезпеку для життя людини, і можуть призвести до великих матеріальних втрат. Джерелами загоряння можуть виявитися електронні схеми ЕОМ, елементи, що перегрілися, які здатні викликати загоряння пальних матеріалів.

Для всіх споруджень та приміщень, у яких експлуатується автономна частина «Системи доступу до енциклопедичних знань на природній мові», повинна бути визначена категорія по вибухонебезпечній і пожежній безпеці відповідно до [12]. Відповідні позначення повинні бути нанесені на вхідні двері приміщення.

Споруди та їх приміщення, у яких розміщуються ЕОМ, повинні мати ступінь вогнестійкості не нижче II. Приміщення для обслуговування, ремонту і налагодження ЕОМ повинні відноситися:

* по пожежонебезпеки до категорії В – пожежонебезпечні приміщення, де розташовуються тверді пальні речовини (ТПР);
* по класу приміщення до категорії П ІІ.

Стіни приміщень з ЕОМ виготовляються з не пальних матеріалів. Підходи до засобів пожежогасіння повинні бути вільними.

5.1.2 Електронебезпечні чинники

При роботі з автономною частиною «Системи доступу до енциклопедичних знань на природній мові» на ЕОМ можуть трапитися нещасні випадки, зв'язані з ураженням електричним струмом, викликаним дотиком до оголених місць струмоведучих частин устаткування, або частин, що знаходяться під напругою.

Вплив струму на людину проявляється по-різному, в залежності від його величини:

* струм до 0,6 мА не відчувається людиною;
* струм завбільшки 6 мА призводить до скорочення м'язів тієї частини, тіла, що піддалася його впливу;
* струм 50 мА вражає органи дихання та серцево-судинну систему;
* при досягненні струму порога 100мА настає фібриляція серця і, потім, його зупинка;
* при впливі на тіло людини струму в 3-4 А виникає обвуглювання ділянок тіла.

За способом захисту людини від ураження електричним струмом відео термінали, ЕОМ, периферійні пристосування ЕОМ і оснащення для обслуговування, ремонту і налагодження ЕОМ повинні відповідати I класу захисту. Вимоги електричної і механічної безпеки і методи іспитів чи повинні бути заземлені.

Лінія електромережі для живлення ЕОМ, периферійних пристроїв ЕОМ і устаткування для обслуговування, ремонту і налагодження ЕОМ виконується як окрема групова, трьох провідна мережа, шляхом прокладки фазового, нульового робочого і нульового захисного провідників.

Підключення на розподільному щиті до одного контактного затиску нульового робочого і нульового захисного провідників заборонено.

Площа перетину нульового робочого і нульового захисного провідника в груповій трьох провідній мережі повинна бути не менш площі перетину фазового провідника.

У приміщенні, де одночасно експлуатується чи обслуговується більш п'яти персональних ЕОМ, на видному доступному місці встановлюється аварійний вимикач, за допомогою якого можливо зробити знеструмлення приміщення (за винятком освітлення).

5.2 Організаційні і технічні заходи щодо захисту користувачів від шкідливих і небезпечних факторів.

Для якісної і зручної роботи з автономною частиною «Системи доступу до енциклопедичних знань на природній мові», необхідне проведення проектних заходів: правильна облаштованість, належне дотримання ергономічних характеристик основних елементів робочого місця [14], санітарно-гігієнічних вимог і т.п.

Облаштованість робочого місця користувача при роботі з підсистемою повинне забезпечувати відповідність всіх елементів робочого місця і їхнього розташування.

Площа, яка виділена для одного робочого місця з ЕОМ, повинна складати не менш 6 м2 , а обсяг – не менш 20 м3.

Вимоги до організації робочого місця

При роботі з автономною частиною «Системи доступу до енциклопедичних знань на природній мові» необхідно виконувати вимоги до організації робочого місця, що наведені нижче.

До самостійної роботи допускаються особи, які пройшли медичний огляд, навчання за професією, вступний інструктаж з охорони праці на робочому місці.

Конструкція робочого місця робітника при роботі з ЕОМ (при роботі сидячи) повинна забезпечувати підтримку оптимальної робочої пози з наступними ергономічними характеристиками:

* стопи ніг – на підлозі чи на підставці для ніг;
* стегно – у горизонтальній площині;
* передпліччя – вертикально;
* лікті під кутом 70о-90о до вертикальної площини;
* зап'ястя – зігнуті під кутом не більш 20о щодо горизонтальної площини;
* нахил голови 15о-20о відносно вертикальної площини.

Дисплей та клавіатура мають розташовуватися на оптимальній відстані від очей користувача, але не ближче 600 мм, з урахуванням розміру алфавітно-цифрових знаків та символів.

Розташування екрану відео терміналу має забезпечувати зручність зорового спостереження у вертикальній під кутом ±30° від лінії зору користувача.

Клавіатуру слід розміщувати на поверхні столу або на спеціальній, робочій і такій, яка регулюється за висотою поверхні окремо від столу на відстані 100-300 мм від краю, ближчого до користувача. Кут нахилу клавіатури має бути в межах 5-15°.

5.3 Норми пожежної безпеки на робочому місці користувача

Вимоги безпеки праці перед початком роботи на ЕОМ

Перед початком роботи з автономною частиною «Системи доступу до енциклопедичних знань на природній мові» робітник інструктується первинним інструктажем. Первинний інструктаж завжди проводиться на робочому місці з безпосереднім показом робіт (стажування 1 місяць). Потім, через кожні 6 місяців проводиться повторний інструктаж. Результати інструктажу заносяться в "Журнал реєстрації інструктажів з питань охорони праці". У журналі після проходження інструктажу повинен бути підпис людини, яка інструктувала та робітника. Далі підготовка робочого місця повинна виконується відповідно до нижче приведених пунктів:

* увімкнути систему кондиціонування повітря в приміщенні;
* оглянути робоче місце і привести його в порядок: переконатися, що на ньому немає сторонніх предметів; все устаткування і блоки ЕОМ з'єднані із системним блоком за допомогою сполучних шнурів;
* перевірити надійність установки апаратури на робочому столі. Монітор повинен стояти не на краю столу. Повернути монітор так, щоб було зручно дивитися на екран – під прямим кутом (а не збоку) – та ледь зверху вниз, при цьому екран повинен бути ледь нахилений – нижній його край ближче до робітника;
* перевірити загальний стан апаратури, або перевірити чи справні електропроводка, сполучні шнури, штепсельні вилки, розетки, перевірити заземлення захисного екрана;
* відрегулювати освітлення робочого місця;
* відрегулювати і зафіксувати висоту крісла, зручний для робітника нахил його спинки;
* включити апаратуру комп'ютера перемикачами на корпусі в послідовності: стабілізатор напруги, монітор, системний блок;
* відрегулювати яскравість світіння екрана до яскравості навколишніх його поверхонь у робочій зоні – і не більше 3:1;
* при виявленні будь-яких несправностей роботу не починати, повідомити про це керівника робіт.

Вимоги безпеки праці під час роботи на ЕОМ

Під час роботи з автономною частиною «Системи доступу до енциклопедичних знань на природній мові» необхідно дотримуватися наступних правил:

* при виконанні роботи на комп’ютері слід сидіти прямо, не напружуватися;
* не дозволяються сторонні розмови, подразнюючі шуми;
* постійно слідкувати за функціонуванням систем опалення і кондиціювання;
* для зниження напруженості праці на комп’ютері необхідно рівномірно розподілити і чергувати характер робіт відповідно до їх складності.

Для зменшення негативного впливу на здоров'я працюючих виробничих факторів необхідно застосовувати регламентні перерви. У таблиці 5.1 приведений час регламентованих перерв робітника у залежності від категорії і групи робіт.

Таблиця 5.1 Час регламентованих перерв операторів (користувачів) в залежності від категорії і групи робіт

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категорія  робіт | Група роботи | | | Час перерви при 8-годинній зміні, хв. |
| А, кількість  знаків | Б, кількість  знаків | В, година |
| Оптимальна-І | до 20 000 | до 15000 | до 2 | 20 |
| Припустима-ІІ | 21 000-40 000 | до 30 000 | до 4 | 40 |
| Важка-ІІІ | понад 40 000  не більш 60 000 | понад 30 000  не більш 45 000 | понад 4  не більш 6 | 60 |

Група А – читання інформації з попередні запитом (діалоговий режим роботи).

Група Б – введення інформації.

Група В – творча робота в режимі діалогу ЕОМ (налагодження програм, переклад і редагування текстів і т, д.).

При восьмигодинній робочій зміні регламентована перерва повинна бути такою:

* для I категорії робіт – через 2 години від початку зміни і через 2 години після обідньої перерви (кожен тривалістю 10 хв.);
* для II категорії робіт – через 2 години від початку зміни (тривалістю 15 хв.), через 1,5 і 2,5 годин після обідньої перерви (15 і 10 хв. відповідно чи тривалістю 5-10 хв. Через щогодини роботи, в залежності від характеру технологічного процесу);
* для III категорії робіт – через 2 години від початку зміни (тривалістю 15 хв.), через 1,5 і 2,5 годин після обідньої перерви (тривалістю 20 хв. Кожний тривалістю 5-15 хв. Щогодини роботи, у залежності від характеру технологічного процесу);
* під час нічної зміни, незалежно від групи і категорії робіт, тривалість регламентованих переривів збільшується на 60 хв.

Під час роботи забороняється:

* використовувати рідинні або аерозольні засоби для чищення поверхонь комп’ютера, крім передбачених для цього засобів (для чищення пластмасових поверхонь використовують чисту бавовняну ганчірку, зволожену мильним розчином; екрани та захисні екрани протирають ватою, змоченою спиртом);
* самостійно ремонтувати апаратуру;
* класти будь-які речі на вузли комп’ютера;
* закривати вентиляційні отвори апаратури, що може привести до її перегрівання і виходу з ладу.

Після закінчення роботи необхідно:

* вийти з програмної оболонки і повернутися в середовище операційної системи;
* вимкнути системний блок, принтер, інші периферійні пристрої, монітор;
* вимкнути стабілізатор живлення, якщо комп'ютер підключений до мережі через нього;
* штепсельні вилки витягнути з розетки;
* накрити клавіатуру кришкою для запобігання потрапляння на неї пилу;
* прибрати робоче місце.

5.4 Вимоги безпеки праці

Перед початком роботи з ПК користувач зобов'язаний: провітрити робоче приміщення; перевірити стійкість положення устаткування на робочому столі; перевірити відсутність видимих пошкоджень обладнання, дискет в дисководі системного блоку; перевірити справність і цілісність живлять і сполучних кабелів, роз'ємів і штепсельних з'єднань, захисного заземлення (занулення); перевірити справність.

Відрегулювати положення столу, стільця (крісла), підставки для ніг, клавіатури, екрана монітора; освітленість на робочому місці. При необхідності включити місцеве освітлення; протерти поверхню екрану монітора, захисного фільтра (за його наявності) сухою м'якою тканинною серветкою.

Переконатися у відсутності відображень на екрані монітора, зустрічного світлового потоку. Включити обладнання ПК в електричну мережу, дотримуючись наступну послідовність: стабілізатор напруги (якщо він використовується), блок безперебійного живлення, периферійні пристрої (принтер, монітор, сканер та інші пристрої), системний блок.

Забороняється приступати до роботи при: вираженому тремтінні зображення на моніторі; виявленні несправності обладнання; наявності пошкоджених кабелів або проводів, роз'ємів, штепсельних з'єднань; відсутності або несправності захисного заземлення (занулення) обладнання.

У період роботи за ПЕОМ необхідно передбачати перерви для відпочинку, які мають бути оптимальної тривалості (надто тривалі ведуть до порушення готовності до дії та розладу динамічного стереотипу). Тому, через кожні 40-45 хвилин необхідно робити перерву на 3-хвилини, а через 2 години – на 15 хвилин. Середня сумарна тривалість роботи за ПЕОМ на день не повинна перевищувати 4 годин, а на тиждень – 20 годин.

При використанні захисного екрана або монітора із зниженим випромінюванням час роботи за ПЕОМ може бути збільшено.

Шкідливою є одна й та сама поза протягом тривалого часу. Тому в положенні сидячі треба час від часу змінювати фіксовані робочі пози, робити короткочасні перерви.

Під час роботи на ПЕОМ напружуються м'язи рук. Для підтримки їх тонусу під час перерви рекомендується проводити гімнастичні вправи.

З метою зниження негативного впливу монотонної діяльності доцільно чергувати операції введення тексту і цифрових даних, редагування тексту.

Забороняється залишати ввімкнені ПЕОМ і пристрої без нагляду. Підключення і відключення роз'ємних кабелів пристроїв ПЕОМ робити тільки при відключенні їх з мережі.

Не можна користуватися біля ПЕОМ аерозолями (дезодорантами, тощо). Не допускати попадання води та інших рідин в середину пристроїв комп'ютера. При наявності електроструму на корпусі припинити роботу, вимкнути ПЕОМ від електромережі, сповістити керівника.

У випадку виникнення у каристувача зорового або дискомфорту інших неприємних суб'єктивних відчуттів, що настають, незважаючи на проходження санітарно-гігієнічних і ергономічних вимог, режимів роботи і відпочинку, варто застосувати індивідуальний підхід в обмеженні часу робіт і корекції тривалості перерв для відпочинку, або проводити заміну іншими видами робіт (не пов'язаних з використанням комп'ютера).

По закінченні роботи користувач зобов'язаний: коректно закрити всі активні завдання; вимкнути живлення системного блоку; вимкнути живлення усіх периферійних пристроїв; відключити блок безперебійного живлення; відключити кабель живлення від мережі; привести в порядок робоче місце; про несправність обладнання та інших зауважень по роботі з ПК повідомити безпосереднього керівника або особам, що здійснюють технічне обслуговування обладнання.

5.5 Дії при аварійних ситуаціях

Після закінчення роботи необхідно:

* при раптовому припиненні подавання електроенергії вимкнути послідовно периферійні пристрої, ВТП, процесор і витягнути штепсель вилки з розетки;
* при виявленні ознак горіння (дим, запах горіння), вимкнути апаратуру, знайти джерело займання і вжити заходів для його ліквідації, повідомити керівництво чи системного адміністратора;
* в разі виникнення пожежі негайно повідомити пожежну частину, вжити необхідних заходів для евакуації людей згідно з планом евакуації по підприємству і приступити до гасіння первинними засобами пожежогасіння.

5.5.1 Допомога при уражені електричним струмом

Перша медична допомога при ураженні людини електричним струмом повинна бути спрямована на звільнення її від дії струму та підтримку або відновлення серцевої діяльності і дихання.

Якщо потерпілий при свідомості, але не спроможний розтиснути руки, йому слід запропонувати підскочити або впасти, щоб розірвати електрострумовий ланцюг.

Коли уражений без свідомості, то потрібно негайно знеструмити його шляхом вимикання напруги в мережі рубильником, вимикачем або запобіжником. При неможливості достатньо швидко вимкнути електромережу слід перерізати чи перерубати електропроводи.

При ситуації, коли потерпілого можна відтягти від ушкоджуючи струмоведучих частин, до яких він торкається, рятівник повинен подбати про власну безпеку: по можливості діяти однією рукою, тягнути тільки за суху одежу, уникати доторкання до оголених частин тіла потерпілого та оточуючих металевих деталей. Електроізолюватися від потерпілого можна за допомогою діелектричних рукавиць або обмотаними тканиною руками, ставши на гумовий килимок, суху дошку або інший підручний предмет, що має діелектричні властивості. Відокремивши ураженого від струмопроводу, потрібно ретельно оглянути його.

Якщо в потерпілого наявні ознаки клінічної смерті, то слід невідкладно розпочати серцево-легеневу реанімацію. При "несправжній смерті" єдиним ефективним заходом першої допомоги є негайне проведення штучного дихання (з рота в рот); інколи його доводиться робити протягом кількох годин поспіль. Штучне дихання при працюючому серці, як правило, швидко покращує стан ураженого: шкіра стає нормального кольору, визначаються пульс і артеріальний тиск. Серцево-легеневу реанімацію або тільки штучне дихання по можливості потрібно поєднувати з підшкірним введенням 2 мл кордіаміну або 1 мл 10% розчину кофеїну.

Непритомну людину в подальшому слід утеплити, напоїти великою кількістю води, чаю чи компоту (давати алкоголь і каву шкідливо) та відправити на транспорті до лікарні. При необхідності реанімаційні заходи чи штучне дихання слід безперервно продовжувати і в транспорті.

Під час транспортування до лікарні потрібно особливо уважно стежити за станом потерпілого, тому що в будь-який час у нього може виникнути зупинка дихання і серцевої діяльності. Слід бути напоготові до реанімаційних заходів у дорозі.

# висновки

Під час виконання роботи, було досліджено та проаналізовано учбові процеси вищого навчального закладу, їх зв’язок та вплив на процес формування документів для складання розкладу занять університету. Були знайдені проблеми та недоліки цих процесів та запропоновані зміни, що допоможуть раціоналізувати ці процеси.

Впровадження запропонованих механізмів дозволить автоматично формувати документацію, що покращить якість розкладу занять університету та зменшить час, що витрачається зараз на ці процеси.

Для задач формування документації була розроблена автоматизована система. Для того, щоб система максимально відповідала поставленим вимогам, була зручною та функціональною, перед її проектуванням та розробкою були розглянуті та проаналізовані існуючі аналоги та літературні джерела. В розглянутих аналогах були виявлені недоліки та переваги. Деякі з рішень, що були знайдені в цих системах, були використанні під час проектування та розробки автоматизованої системи.

Розроблена автоматизована система буде взаємодіяти з системою складання розкладу занять. Ця особливість була врахована під час проектування системи. Проектування та розробка системи проводилась з використанням сучасних методологій та підходів до вирішення цих питань.

Проектування системи відбувалося з використанням об’єктно-орієнтованого підходу. Використання шаблонів проектування та окремі дизайнерські рішення розробника дозволили зробити систему гнучкою, що легко піддається модифікації, зрозумілою та надійною.

Використання сучасних гнучких методологій розробки дозволить і надалі розширювати функціональність системи або вносити зміни до її поточної функціональності.

Інтерфейс програми було спроектовано з урахуванням потреб цільового користувача. Було досягнуто основної мети зробити інтерфейс зручним та зрозумілим. Інтерфейс мав бути спроектований так, щоб будь які дії займали у користувача мінімальну кількість часу, адже планується велика інтенсивність використання системи.

Формат вихідної документації був складений таким чином, щоб він співпадав з форматом, що на даний час використовується учбовим відділом.

Велику увагу було приділено тестуванню та відлагодженню системи. Використання сучасних інструментів тестування та налагодження та використання різних підходів та методів тестування, що найкраще підходять до тестування кожного окремого методу дозволили зробити систему надійною та стійкою до можливих помилок.

Для безпечної роботи з автоматизованою системою, були дослідженні шкідливі та небезпечні фактори, що можуть виникнути під час її функціонування. Були розроблена система вимог, прийомів та інструкцій, що дозволить зменшити ці фактори та зробити роботу з системою максимально безпечною.

Після закінчення розробки системи, була написана уся необхідна програмна документація. Наявність якісного керівництва користувача дозволить методистам учбового відділу, що будуть використовувати систему, швидко вивчити можливості програми та почати її використовувати. Задокументований опис та текст програми спростить подальше супроводження системи.

# ЛІТЕРАТУРА

1. Google (пошуковик) [https://uk.wikipedia.org/wiki/Google\_(пошуковик)](https://uk.wikipedia.org/wiki/Google_(%D0%BF%D0%BE%D1%88%D1%83%D0%BA%D0%BE%D0%B2%D0%B8%D0%BA))
2. <http://google.com>
3. Яндекс Пошук [https://uk.wikipedia.org/wiki/Яндекс.Пошук](https://uk.wikipedia.org/wiki/%D0%AF%D0%BD%D0%B4%D0%B5%D0%BA%D1%81.%D0%9F%D0%BE%D1%88%D1%83%D0%BA)
4. <http://yandex.ua>
5. <https://yandex.ru/support/search/search-wizards/wizards.xml>
6. <http://start.csail.mit.edu/>
7. Семантична павутина <https://uk.wikipedia.org/wiki/Семантична_павутина>
8. RDF <https://uk.wikipedia.org/wiki/RDF>
9. SPARQL <https://uk.wikipedia.org/wiki/SPARQL>
10. SPARQL endpoints <https://www.w3.org/wiki/SparqlEndpoints>
11. AWS SimpleDB <https://aws.amazon.com/ru/simpledb/>
12. Amazon SimpleDB <https://uk.wikipedia.org/wiki/Amazon_SimpleDB>
13. SimpleDB - Boto 3 Docs 1.3.1 Documentation <http://boto3.readthedocs.io/en/latest/reference/services/sdb.html>

# (old) ЛІТЕРАТУРА

9. Крёнке Д. Теория и практика построения баз данных. 8-е изд. / Дэвид Крёнке. – СПб.: Питер, 2003. – 800 с.

10. Нейгел, К. C# 4.0 і платформа .NET Framework 4 для професіоналів: Пер. с англ. [Текст] / К. Нейгел, Б. Івьєн, Д. Глінн, К. Уотсон. – М.: ТОВ "В.Д. Вильямс", 2011. – 1440 с.: іл.

11. Ріхтер, Дж. Програмування на платформі Microsoft .NET Framework, 2-е видання [Текст] / Дж. Рихтер. – М.: Русская редакция, 2003. – 796 с.

12. Ріхтер, Дж. CLRviaC#. Програмування на платформі Microsoft .NET Framework 3.0 на мові C# [Текст] / Дж. Рихтер. – СПб.: Пітер, 2007. – 656 с.

13. Троелсен, Е. C# 2008 и платформа .NET 3.5 [Текст] / Е. Троелсен. – СПб.: Пітер, 2008. – 796 с.

14. Буч Г. Объектно-ориентированное проектирование с примерами / Г. Буч. – М.: Конкорд, 1992. – 519 с.

15. Ларман К. Применение UML и шаблонов проектирования. Введение в объектно-ориентированный анализ и унифицированный процесс UP / Крэг Ларман. – М.: Вильямс, 2004. – 624 с.

16. Макгрегор Дж. Тестирование объектно-ориентированного программного обеспечения / Джон Макгрегор, Девид Сайкс. – М.: ДиаСофт, 2002. – 417 с.

17. Закон України «Про охорону праці», нова редакція, м. Київ, від 21.11.2002 року, №229-IV.

18. ДСанПіН 5.5.6.009-98 «Влаштування і обладнання кабінетів комп'ютерної техніки в навчальних закладах та режим праці учнів на персональних комп'ютерах».

19. ДСТУ 2293-99 «Охорона праці. Терміни та визначення основних понять».

20. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень».

21. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98.

22. ДСанПіН 3.3.6.096-2002 «Державні санітарні правила і норми при роботі з джерелами електромагнітних полів».

23. НПАОП 0.00-1.31-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин».

24. ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку».

25. НАПБ В.01.050-98/920 «Правила пожежної безпеки для закладів, установ і організацій системи освіти України».