НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____2016 р.

Дипломна робота

___першого (бакалаврського)_____**рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності 7.05010102, 8.05010102 Інформаційні технології проектування
_____7.05010103, 8.05010103 Системне проектування
(код та назва спеціальності)

на тему: «Реалізація багатошарової клієнт-серверної архітектури на прикладі програми для тестування учнів середніх класів»

Виконав: студент 4 курсу, групи ДА-22
(шифр групи)

_____ Кравчук Євгеній Сергійович_____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник _____к.т.н., доц. Безносик О.Ю._ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант економічний проф. док. ек. н. Семенченко Н. В.__ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Нормоконтроль _____ст. викладач Бритов О.А._____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____
(підпис)

Київ – 2016 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський).
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2016 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Кравчуку Євгенію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) «Реалізація багатошарової клієнт-серверної архітектури на прикладі програми для тестування учнів середніх класів» керівник проекту (роботи) Безносик О.Ю., к.т.н., доц.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 12 травня 2016 р. № 50-ст

2. Строк подання студентом проекту (роботи) 08.06.2016

3. Вихідні дані до проекту (роботи)

Досліджені існуючі архітектури типу клієнт-сервер та архітектури додатків.

Створена серверна частина додатку на основі фреймворку Spring Framework та клієнт у вигляді веб-додатку з використанням AngularJS на основі спроектованої архітектури системи для тестування учнів середніх класів.

Результати досліджень мають включати наступні пункти: архітектура програмного продукту, яка відповідає заявленим вимогам, демонстрація роботи програми.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Зробити огляд існуючих архітектур.
2. Огляд існуючих програмних рішень та технологій
3. Створити програмної реалізації додатку
4. Створити програмну реалізацію додатку.
5. Функціонально-вартісний аналіз програмного продукту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Архітектура системи – плакат.
2. Архітектура Spring Framework – плакат.
3. Представлення програми – плакат.

6. Консультанти розділів проекту (роботи)[1]

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	проф. док. ек. н. Семенченко Н. В.		

[1] Консультантом не може бути зазначено керівника дипломного проекту (роботи).

7. Дата видачі завдання 01.02.2016

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Збір інформації	15.02.2016	
3	Вивчення наявних архітектур типу клієнт-сервер	28.02.2016	
4	Аналіз багатошарових архітектур	10.03.2016	
5	Аналіз способів взаємодії клієнта та сервера	15.03.2016	
6	Огляд існуючих програмних рішень	25.03.2016	
7	Оформлення дипломної роботи	31.05.2016	
8	Отримання допуску до захисту та подача роботи в ДЕК	08.06.2016	

Студент _____ Є.С. Кравчук
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ О.Ю. Безносик
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

бакалаврської роботи Кравчука Євгенія Сергійовича
на тему «Розробка багатошарової клієнт-серверної архітектури
на прикладі програми для тестування учнів середніх класів»

Тенденції до покращення якості освіти визначаються широкою інформатизацією даної сфери суспільного життя. Як результат, завдання автоматизації контролю знань учнів за допомогою програмного забезпечення постає дуже актуальною задачею, адже вирішення її дає значну вигоду в економічному аспекті та значно підвищує продуктивність навчального процесу.

Метою даної роботи є створення програми для тестування учнів середніх класів на основі багатошарової клієнт-серверної архітектури.

Основний акцент у розробці даного продукту був зроблений на побудову архітектури додатку та прикладу клієнтської частини, що демонструє його можливості, зрозумілий учням інтерфейс для пересічного користувача та високу швидкодію.

Даний додаток розроблявся з використанням фреймворків Spring Data JPA, Spring Data REST і AngularJS. Використання обраних технологій дозволило виконати поставлене завдання в повній мірі і створити основу для зручного розширення функціоналу.

Загальний обсяг роботи 95 с. , 37 рис., 11 табл., 15 джерел.

Ключові слова: клієнт-сервер, багатошарова архітектура, SOAP, REST, Spring Framework.

АННОТАЦИЯ

бакалаврской работы Кравчука Евгения Сергеевича
на тему «Разработка многослойной клиент-серверной архитектуры на примере программы для тестирования учеников средних классов»

Тенденции к улучшению качества образования определяются широкой информатизацией данной сферы общественной жизни. Как результат, задание автоматизации контроля знаний учеников с помощью программного обеспечения стаѐт очень актуальной задачей, ведь еѐ решение дает значительную выгоду в экономическом аспекте и значительно повышает продуктивность учебного процесса.

Целью данной работы есть создание программы для тестирования учеников средних классов на основе многослойной клиент-серверной архитектуры. Основной акцент в разработке данного продукта был сделан на проектирование архитектуры приложения и примера клиентской части, который демонстрирует его возможности, понятный для рядового пользователя интерфейс и высокий уровень быстродействия.

Данное приложение разрабатывалось с использованием фреймворков Spring Data JPA, Spring Data REST і AngularJS. Использование выбранных технологий позволило выполнить задание в полной мере и создать основу для удобного расширения функционала.

Общий объѐм работы 95 с., 37 рис., 11 табл., 15 ссылок.

Ключевые слова: клиент-сервер, многослойная архитектура, SOAP, REST, Spring Framework.

ABSTRACT

of a Bachelor's thesis of Yevhenii Kravchuk

"Implementation of multi-layer client server architecture through the example of application for middle school students testing"

Tendencies for improving the quality of education are determined by wide informatization of this life sphere. As a result, the task of automating control of pupils' knowledge using software is on high demand, because it gives benefit from an economic perspective and considerably increases productivity of the studying process.

The objective of the work is to develop an application based on multilayer architecture for middle school students testing. The process is focused on building multi-layer architecture of an application and client side sample that demonstrates its features through intuitive user-friendly interface and high-speed performance.

The product was designed using Spring Data JPA, Spring Data REST and Angular JS frameworks. Use of abovementioned technologies helped to fully complete the task and to develop the base for great extension of functionality.

The total amount of work: 95 p., 37 pic., 11 tables, 15 links.

Keywords: client-server, multi-layer architecture, SOAP, REST, Spring Framework

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

MVC – Model View Controller

REST – REpresentational State Transfer

SOAP – Simple Object Access Protocool

JSON – JavaScript Object Notation

XML – eXtensible Markup Language

HTTP – Hyper Text Transfer Protocol

RMI – Remote Method Invocation

CORBA – Common Object Request Broker Architecture

EE – Enterprise Edition

JPA – Java Persistence API

EJB – Enterprise Java Beans

ПП – програмний продукт

СДН – система дистанційного навчання

СКБД – система контролю баз даних

ВСТУП

Сучасний темп розвитку суспільства надзвичайно високий, це зумовлено потребою людини встигати за змінами, обробляти величезну кількість інформації, яка надходить з усіх точок земної кулі. Тому інформаційні технології мають сьогодні пріоритетне значення в багатьох сферах діяльності й визначають майбутній розвиток суспільства.

Ми живемо в інформатизованому світі, тому кожній школі, інституту, університету та іншим освітнім закладам зараз потрібно забезпечити, крім комп’ютеризації навчального процесу, реалізацію цільової програми інформатизації закладу освіти.

Інформатизація освіти в Україні є одним із пріоритетних напрямів реформування. У широкому розумінні – це комплекс соціально-педагогічних перетворень, пов'язаних з насиченням освітніх систем інформаційною продукцією, засобами й технологіями, у вузькому – впровадження в заклади системи освіти інформаційних засобів, що ґрунтуються на мікропроцесорній техніці, а також інформаційної продукції і педагогічних технологій, які базуються на цих засобах.

Автоматизація контролю знань учнів за допомогою комп’ютерної техніки та програмного забезпечення є економічно вигідним та ефективним засобом управління процесом навчання. Існує багато форм контролю – екзамен, залік, контрольна робота. Але найбільш коректним засобом вимірювання знань на сьогоднішній день є проведення тесту. Тест – це сукупність спеціальним чином підготовлених і підібраних завдань, що дозволяє провести виявлення характеристик процесу навчання. Однією з найголовніших переваг тестів є те, що вони дозволяють провести контроль знань по всім питанням цільового матеріалу в однакових умовах, застосовуючи при цьому до усіх без винятку

одну і ту ж, заздалегідь розроблену шкалу оцінок. Це значно підвищує об’єктивність, чіткість та обґрунтованість оцінки у порівнянні, наприклад, з екзаменом.

В епоху інформаційного суспільства все більш актуальними стають різноманітні комп’ютерні системи тестування, здатні доповнити чи замінити традиційні методи контролю й методики викладання. Завдяки комп’ютерним системам стало набагато зручніше проводити тестування в усіх сферах, де застосовувалися і застосовуються звичайні тести. Наприклад, дистанційна освіта, котра стала поширеним способом поширення знань.

Перехід від «паперового» до комп’ютерного тестування, значно розширює можливості контролю та оцінювання рівня знань учнів, надаючи швидкий і об’єктивний результат, дозволяє більш ефективно використовувати час, відведений на організацію навчального процесу.

Здійснення процесу тестування в комп’ютерному класі з мережею дозволить зекономити час і одночасно протестувати більшу кількість учнів. Тестування може відбуватися і через мережу Internet, що дає можливість учням, що знаходяться в іншій точці планети, проходити тестування і не відставати від шкільної програми. Автоматизоване виставлення оцінок і централізований збір результатів та побудови статистик на порядок збільшать ефективність роботи викладача.

Додаток для тестування дозволить працювати з питаннями в електронному вигляді, використовувати всі види цифрової інформації для відображення змісту тесту. Електронна форма завдань тестів дає більшу гнучкість в їх модифікації.

Система для тестування учнів середніх класів буде забезпечувати можливість створення тестів викладачами кожного предмету, створення тестових питань різних типів, перегляд статистики успішності учнів та

можливість виставлення оцінок учням, доступ до яких матимуть батьки.

Інтерфейс буде представлений Web-клієнтом з підказками та поясненням призначення основних елементів інтерфейсу користувача. Безпека додатку буде реалізована шляхом авторизації та аутентифіцаії користувачів, виділення різних ролей користувачів (вчитель, учень, адміністратор системи, батьки). Система матиме клієн-серверну архітектуру у загальному вигляді. Як клієнт, так і сервер будуть побудовані у вигляді багатошарової архітектури з використанням архітектурного патерна MVC з представленням усіх даних додатку у вигляді SQL-бази даних на сервері та Angular.js на стороні Web-клієнта.

Отже, темою дипломної роботи є реалізація багатошарової клієнт-серверної архітектури на прикладі програми для тестування учнів середніх класів з базою даних, що міститиме тести, тестові питання, дані користувачів; шаром логіки та формування статистики. Демонстрація роботи системи буде проведена через користувацький веб-інетрфейс.

1. ОГЛЯД ІСНУЮЧИХ АРХІТЕКТУР

1.1 Визначення

Архітектура інформаційної системи – концепція, що визначає модель, структуру, виконувані функції і взаємозв'язок компонентів інформаційної системи.

Клієнт-сервер – обчислювальна чи мережева архітектура, в якій завдання чи мережеве навантаження розподілені між постачальниками послуг (сервісів), котрі називають серверами, і замовниками послуг, котрі називають клієнтами.

Сервер – це програма, що надає деякі послуги іншим програмам і обслуговує запити клієнтів на отримання ресурсів певного виду.

Клієнт – це програма, що використовує послугу, надану програмою сервера.

Тобто клієнт і сервер – це просто ролі, виконувані програмами. Один і той же додаток може бути як клієнтом, так і сервером одночасно. Фізично клієнт і сервер – це програмне забезпечення. Взаємодія, зазвичай, відбувається через комп'ютерну мережу посередництвом мережевих протоколів, але іноді клієнт та сервер розміщуються на одному комп'ютері.

1.2 Опис моделі клієнт-сервер

Обчислювальна модель клієнт-сервер зайняла домінуюче місце серед методів розподілених обчислень. Модель передбачає розділення додатку на окремі частини, що розміщуються на різних платформах для більшої ефективності. Як правило, це значить, що програма представлення даних знаходиться на пристрої користувача, а програма управління даними

знаходиться на сервері, схематичне зображення цього показано на рисунку 1.1. Таке розподілення дозволяє позбавитись недоліків персональних комп’ютерів, як-то невелика обчислювальна потужність та надійність, а також ізольованість окремих персональних комп’ютерів один від одного.

Як правило, користувачам персональних комп’ютерів потрібні і висока обчислювальна потужність і широкі можливості та корисні властивості персональних комп’ютерів. В залежності від додатку та використаного програмного забезпечення обробка даних може бути розподілена по-різному. Серверне програмне забезпечення приймає запити від клієнтського програмного забезпечення і повертає йому результати.

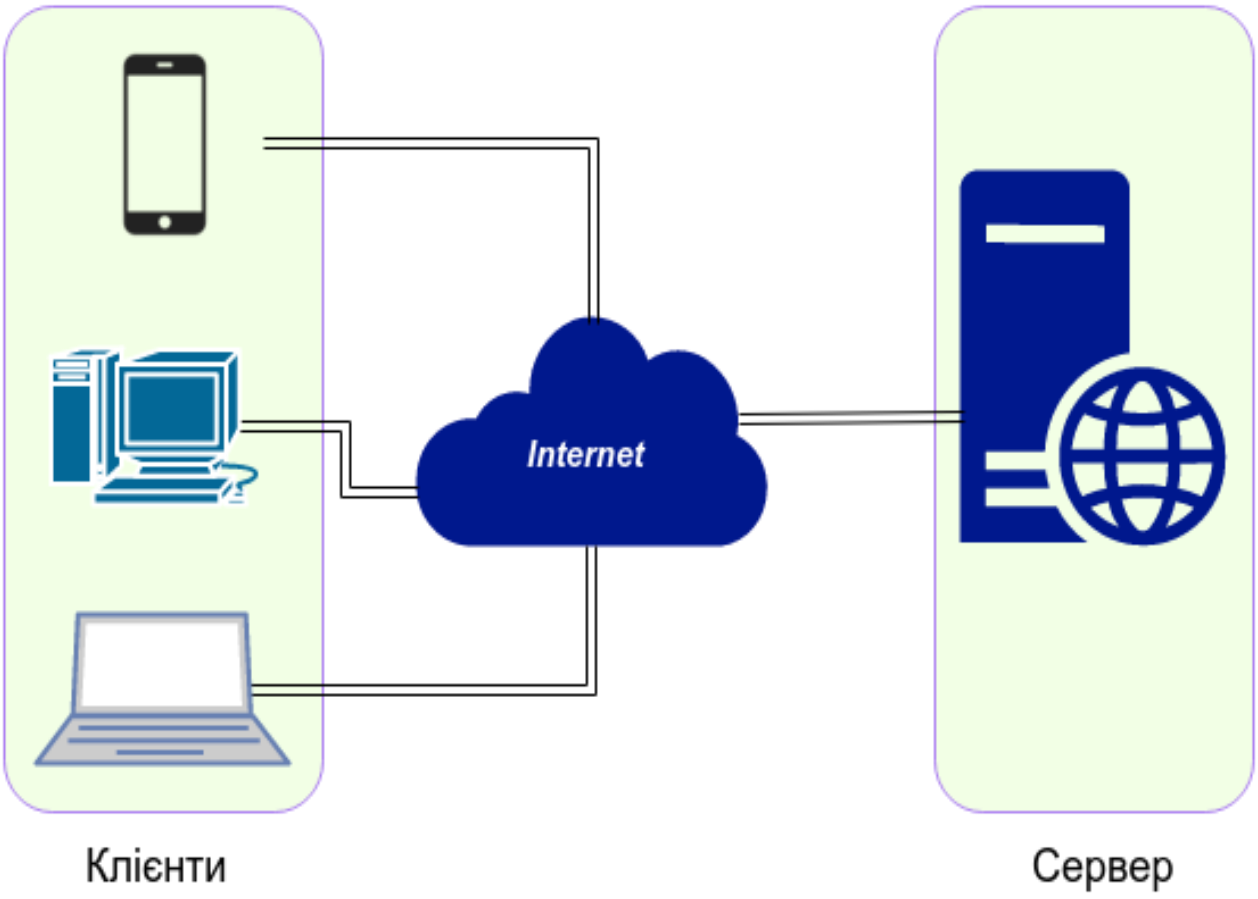

Рисунок 1.1 – Архітектура клієнт-сервер

1.3 Класи клієнт-серверних додатків

В рамках загальної структури додатків типу клієнт-сервер, виконувана робота може бути розділена між клієнтом і сервером по-різному. Точна частка виконуваних операцій і обсяг переданих по мережі даних залежать від природи інформації, що міститься в базі даних, підтримуваних типів додатків, доступності обладнання, яке може працювати спільно, а також від характеру використання даних.

Схеми деяких основних класів додатків показані на рисунках 1.2, 1.3, 1.4, 1.5. Можливі також інші варіанти розподілу задач між сервером і клієнтом.

На рисунках зображено чотири класи додатків з різними варіантами розподілу задач між сервером і клієнтом.

1.3.1 Обробка даних на базі хоста

Дана схема не є справжнім додатком клієнт-сервер, а відноситься до традиційного оточення мейнфрейма, коли вся або майже вся обробка даних здійснюється на головній обчислювальної машині. Найчастіше в подібному обчислювальному середовищі інтерфейс користувача має вигляд примітивного терміналу.

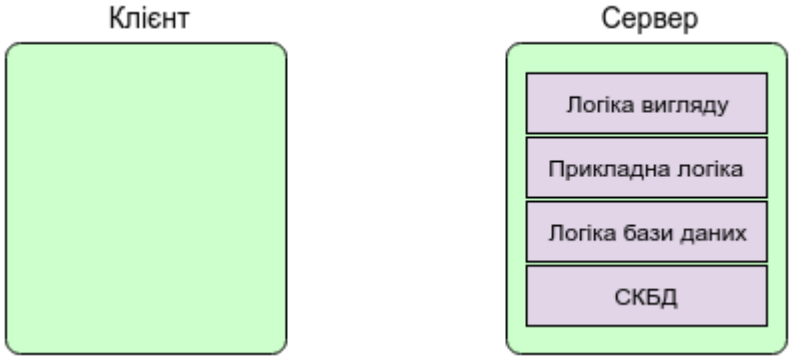

Рисунок 1.2 – Обробка даних на базі хоста

Але навіть якщо користувач використовує персональний комп'ютер, його роль випадку обробки даних на базі хоста обмежується емуляцією терміналу. Така схема є доцільною, коли основна обчислювальна потужність концентрується на стороні сервера. Як правило, клієнти в цьому випадку мають потужність значно меншу, ніж потрібна для виконання задач, для яких створений сервер.

1.3.2 Обробка даних на базі сервера

Найпростішим класом конфігурації клієнт-сервер є схема, в якій клієнт відповідає лише за надання графічного інтерфейсу користувача, тоді як практично вся обробка даних здійснюється на сервері. Це дозволяє знизити навантаження на серверну частину, особливо у випадках, коли логіка вигляду складна і потребує багато системних ресурсів.

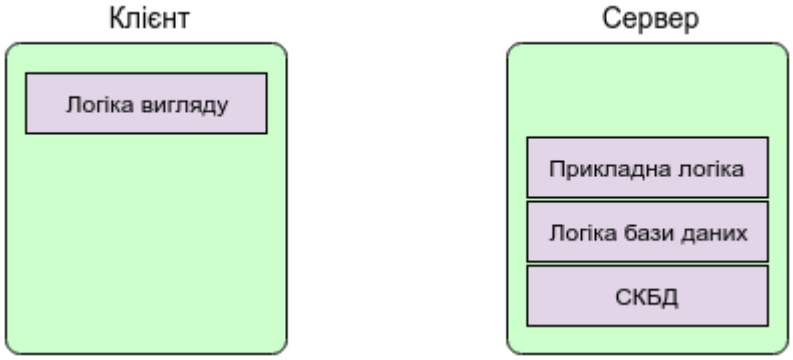

Рисунок 1.3 – Обробка даних на базі сервера

1.3.3 Обробка даних на базі клієнта

Дана схема демонструє зворотній підхід: практично вся обробка даних здійснюється на стороні клієнта, за винятком процедур перевірки цілісності даних та іншої логіки, що відноситься до обслуговування бази даних, які краще виконувати на сервері. Як правило, складні функції для роботи з базою даних

розташовуються на стороні клієнта. На сьогоднішній день реалізації даної схеми є найбільш поширеними реалізаціями архітектури клієнт-сервер. Вона дозволяє користувачеві працювати з додатками, що відповідають його локальним потребам.

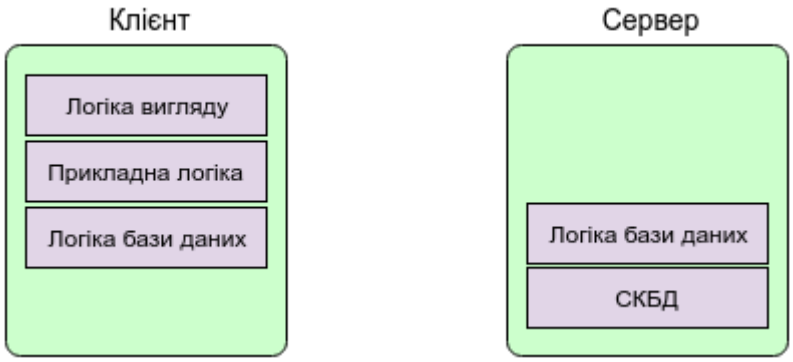

Рисунок 1.4 – Обробка даних на базі клієнта

1.3.4 Спільна обробка даних

У даній конфігурації обробка даних оптимізована таким чином, щоб використовувати сильні сторони як клієнта, так і сервера, а також самого факту розподілу даних.

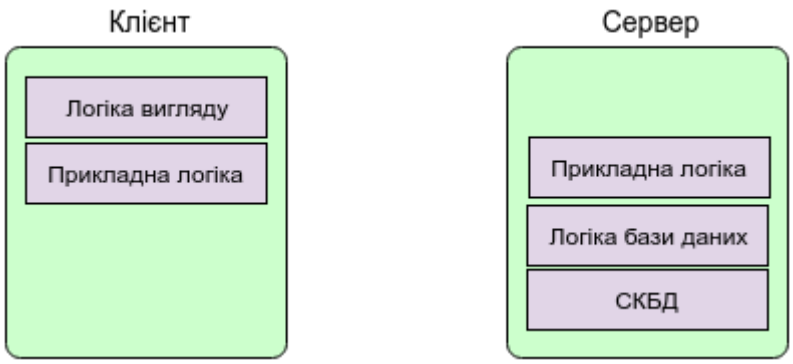

Рисунок 1.5 – Спільна обробка даних

Такі зміни набагато складніші в установці і обслуговуванні, але в довгостроковій перспективі вони дозволяють забезпечити кращі показники продуктивності та ефективності використання мережевих ресурсів, ніж інші методи реалізації архітектури клієнт-сервер.

Рисунок 1.4 та рисунок 1.5 відповідають конфігураціям, в котрих суттєве навантаження покладено на сторону клієнта, такі типи клієнтів називають "товстими". *Товстий або Rich-клієнт* в архітектурі клієнт-сервер – це додаток, що забезпечує розширену функціональність незалежно від центрального сервера. Часто сервер в цьому випадку є лише сховищем даних, а вся робота по обробці і представленню даних переноситься на машину клієнта.

1.4 Архітектура серверної частини

1.4.1 Архітектура, як поняття

Індустрії програмного забезпечення притаманна властивість, що будь-який термін може мати багато суперечливих трактувань. Ймовірно, найбільш абстрактним виявилось поняття *архітектура*. Зрештою, можна назвати два загальних варіанта. Перший пов'язаний з розбиттям системи на найбільш значимі складові частини; в другому випадку маються на увазі деякі конструктивні рішення, котрі після їх прийняття важко піддаються внесенню змін. Також, росте розуміння того, що існує більше одного способу описання архітектури і ступінь важливості кожного з них змінюється з плином життєвого циклу системи.

Ральф Джонсон, науковий співробітник, професор кафедри комп'ютерних наук в Університеті штату Іллінойс в Урбана-Шампейн, один з чотирьох авторів класичної книги «Design Patterns» про шаблони проектування програмного забезпечення, підтвердив думку про те, що архітектура – цілком суб'єктивне поняття, котре, в кращому випадку, відображає спільну точку зору

команди на результати проектування системи. Зазвичай це згода у питанні ідентифікації головних компонентів системи та способів їх взаємодії, а також вибір таких рішень, котрі інтерпретуються як основоположні і не підлягають змінам у майбутньому.

В даному розділі представлено структуру системи після декомпозиції у вигляді архітектурних *шарів (layers)*. В більшості корпоративних додатків відслідковується та чи інша форма архітектурного «розшарування», але в деяких ситуаціях більшого значення можуть набувати інші підходи, пов'язані, наприклад з організацією *каналів (pipes)* або *фільтрів (filters)*. Однак увага у даному розділі концентрується на архітектурі шарів як на найбільш плідній структурній моделі.

1.4.2 Розбиття на шари

Концепція шарів – одна з загально використовуваних моделей, використовуваних розробниками програмного забезпечення для розділення складних систем на більш прості частини. Рисунок 1.6 демонструє приклад розшарованої системи.

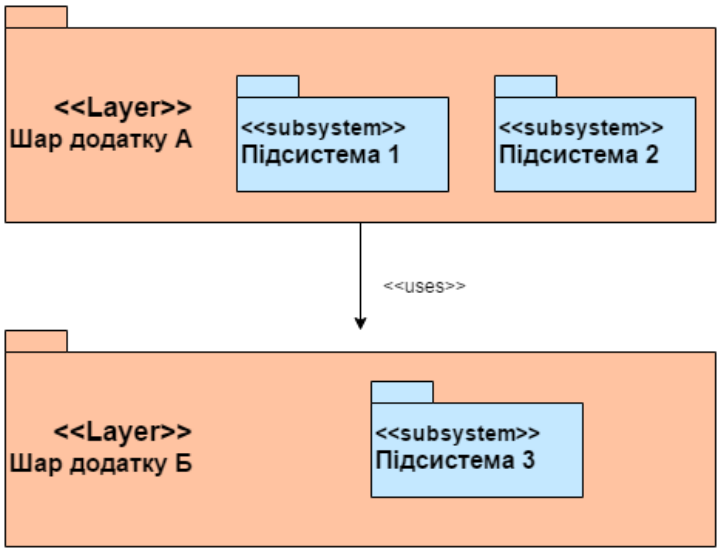

Рисунок 1.6 – Розділення додатку на шари

В архітектурах комп’ютерних систем, наприклад, розрізняють шари коду на мовах програмування, функцій операційної системи, драйверів пристроїв, наборів інструкцій центрального процесора і внутрішньої логіки чипів. Описуючи систему в термінах архітектурних шарів, зручно сприймати підсистеми, з яких вона складається у вигляді «багатошарового пирога». Шар більш високого рівня користується послугами, що надає нижній шар, але той не знає про існування сусіднього верхнього рівня. Більше того, зазвичай кожен проміжний шар приховує нижній шар від верхнього. Не в кожній архітектурі шари настільки «непроникні», але в більшості випадків це так.

Розділення системи на шари надає цілий ряд переваг:

- окремий шар можна сприймати як єдине самодостатнє ціле, не піклуючись про наявність інших шарів;
- можна обрати альтернативну реалізацію базових шарів;
- залежність між шарами зводиться до мінімуму;
- кожен шар є кандидатом на стандартизацію;
- якісно створений шар може слугувати основою для декількох різних шарів більш високого рівня.

Схемі розшарування властиві певні недоліки: шари здатні вдало інкапсулювати багато, але не все; модифікація одного шару одночасно пов’язана з потребою внесення *каскадних змін* в інші шари.

Класичний приклад з області корпоративних додатків: поле, додане в таблицю бази даних, підлягає відображенню його в графічному інтерфейсі і повинне залишити відбиток у кожному проміжному шарі. Другим недоліком є те, що наявність додаткових шарів знижує продуктивність системи.

При переході від шару до шару сутності зазвичай піддаються трансформації з одного представлення в інше. Не зважаючи на це, інкапсуляція

нижче розташованих шарів дозволяє досягнути істотних переваг. Наприклад, оптимізація шару транзакцій зазвичай приводить до підвищення продуктивності всіх шарів, що розташовані вище.

Поняття шару набуло очевидної значущості в середині 1990-х років з появою архітектури клієнт-сервер. Це були системи з двома шарами, клієнт відповідав за роботу інтерфейсу користувача і виконання коду додатка, а роль сервера виконувала СКБД.

1.4.3 Двошарова архітектура

Двошарова архітектура програмного забезпечення зазвичай відповідає моделі “товстого” клієнта. В такій моделі серверні компоненти системи відповідають, головним чином, за організацію зберігання і доступу до даних, а всі або більшість функцій прикладної обробки даних виконуються на стороні клієнтської частини.

Схема роботи дворівневої архітектури продемонстрована на рисунку 1.7. Тут шари прикладних рішень і засоби підтримки виконання програм прикладного шару, що входять в системний шар функціонують на робочій станції, а засоби організації зберігання і доступу до даних - здебільшого на сервері.

Переваги та недоліки даного підходу зведені у таблицю 1.1. Засобом організації зберігання і доступу до даних зазвичай виступає СКБД з підтримкою великої кількості користувачів.

Дана архітектура довгий час була пануючою, але з плином часу та ростом потреб індустрії проявлялися її недоліки, котрі призвели до появи нових архітектур, як, наприклад, тришарова, а пізніше багатошарова архітектура програмних додатків.

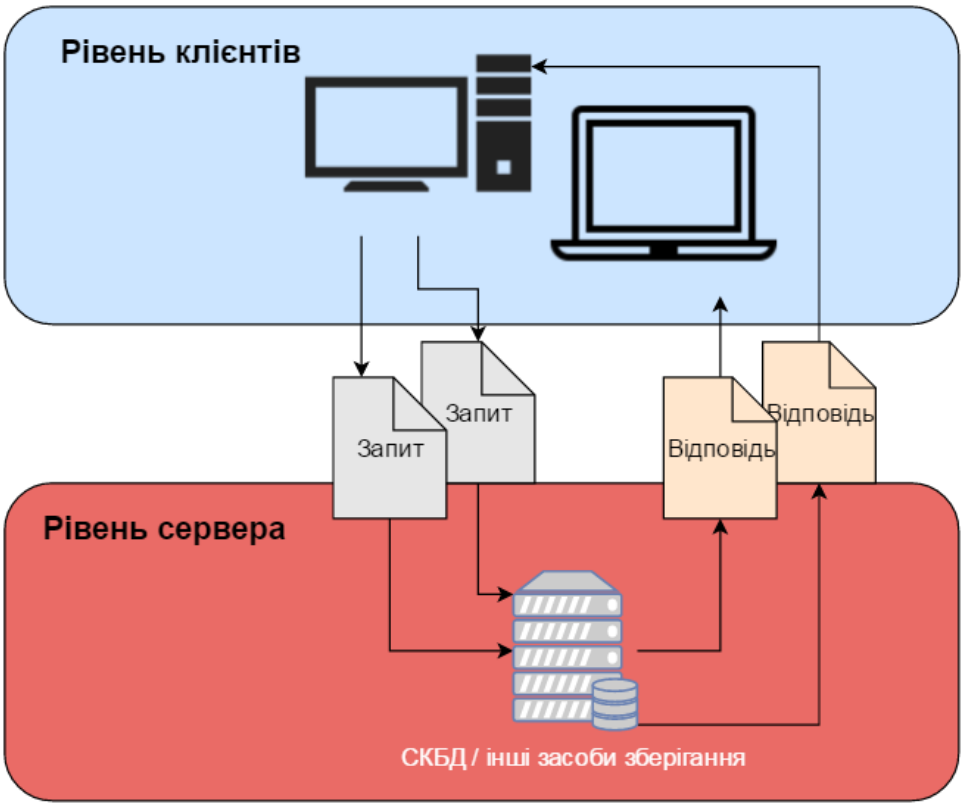

Рисунок 1.7 – Двошарова архітектура

Таблиця 1.1 – Переваги та недоліки двошарової архітектури

Переваги	***Недоліки***
Простота системи, у порівнянні з тришаровою і багатошаровою архітектурами	Необхідність більш потужного комп'ютера в якості сервера та потужних клієнтських машин, здатних забезпечити і бізнес логіку і графічний інтерфейс
Гарантія цілісності даних	Відсутність масштабування. Слабкий захист від взлому
Повна підтримка одночасної роботи багатьох користувачів	Бізнес-логіка повністю на стороні клієнта. При її зміні треба повністю оновлювати клієнтське ПЗ

1.4.4 Трьохшарова архітектура

На сьогоднішній день, усі бізнес-додатки мають можливість доступу до даних, як частину їх базового функціоналу. В той час як реляційні сервери баз даних набули популярності близько 25 років тому, промисловість перейшла від моделі один шар, мейнфреймів, до моделі клієнт-сервер, де є клієнт, який виконує логіку вигляду і більшу частину бізнес-логіки, та сервер зі зберіганням даних і бізнес-логіки в формі збережених запитів.

До початку 1990-х років ця модель зламалася через високі витрати на технічне обслуговування і відсутність поділу проблем і ми перейшли до архітектури трьохшарової, принципова схема роботи якої показана на рисунку 1.8. Детальний опис шарів наведено у таблиці 1.2.

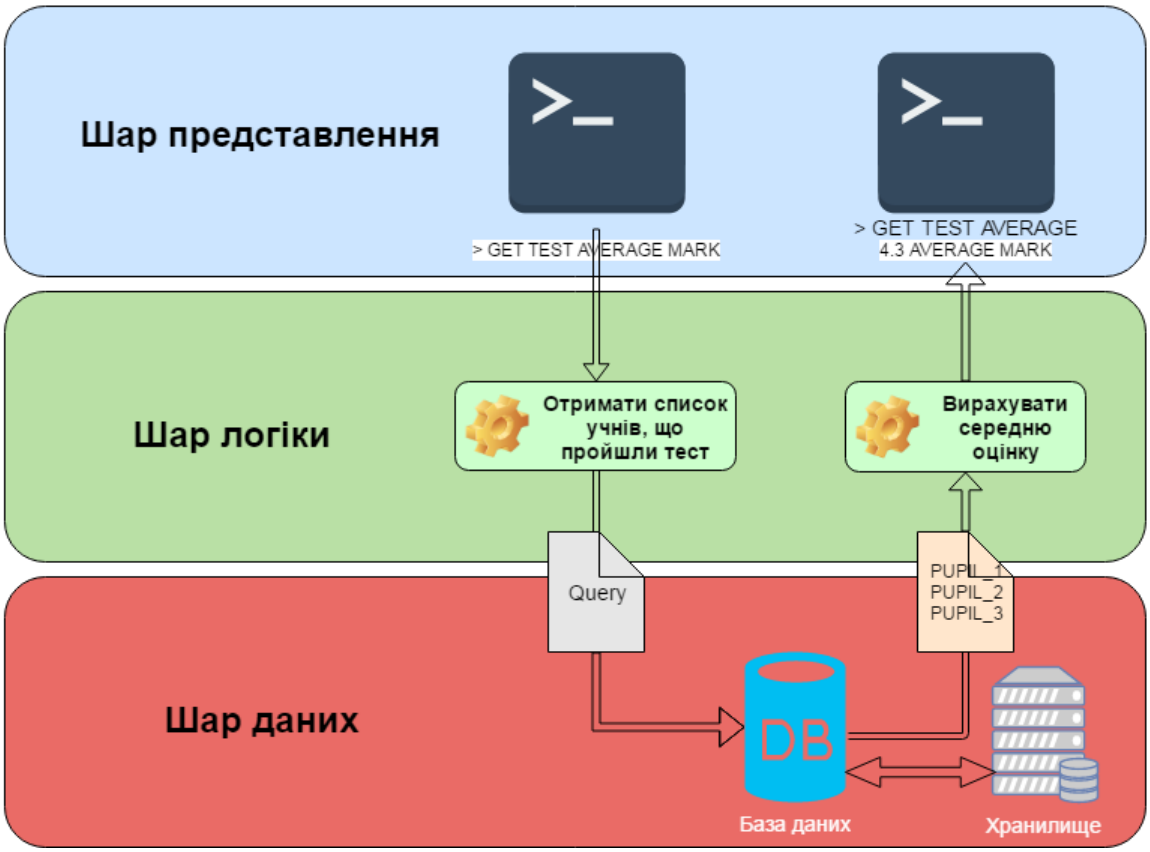

Рисунок 1.8 – Трьохшарова архітектура

Основними шарами даної архітектури є *шар представлення*; *домен,* котрий ще називають шаром бізнес-логіки та *шар даних*, який узагальнює джерела даних. Кожен з них має визначені функції та несе відповідальність за частину роботи, виконувану додатком, шари можуть розміщуватися не тільки локально на одному пристрої, а і бути розділеними, наприклад представлення на клієнтській частині, а бізнес-логіка і джерело даних – на серверній частині додатку.

Таблиця 1.2 – Функції шарів трьохшарової архітектури

Шар	***Функції***
Представлення	Надання послуг, відображення даних, обробка подій користувацького інтерфейсу, обслуговування HTTP-запитів, підтримка функцій командної строки та API пакетного використання.
Домен	Бізнес-логіка додатку, специфічні алгоритми
Джерело даних	Запити до бази даних, обмін повідомленнями, управління транзакціями тощо.

Шару *представлення* стосується усе, що пов’язане зі взаємодією користувача з системою. Він може бути простим, як командна строка чи текстове меню, але зараз користувачу, ймовірніше за все, доведеться мати справу з графічним інтерфейсом, оформленим у стилі «товстого» клієнта, як-то популярні технології для створення графічного користувацького інтерфейсу на кшталт Windows, Swing, JavaFX, або бути заснованим на мові розмітки HTML.

Головна задача шару представлення – транслювати команди користувача у формат, зрозумілий шару бізнес-логіки.

Логіка домену – описує основні функції додатку, призначені для досягнення поставленої перед ним цілі. До цих функцій належать обчислення на основі введених і збережених процедур, перевірка усіх елементів даних і обробка команд, що надходять від шару представлення, а також передача інформації шару джерела даних.

Іноді шари організовують таким чином, щоб бізнес-логіка повністю приховувала джерело даних від представлення. Однак частіше код представлення може звертатися до джерела даних безпосередньо.

Хоча такий варіант менш бездоганний з теоретичної точки зору, в практичному використанні він нерідко більш зручний та доцільний; код представлення може інтерпретувати команду користувача, активізувати функції джерела даних для отримання відповідних порцій інформації з бази даних, звернутися до засобів бізнес-логіки для аналізу цієї інформації і виконання необхідних розрахунків і тільки після цього відобразити відповідну картину на екрані.

Джерело даних – це підмножина функцій, що забезпечують взаємодію зі сторонніми системами, котрі виконують завдання в інтересах додатку. Код цієї категорії несе відповідальність за моніторинг транзакцій, управління іншими додатками, обмін повідомленнями тощо. Для більшості корпоративних додатків основна частина логіки джерела даних концентрується в коді СКБД.

1.4.5 Багаторівнева архітектура

У кінці 1990-х років, індустрія розширила поняття трирівневої архітектури до багаторівневої. Логічно модель має таку ж саму структуру, але всеохоплююче використання Інтернету внесло свої корективи, ставши важливою частиною багатьох програмних додатків.

Веб-сервіси (а пізніше REST дані) стали більш інтегровані в додатки. Як наслідок, шар даних, як правило, стали розщеплювати на рівень зберігання даних (сервер баз даних) і рівень доступу до даних (DAL, data access layer). У комплексних системах для уніфікації доступу до баз даних і веб-сервісів розробляють додатковий рівень класів-обгорток.

Веб-браузери були менш потужним, ніж традиційні додатки клієнтського рівня і логіка користувацького інтерфейсу розділилися між браузером з JavaScript і сервером з додатком веб-сервера, що містить у собі логіку користувацького інтерфейсу.

Шари все далі і далі набували більш розмитого характеру із додаванням збережуваних процедур усіма основними постачальниками баз даних і баз даних з відкритим вихідним кодом. Це призвело до поширення практики переносу деяких частин бізнес-логіки від бізнес-рівня на рівень бази даних, тобто з’явилася концепція створення *рівнів в межах рівнів*.

Так як під впливом Інтернету, технологічних інновацій і сервісів архітектура додатку стала більш розмитою, трьохшарова модель додатку розвинулася у багаторівневу архітектуру, концептуальна схема якої показана на рисунку 1.9.

Як згадувалося вище, вона з’явилася завдяки створенню декількох рівнів в інших рівнях, а саме в рівні *представлення*:

- компоненти графічного інтерфейсу, котрі відповідають за відображення графічних елементів;
- компоненти процесів графічного інтерфейсу, котрі реагують на події, що відбуваються у графічному інтерфейсі.

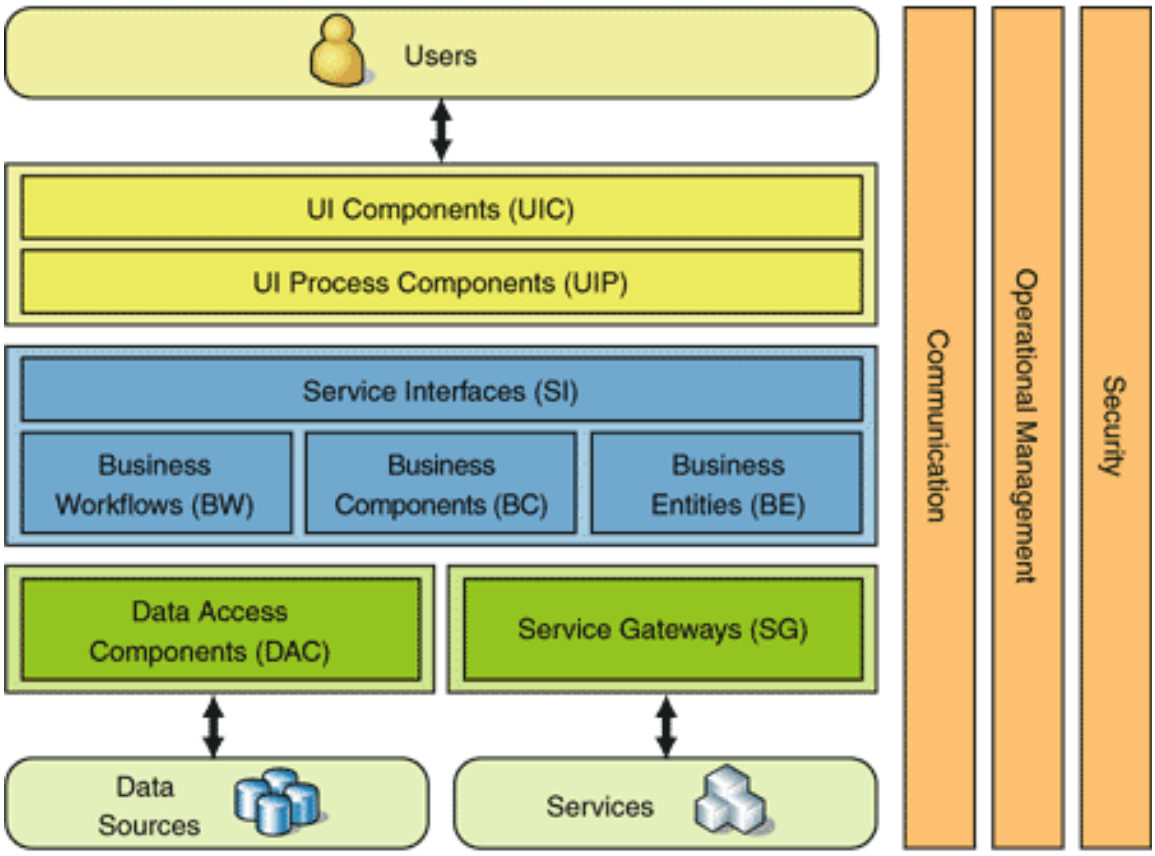

Рисунок 1.9 – Приклад схеми багатошарової архітектури

Великі корпоративні додатки часто структуровані навколо бізнес-процесів та бізнес-компонентів. Ці поняття розглядаються в рамках цілого ряду компонентів, сутностей, агентів та інтерфейсів бізнес-рівня:

- *бізнес-компоненти* – програмні реалізації концепцій чи процесів. Вони складаються з усіх артефактів необхідних для представлення, реалізації, розгортування конкретної концепції як автономного елемента більшої системи, котрий можна використовувати повторно.
- *бізнес-сутності* – це структури, що виступають контейнерами даних. Вони інкапсулюють та приховують деталі специфічного формату представлення даних. Наприклад, бізнес сутність може

інкапсулювати набір записів, отриманих з бази даних. Пізніше, ця ж бізнес-сутність може бути змінена для огортання в XML-документ з мінімальним впливом на інші частини додатку.

- *сервісні інтерфейси* – додаток може надавати частину його функціоналу як сервіс, котрий можуть використовувати інші додатки. В ідеалі він приховує деталі реалізації і надає тільки тонкий шар інтерфейсу.
- *бізнес-процеси* – відображають діяльність бізнесу на високих рівні абстракції системи, як-то обробка замовлення, підтримка користувача, закупка товару.

Шар даних теж зазнав певних внутрішніх метаморфоз, внаслідок чого з’явилися наступні шари:

- *компоненти доступу до даних* – ізолюють бізнес-шар від деталей реалізації, специфічних для сховища даних. Дозволяє мінімізувати вплив зміни постачальника бази даних, зміни представлення даних, наприклад, схеми бази даних, інкапсулює весь код, що маніпулює конкретною одиницею даних в одному місці, що надзвичайно спрощує підтримку та тестування.
- *сервісні шлюзи* – бізнес-компоненти часто повинні отримувати доступ до внутрішніх та зовнішніх сервісів чи додатків. Сервісний шлюз – це компонент, що інкапсулює інтерфейс, протокол та код, потрібний для використання сервісів. Наприклад, бізнес-рішення часто потребує інформацію з деякого сервісу для завершення бізнес-процесу. Воно делегуватиме всю взаємодію з цим сервісом шлюзу. Сервісний шлюз надає можливість з меншими зусиллями змінити зовнішній сервіс на інший. Також даний підхід надає змогу емулювати зовнішній сервіс, наприклад, для тестування доменного рівня.

Надодачу до описаних шарів багатошарова архітектура визначає набір *фундаментальних* сервісів, котрі потенційно можуть використовувати усі інші шари. Ці сервіси діляться на три базові категорії:

- *шар безпеки* – сервіси цього шару підтримують безпеку додатку;
- *шар операційного управління* – ці сервіси оперують компонентами і зв'язаними з ними ресурсами і також торкаються таких вимог як масштабованість та відмовостійкість;
- *шар сервісів комунікації* – сервіси, котрі надають можливість спілкуватися різним шарам між собою.

Переваги даної архітектури – гарна точка відправлення для побудови власних додатків. Розробнику, що використовує даний підхід, дістаються найбільші позитивні риси розшарованого додатку. Але є й певні аспекти архітектури, які додають відповідальності, а саме, для важких, комплексних рішень необхідно правильно розділяти доменний рівень, особливо, якщо можливість повторного використання компонентів є в пріоритеті або якщо розробник проектує сімейство рішень, що базується на наборі компонентів. У такому випадку типовим є заміна одного бізнес-шару класичного тришарового додатку трьома.

1.5 Моделі взаємодії клієнта і сервера

Метою цього розділу є порівняння характеристик продуктивності Simple Object Access Protocol (SOAP), XML-RPC та Representational State Transfer (REST), котрі являються методами підтримки взаємодії веб-сервісів. Вони відрізняються як контекстом, так і використанням: SOAP – це протокол, в той час як REST – це архітектурний стиль. SOAP є добре продуманим протоколом, що використовується в веб індустрії і стандартизується World Wide Web Consortium-ом (W3C). REST в свою чергу – це резульат дисертації

“Architectural Styles and the Design of Network-Based Software Architecture” 2000-го року, автором якої є Рой Філдинг.

REST набирає популярності через свою простоту, масштабованість і архітектурну залежноість від World Wide Web. Великі корпорації, як-то Google і Amazon, зокрема, використовують REST у переважній більшості своїх продуктів. Найголовнішою відмінністю між цими моделями є те, що SOAP є сильнозв’язаною системою (tightly coupled system), водночас REST – слабозв’язана система; обидві мають переваги і недоліки.

1.5.1 Веб-сервіси

Веб-сервіси – категорія веб-рішень, котрі використовуються в таких сферах, як банкінг, онлайн-покупки і соціальні мережі. Веб-сервіси головним чином складаються з сервера та клієнта. Сервер надає визначені сервіси клієнтам через всесвітню павутину. Клієнти можуть бути багатьох типів і, в залежності від того, як сервіс спроектовано, мають мінімальну кількість коду чи взагалі не містять його.

Іноді термін “веб-сервіс” плутають з поняттям “веб-сайту”. Одна важлива відмінність між ними полягає в тому, що веб-сервіси не надають ніякого GUI користувачу, на відміну від веб-сайтів.

Веб-сервіси і веб-сайти цілковито різні речі. Веб-сайт являється множиною веб-сторінок, які, в свою чергу, можуть включати в себе декілька веб-сервісів. Наприклад, одна сторінка може бути використана для прослуховування музики, здійснення покупок, перегляду прогнозу погоди, тобто функціями, яким відповідають веб-сервіси, вбудовані в дану веб-сторінку.

Ще однією функцією веб-сервісів є з’єднання програм на різних вузлах в мережі Інтернет одна з одною, пересилаючи великі об’єми даних більш ефективно і незатратно, ніж було до їх появи.

1.5.2 Обмін даними на основі XML

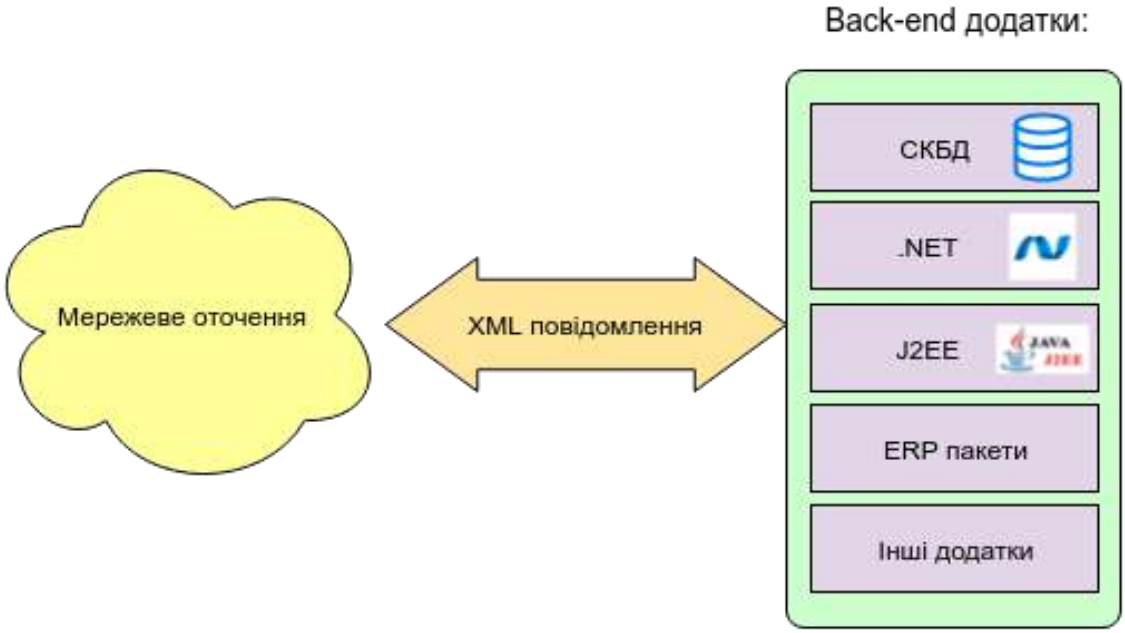

Рисунок 1.10 – взаємодія веб-сервісу з back-end додатками

Extensible Markup Language (XML) – це найбільш типовий формат обміну даними між веб-сервісами. Додатки веб-сервісів можуть узгоджувати серії обміну повідомленнями, використовуючи стандартизовані XML документи. Таким чином, веб-сервіс також може бути описаний набором стандартів на основі XML, що роблять можливим електронне спілкування і взаємодію незалежно від платформи чи специфічних технологій, що використовуються взаємодіючими сторонами.

Як видно на рисунку 1.10, веб-сервіси включають логіку і дані і забезпечують стандартний спосіб взаємодії між системами. Вони представляють таким програмам, як СКБД, і платформам на кшталт .NET і J2EE стандартний шлях взаємодіяти з мережею. Інтерфейс веб-сервісу отримує XML-повідомлення з мережі і транслює повідомлення у формат, зрозумілий взаємодіючим системам. Реалізація підходу трансляції XML-повідомлення може бути створена з використанням будь-якої мови програмування.

Розподілені об’єктні технології такі як Corba, Orbacus, Java RMI були розроблені для задоволення потреб розподілених обчислень, але всім не вистачало здатності взаємодіяти одна з одною. Для усунення цього недоліку взаємодії, W3C розробив Simple Object Access Protocol (SOAP), котрий використовував галузеві стандарти такі як XML та HTTP (Hyper Text Transfer Protocol) для полегшення обміну між програмними додатками, незалежно від платформ, для яких вони розроблені. SOAP – це технологічна специфікація, розроблена для спрощення спілкування веб-сервісів з іншими системами.

Інший конкуруючий підхід, що підтримує розробку інтерфейсів веб-сервісів є Representational State Transfer (REST). REST – це набір архітектурних принципів для розробки веб-додатків. Принципи REST фокусуються на способах адресації станів ресурсу і передачі ресурсів через HTTP.

Не дивлячись на те, що як REST, так і SOAP є методами конструювання веб-сервісів, вони відрізняються механізмами обробки даних та надання сервісу. SOAP – це протокол, в той час як REST – архітектура, відтак, не є доцільним порівнювати ці дві технології, до того ж це є поширеною помилкою. Але можна провести нейтральну оцінку продуктивності і можливостей, запропонованих розробникам і архітекторам, що використовують SOAP- і REST- методології під час розробки веб-сервісів.

1.5.3 SOAP

Це протокол для обміну веб-сервісами структурованою інформацією в комп’ютерних мережах. Мова для опису веб-сервісів WSDL (Web Service Description Language) використовується разом з SOAP, щоб зробити повідомлення доступними у Web посередництвом веб-сервісів. Загалом, SOAP сумісно з WSDL називається SOAP Web services.

Без протокола SOAP веб-сервіси не можуть працювати кросплатформенно, і зазвичай несумісні. SOAP був розроблений для

заповнення цієї прогалини, полегшуючи розробнику конструювання веб-програми незалежно від вищезгаданих обмежень. SOAP в данний момент залежить від HTTP, що лежить в основі, але може використовувати інші протоколи передачі даних. Microsoft були першими, хто почав розроблювати SOAP, а далі й інші відомі корпорації почали вносити свій вклад у розробку.

Раніше в історії персональних комп'ютерів, базова обробка даних складалася з одного персонального комп'ютера з використанням простих додатків, що залежали від локальних ресурсів. Розробки у сфері комп'ютерних мереж поступово популяризували концепції комп'ютерних мереж LAN, MAN та WAN, даючи можливість спілкуватися комп'ютерам, що фізично знаходяться у різних місцях.

Протоколи веб-сервісів дали можливість комп'ютерам взаємодіяти на однакових платформах, технологіях та операційних системах. Потреба у протоколі, незалежному від цих обмежень породила протокол SOAP, котрий використовує XML для спілкування поверх транспортного рівня.

Призначення SOAP-а – надати мінімальний транспортний функціонал, поверх якого можуть бути побудовані більш складні протоколи та процеси взаємодії. Механізм взаємодії схематично продеманстровано на рисунку 1.11.

Для розробки клієнта треба знати URL кінцевої точки сервера, яка відповідає за обробку повідомлень. Це стає ще легшим з використанням WSDL, форматом XML схеми, який надає можливість представити мережеві сервіси у вигляді множини кінцевих точок, що обробляють повідомлення.

Як показано на рисунку, програми і додатки на сайті 1, що працює на платформі Java, прив'язані до WSDL і представлені у Web-і як веб-сервіси з використанням SOAP.

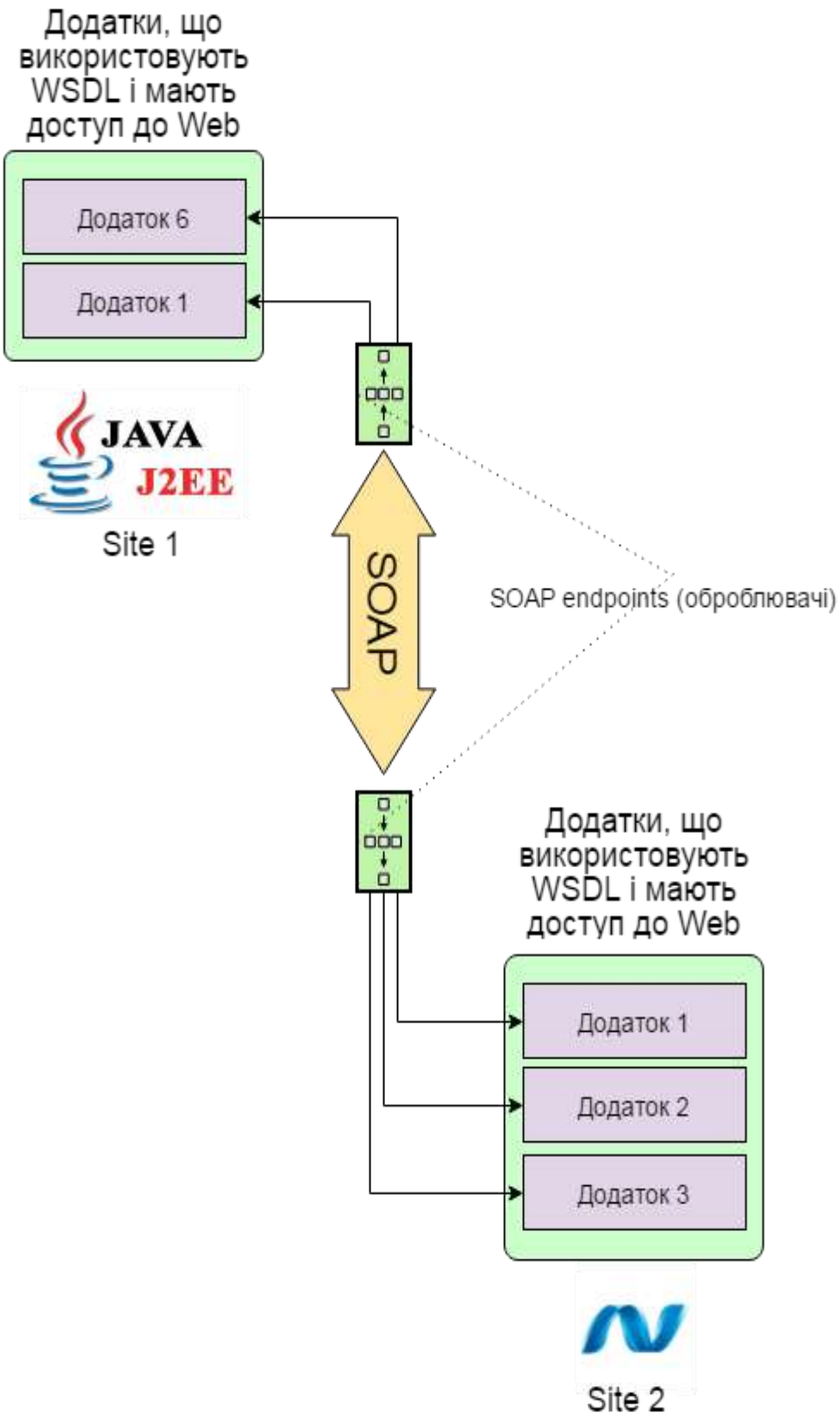

Рисунок 1.11 – Механізм SOAP

Сайт 1 зв’язаний з сайтом 2, котрий працює на платформі .Net, з використанням протоколу SOAP. Оброблювачі перетворюють повідомлення і конвертують їх у вигляд, що прийнятний для використання на іншій платформі.

Аналізуючи вищеописаний приклад, можна сказати, що весь цей процес має клієнт-серверну модель типу запит-відповідь. Офіційно протокол був стандартизований консорціумом W3C у 2003 році, використовується для представлення бізнес даних через інтерфейси і сервіси, що обмінюються цілими документами чи відображають дані на об’єкт, використовуючи назви методів та вхідні й вихідні параметри. Наступний уривок коду є прикладом типового SOAP повідомлення:

```
        POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: 299
<?xml version=”1.0”>

<soap:Envelope xmlns:soap="http://w3.org/2003/05/soap-envelope">
        <soap:Header>
        </soap:Header>
                <soap:Body>
                        <m:GetStockPrice xmlns:m=”http://www.example.org/stock”>
                                <m:StockName>IBM</m:StockName>
                        </m:GetStockPrice>
                </soap:Body>
</soap:Envelope>
```

Використані елементи маєть наступні значення:

- *Envelope* – є індикатором початку та кінця повідомлення;
- *Header* – містить опціональні допоміжні дані для обробки повідомлень;
- *Body* – складається з головних XML даних;

Щоб бути незалежними від HTTP, SOAP має перевагу, а саме, елемент *attachment* - надсилання прикріплених файлів, що часто вважають найкращим рішенням для сервісів, що мають справу з прикріпленими файлами.

Через свою надійність та рівень безпеки протоколи SOAP широко використовуються у банківських системах.

1.5.4 REST

REST – це архітектура для розробки веб-сервісів, що намагається імітувати архітектури, котрі використовують HTTP чи схожі протоколи, шляхом обмеження інтерфейсів набором стандартних і добре знайомих розробникам операцій (наприклад, GET, PUT/PATCH, POST та DELETE для HTTP).

Акцент зроблений на взаємодії з ресурсами, що мають збережений стан, а не з повідомленнями та операціями. Можна сказати, що ця архітектура розроблена для того, щоб показати, що існуючого HTTP достатньо для побудови веб-сервісу, та демонстрації маштабування.

Рой Філдинг, один із основних авторів HTTP, визначив у своїй докторській дисертації набір архітектурних принципів для Web, котрі називаються REST, що фокусується на системних ресурсах, включаючи як ресурси можуть бути адресовані та передані по HTTP.

Головною ідеєю REST було скоріш використання добре розвиненого HTTP для передачі даних між пристроями, ніж використання протоколу, побудованого поверх рівня HTTP, для передачі повідомлень. Додаток, розроблений слідуючи принципам даного архітектурного підходу, буде використовувати HTTP для виконання викликів між пристроями не спираючись на складні об’єктні механізми на кшталт CORBA (Common Object Request Broker Architecture), RPC (Remote Procedure Call), чи SOAP. Відповідно, REST-додатки використовують функції HTTP запиту для надсилання,

зчитування та видалення даних, використовуючи повну функціональність HTTP CRUD (Create, Read, Update і Delete) операцій. До того ж REST може використовувати HTTPS, надаючи безпечне передавання даних.

CRUD операції разом з функціями HTTP REST та відповідними SQL операціями показані у таблиці 1.3. На відміну від SOAP, REST принципи легкі для розуміння та застосування розробнику, котрий знайомий з використанням HTTP.

Таблиця 1.3 – Відповідність HTTP-функцій SQL-операціям

CRUD-операції	***Ключові слова REST (HTTP)***	***Оператори SQL (база даних)***
CREATE – створити чи додати нові сутності	POST	INSERT
UPDATE – оновити чи редагувати існуючі дані	PUT	UPDATE
READ – зчитати, отримати дані	GET	SELECT
DELETE – видалити інсуючі дані	DELETE	DELETE/DROP

1.5.5 RPC. XML-RPC

RPC (Remote Procedure Call) – віддалений виклик процедур з допомогою XML. Сама методика віддаленого виклику відома давно і використовується в таких технологіях як CORBA, DCOM, SOAP. RPC слугує для побудови розподілених клієнт-серверних обчислень.

На дуже спрощеному прикладі механізм роботи виглядає наступним чином: додаток, виконучи обробку деяких даних на локальному комп’ютері звертається до деякої процедури. Якщо її реалізація присутня в програмі, то

процедура приймає параметри, виконує дію і повертає деякі дані. Якщо це віддалений виклик треба знати, де буде виконуватися процедура.

Запит на виконання процедури разом з параметрами записується у вигляді XML-документа і передається по мережі посередництвом HTTP на інший комп’ютер, де з XML-документа вилучається ім’я процедури, параметри та інша потрібна інформація. Після завершення роботи процедури формується відповідь, і вона передається комп’ютеру, що надіслав запит.

Але формат обміну даними при класичній моделі RPC при класичній моделі RPC залишається бінарним, що означає складність у роботі з ним, якщо потрібно організувати роботу розподіленої системи, де між окремими ділянками мережі сотять фаєрволи та проксі-сервери. До того ж різні платформи мають свої тонкощі у використанні.

Ця проблема передувала створенню похідної технології XML-RPC компанією UserLand Software Inc. Передача даних у них виконується протоколом HTTP, формат даних – XML. Це знімає обмеження, накладені на конфігурацію мережі, так і на маршут слідування пакетів, - виклики XML-RPC являють собою простий тип даних text/xml і вільно проходять через шлюзи там, де допускається ретрансляція HTTP-трафіку.

Повідомлення XML-RPC передаються методом POST протокола HTTP. Вони бувають трьох типів: запит, відповідь і повідомлення про помилку. Протокол передбачає сім простих типів даних і два складних, для передачі параметрів методу і повертаємих значень.

Більш складні типи даних, наприклад об'єкти, треба передавати у бінарному вигляді чи заміняти структурами. Не дивлячись на те, що протокол є дещо застарілим, але він ще й досі використовується і має певні переваги та недоліки.

Таблиця 1.4 – Основні характеристики технології XML-RPC

Характеристика	***XML-RPC***
Скалярні типи даних	Є
Структури	Є
Масиви	Є
Іменовані масиви та структури	Відсутні
Кодування, визначені розробником	Відсутні
Типи даних, визначені розробником	Відсутні
Деталізація помилок	Є
Легкість практичного застосування та засвоєння	Легкий для розуміння і практичного використання

1.6 Висновки

В цьому розділі були розглянуті основні класи клієнт-серверних архітектур, типи архітектур клієнт-серверних додатків, їх переваги та недоліки. Велика частина уваги приділена опису способів спілкування комп'ютерів у клієнт-серверній архітектурі. Проаналізувавши переваги та недоліки, можна зробити висновок, що для системи для тестування учнів середніх класів є доцільним обрання тришарової архітектури, як каркасу додатка. Ріст потужності клієнтських машин дозволяє реалізувати роботу системи в рамках моделі клієнт-сервер з «товстим» клієнтом, що відповідає сучасним тенденціям у галузі веб-додатків. Важливим аспектом є обрання типу взаємодії і з огляду на переваги та недоліки вищеописаних типів взаємодії, буде обрано архітектурний стиль REST, як швидкий для вивчення та легший для реалізації і взаємодії. Наступний розділ матиме за мету пояснити актуальність створення нової системи та підібрати інструменти для реалізації.

2. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА ТЕХНОЛОГІЙ

2.1 Огляд існуючих програмних рішень на ринку систем електронного навчання

Додатки для навчання можна умовно поділити на три наступні групи:

- інструментальне ПЗ для створення електронних навчальних матеріалів;
- платформи для розміщення матеріалів (надання доступу студентам чи школярам до матеріалів) і обліку діяльності учнів;
- платформи для інтерактивної взаємодії учасників навчального процесу (вебінари, форуми, чати, соціальні мережі).

Ці інструменти можуть бути представлені як окремими програмними продуктами, так і бути частково реалізовані в одному з них. Логічно, що їх об'єднання відбувається навколо платформи для розміщення матеріалів.

Платформа для інтеграції матеріалів може бути представлена в мережі Інтернет, за рахунок чого учасники навчального процесу отримують можливість електронного навчання «через одне вікно».

Така платформа отримала назву «система управління навчанням». Також часто вживається абревіатура СДН (система дистанційного навчання), тому що ці системи використовуються найчастіше для дистанційного навчання або для дистанційної підтримки навчального процесу.

Рисунок 1.11 показує основні аспекти, які беруться до уваги, коли мова йде про електронні системи для навчання.

Рисунок 2.1 — Можливості системи управління навчанням[14]

Система управління навчанням (LMS – Learning Management System) – це мережева платформа, що дозволяє:

- розміщувати електронний навчальний матеріал різних форматів;
- розмежовувати доступ до навчального матеріалу;
- здійснювати контроль за ходом вивчення матеріалу і виконання завдань;
- організовувати взаємодію учасників навчального процесу засобами мережевих комунікацій;
- розробляти електронний навчальний матеріал.

Для того, щоб зрозуміти потребу у створенні нового рішення, далі розглядються існуючі системи, показано їх переваги та недоліки, сфера використання і стислий огляд інтерфейсу користувача.

2.1.1 MOODLE

MOODLE – Modular Object-Oriented Dynamic Learning Environment – модульне об'єктно-орієнтоване динамічне середовище навчання. Вільно розповсюджувана система управління навчанням з відкритим вихідним кодом, яка використовується для створення веб-сайтів дистанційного навчання.

Автор системи - австралієць Мартін Дугіамас (Martin Dougiamas) - фахівець в області комп'ютерних наук і освіти, який захистив докторську дисертацію з проблематики використання вільного програмного забезпечення в інтернет-навчанні.

Рисунок 2.2 — Інтерфейс системи MOODLE

Moodle стала практичним результатом його досліджень, які тривають і донині. Навчальний матеріал в системі структурований в курсах. Курси можуть бути реалізовані в різних форматах уявлення вмісту. Вміст курсів представляють ресурси (веб-сторінки, файли), елементи діяльності – activities (завдання, тести, форуми, анкети і) і блоки (додаткова навігація, інформація або функціонал на сторінці курсу). Куратор чи вчитель може планувати онлайн-навчання, гнучко налаштовувати доступ до навчальних елементів, робити масову і вибіркову розсилку, аналізувати результати діяльності учнів. Можлива реалізація навчання по групах в рамках одного курсу, а також оперування «глобальними групами» в рамках всього сайту або категорії курсів. Є потужна система розробки тестів і аналізу результатів тестування. Можливо відтворення SCORM 1.2. Є вбудована система обміну повідомленнями (не миттєво). Кожен користувач має домашню сторінку (персональний простір). Високий ступінь занурення в інтерактивне і проектне навчання забезпечують такі елементи як глосарій (складання бази даних визначень-значень), база даних, вікі (спільне створення наборів веб-сторінок), семінар.

Переваги:

- найбільш популярна LMS в світі;
- безкоштовна, з відкритим вихідним кодом;
- велике онлайн-спільнота ;
- детальна документація і безліч навчальних посібників;
- має вбудовану систему розробки курсів, здатну реалізувати велику кількість педагогічних технологій, в тому числі проектне навчання.

Недоліки:

- складна для освоєння;
- вимоглива до сервера
- для розширення функціоналу або виправлення виникають в системі програмних помилок може знадобитися допомога професіоналів.

2.1.2 eFront

В порівнянні з Moodle, система eFront більше підходить для бізнес-структур чи невеликих навчальних центрів, ніж для освітніх організацій. Ця теза підтверджується списком організацій, які обрали eFront як системи управління навчанням, представленим на офіційному сайті проекту, а також тим фактом, що знайти в інтернеті більше одного живого прикладу хороших сайтів на цій платформі виявилося занадто важко. Однак це ні трохи не применшує достоїнств даної системи. Крім того, досвід Вищої школи економіки дозволяє припустити, що в цій системі було приділено недостатньо уваги з боку освітніх організацій. Однак, система eFront більше орієнтована на дистанційне, ніж змішане навчання (на відміну від Moodle). На користь бізнесу також говорить наявність в функціоналі редакції Enterprise можливості наділяти користувачів посадами і розподіляти по відділах, а також здійснювати пошук персоналу за компетенціями.

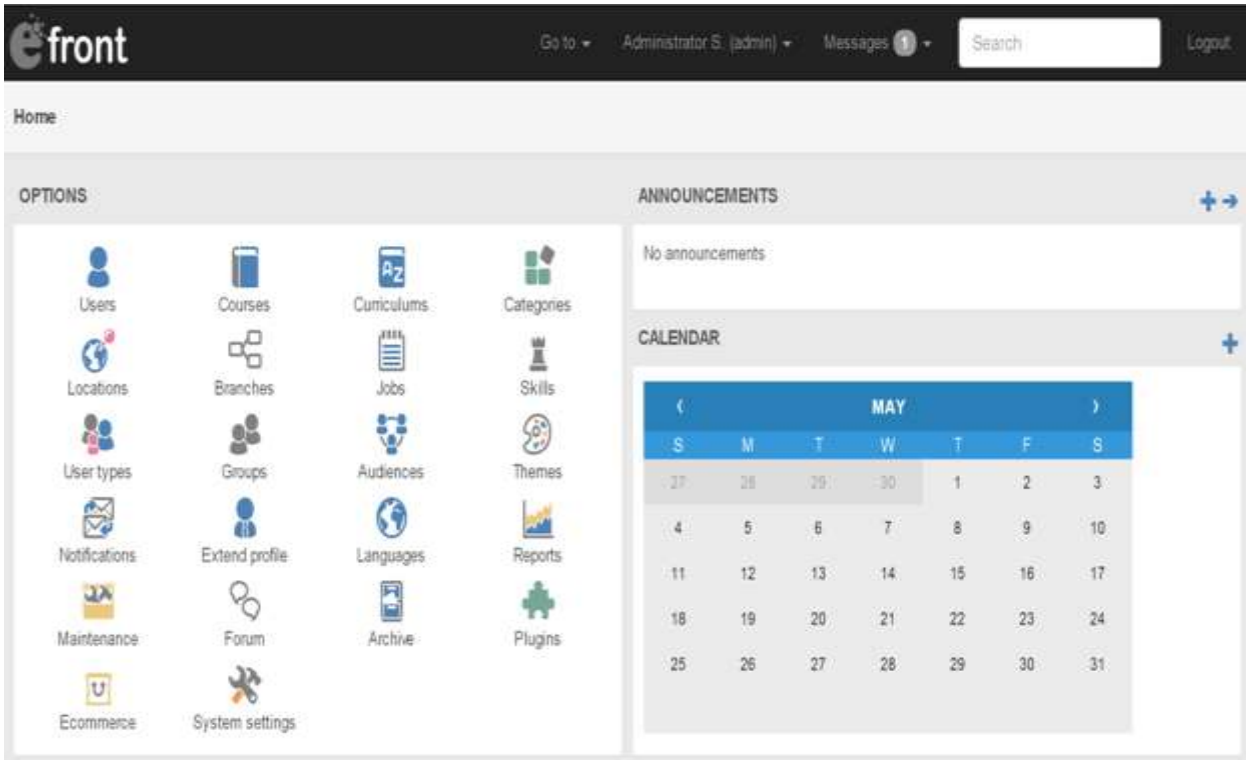

Рисунок 2.3 — Інтерфейс eFront

В системі чітко виділені модулі, навігація по яким представлена іконками на головній сторінці, а далі вкладками і стандартними елементоми. Навчальний матеріал представлений в уроках, які можуть бути об'єднані в курси, а ті, в свою чергу, можуть стати складовою навчального плану.

Урок складається з модулів, які можна ввімкнути або вимкнути. Макет уроку легко налаштовується. Навчальний матеріал в Уроці може бути представлений структурованими веб-сторінками, файлами, тестами, проектами, опитуваннями, SCORM, форумами.

Після установки відповідного доповнення можна також забезпечити інтеграцію з системою веб-конференцій Big Blue Button. Логіка вивчення матеріалу в уроці може бути визначена за допомогою набору правил, а навчання студент може почати і продовжити просто натиснувши на кнопку Play. Користувач в системі має власну домашню сторінку (Dashboard) з планом занять. Ролі користувачів мають різні набори прав, тому адміністратор системи не може паралельно виконувати функції куратора та студента (на відміну від Moodle). Однак є можливість прикріпити кілька профілів до одного логіну.

Переваги:

- використання технології Ajax робить роботу з системою комфортною та швидкою;
- чітка верстка веб-сторінок, стабільна робота програмної оболонки;
- докладні звіти про діяльність користувачів з гнучкою фільтрацією;
- наявність системи виявлення «прогалин» в компетенціях і автоматичний підбір навчального матеріалу для їх усунення (Enterprise Edition).

Недоліки:

- відносно невелика кількість інструментів для створення навчальних матеріалів;
- мало додаткових модулів;
- невелике співтовариство користувачів.

2.1.3 NDsmart

Система дистанційного навчання NDsmart – це сучасний освітній портал, призначений для комфортного, швидкого і ефективного навчання. Система містить безліч інструментів управління та моніторингу, а також має широкі можливості доопрацювання за бажанням замовника.

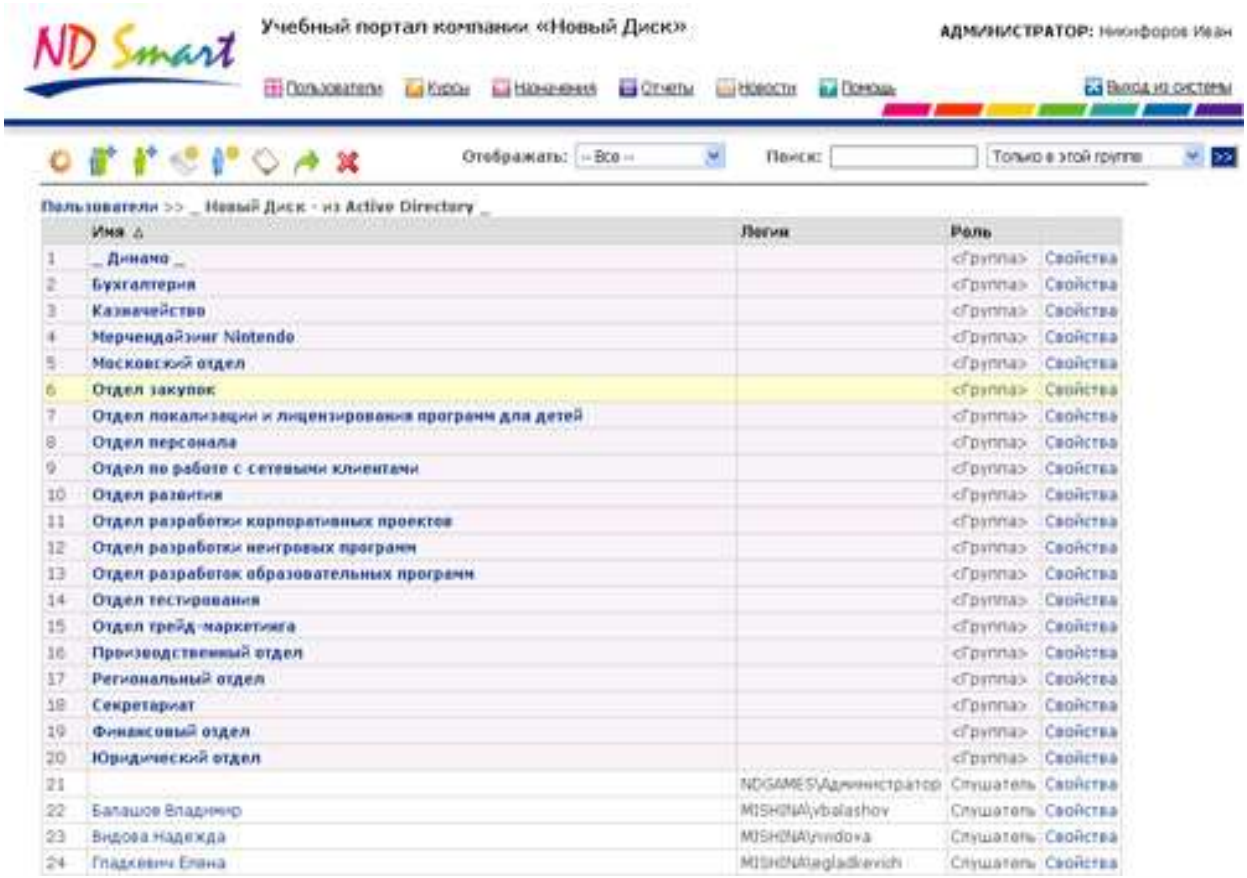

Рисунок 2.4 — Інтерфейс NDsmart

Для кожного користувача надається індивідуальний вхід в систему. Це дозволяє контролювати процес навчання кожного співробітника і виключає можливість використання системи випадковими користувачами через мережу.

Використання інтегрованої з Microsoft Windows Server Active Directory аутентифікації дозволяє користувачам при вході в систему не вказувати ім'я облікового запису та пароль, а самі облікові записи створюються автоматично при першому вході в систему, що значно спрощує роботу адміністратора.

Кожен користувач має одну з трьох ролей: слухач, викладач або адміністратор. Слухачі можуть проходити призначені ним курси і переглядати звітність за результатами навчання, залишати коментарі та вказувати рейтинги для курсів з використанням вбудованої системи зворотного зв'язку. На викладача покладено функції управління процесом навчання: це призначення курсів слухачам, перегляд звітів про їхню діяльність та редагування облікових записів.

Права окремого викладача на виконання кожного з цих дій можуть бути індивідуально налаштовані адміністратором. Адміністратор здійснює управління системою: це реєстрація, редагування і видалення облікових записів слухачів і викладачів, перегляд і редагування звітів, призначення курсів, призначення прав викладачів, створення та редагування новин.

Переваги:

- контекстно-залежна система довідки;
- вбудована інтеграція з Microsoft Windows Server Active Directory;
- можливість розподілу користувачів по групах з будь-яким рівнем вкладеності;
- можливість розподілу курсів по розділах;
- користувач має вбудовані фільтри, що дозволяють відображати курси в залежності від статусу, тематики, назви та інших параметрів;
- Гнучка система призначення курсів;

Недоліки:

- низькоякісний інтерфейс,
- доопрацювання та розширення дорого коштує.

2.1.4 Oracle Learning Management

Oracle Learning Management (OLM) – це корпоративна система управління навчанням, що надає ефективне, інтегроване, масштабоване інтернет-рішення для регулювання процесів навчання і підвищення кваліфікації співробітників,

партнерів і клієнтів компанії в зручний для них час і в зручному місці. OLM входить в систему Oracle HRMS комплексу додатків для бізнесу Oracle E-Business Suite.

OLM підтримує всі види діяльності з навчання (як в рамках традиційного, так і в режимі онлайн навчання): проектування курсів і програм навчання, планування і забезпечення ресурсів процесу навчання (аудиторії, інструктори, обладнання, дистанційні курси і т.д.) , зарахування на курси в аудиторіях і он-лайн-курси, ведення всієї історії навчання співробітників, облік фінансування.

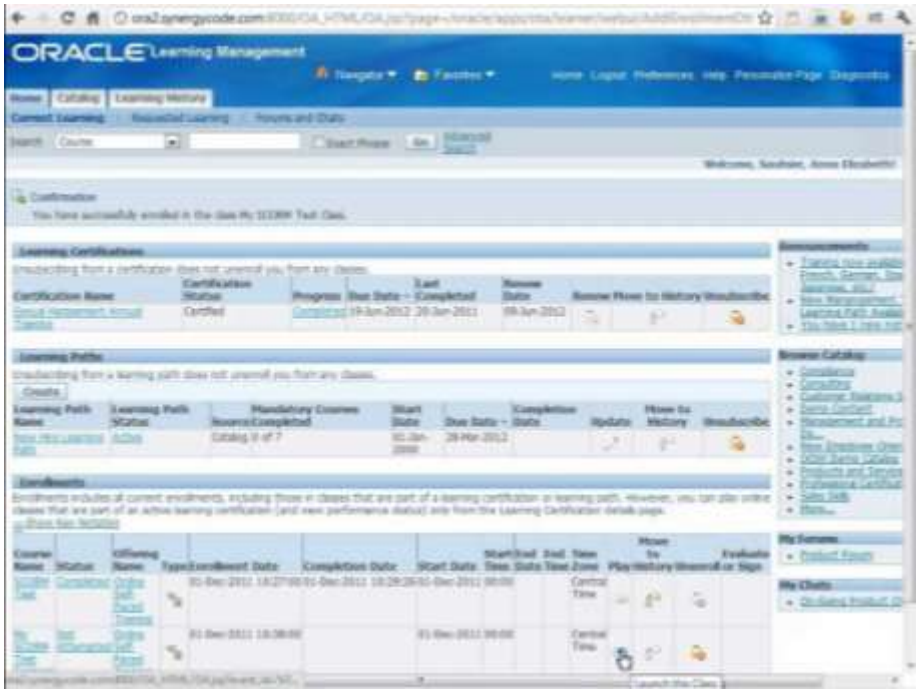

Рисунок 2.5 — Інтерфейс OLM

Система управління навчанням OLM:

- дозволяє об'єднати в єдиному інформаційному співтоваристві всіх учасників процесу навчання: учнів, викладачів, менеджерів навчального процесу і постачальників освітніх програм;

- охоплює всі стадії процесу навчання: складання курсів, планування навчального процесу, доставку слухачам курсів та інших необхідних матеріалів, контроль і аналіз проходження навчання;
- для великої організації це найбільш якісний, вигідний і ефективний спосіб оперативного навчання безлічі співробітників з дотриманням єдиних, прийнятих в організації стандартів і правил;
- надає можливість персоналізації навчання. Для кожної групи слухачів і для кожного окремого слухача може бути спроектований індивідуальний план навчання.

До основних недоліків даної системи можна віднести важку СДН, та обов’язковість встановлення СУБД Oracle, занадто важкий інтерфейс.

2.1.5 REDCLASS

Система дистанційного тренінгу REDCLASS версії - це комплекс програмно-апаратних засобів, навчальних матеріалів і методик навчання, які дозволяють дистанційно навчатися, підвищувати кваліфікацію, контролювати знання в будь-яких галузях діяльності людини, а також виробляти практичні навички по експлуатації і управління програмними продуктами, обладнанням і технологіями .

Для вирішення завдань дистанційного тренінгу REDCLASS володіє наступними засобами:

- середовище емуляції вправ дозволяє формувати і перевіряти навички роботи слухачів з системами, що володіють віконним інтерфейсом. Наприклад, може емулювати роботу MS Word для навчання користувачів цієї системи. Вправи для середовища емуляції створюються в Конструкторі вправ. Конструктор вправ дозволяє створювати вправи з розгалуженим сценарієм виконання і різними системами оцінювання дій користувачів;

- віртуальні лабораторії надають слухачам можливість роботи з реальними, а не емульованими, програмно-апаратними комплексами (стендами) в віддаленому режимі. Устаткування знаходиться в навчальному центрі, а слухачі отримують доступ до нього зі свого робочого місця. Віртуальні лабораторії дозволяють дати практику самостійної роботи, не обмеженої можливостями емулятора;
- електронний підручник призначений для доставки мультимедійного контенту на робоче місце слухача. Розробка курсів ведеться в автономному середовищі;
- система тестування призначена для контролю успішності слухачів. В системі передбачені тестування, що дозволяють здійснювати вхідний, вихідний і проміжний контроль знань, а також самооцінювання;
- система управління процесом навчання дозволяє організувати процес навчання в модулі управління каталогом курсів, користувачами системи та їх правами.

2.2 Огляд існуючих технологій

Платформа Java має велику кількість технологій та програмних платформ, фреймворків, що були створені для вирішення широкого спектру прикладних задач. Створення якісних додатків мовою Java не є легким завданням. Насправді, це дуже складно і може не дати того «насиченого» фронт-енду, котрого очікують користувачі. Це і стало поштовхом для створення веб-фреймворків. Як функціональні, так і не функціональні вимоги до веб-додатків зумовили появу потреби у створенні різноманітних веб-фреймворків, але, з іншого боку, це призвело до того, що кожен розробник, починаючи проект «з нуля», зіштовхується з питанням вибору потрібного інструмента.

В данний час найпопулярнішими технологіями створення Java-додатків є набір специфікацій Java EE та Spring фреймворк. Серед Java-розробників довгий час постійно ведуться дискусії на рахунок доцільності використання Java EE чи Spring Framework при розробці корпоративних систем і консенсусу не досягнуто ще з часу створення Spring, тому спільнота перейшла до певної «локалізації» спорів у контексті конкретних систем, відповідно, з'ясування доцільності використання того чи іншого інструмента зводиться до формування задачі і підбору конкретного рішення для її розв'язання. Цей факт дає змогу спочатку провести огляд технологій, потім Для вибору інструментів для реалізації додатку буде проведено огляд цих технологій.

2.3 Огляд Java Enterprise Edition

Java Platform, Enterprise Edition, скорочено Java EE — обчислювальна корпоративна платформа Java. Платформа надає API та виконавче середовище для розробки і виконання корпоративного програмного забезпечення, включаючи мережеві та веб-сервіси, та інші масштабовані, розподілені додатки. Java EE розширює стандартну платформу Java (Java SE - Java Standart Edition)[http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html].

J2EE є промисловою технологією і, в основному, її використовують у високопродуктивних проектах, у яких необхідна надійність, масштабованість, гнучкість.

Компанія Oracle, яка придбала Sun (фірму, що створила Java), активно просуває Java EE у поєднанні зі своїми технологіями, зокрема з СКБД Oracle.

2.3.1 Архітектура

Java EE складається з набору специфікацій, що реалізуються різними *контейнерами.* Контейнерами називаються засоби оточення часу виконання Java EE, що надають розміщеним на них компонентам певної служби,

наприклад управління життєвим циклом розробки, ін’єкція залежностей, паралельний доступ тощо. Такі компоненти використовують точно визначені контракти для зв’язування з інфраструктурою Java EE і з іншими компонентами.

Перед розгортанням, тобто доведеннями системи до готовності використання, вони повинні запаковуватися стандартним способом. Java EE повністю інтегрується з Java SE, що означає, що API-інтерфейси Java SE можуть використовувутися будь-якими компонентами Java EE. Рисунок 2.1 ілюструє логічні зв’язки між контейнерами. Стрілками представлені протоколи, використовувані одним контейнером для доступу до іншого.

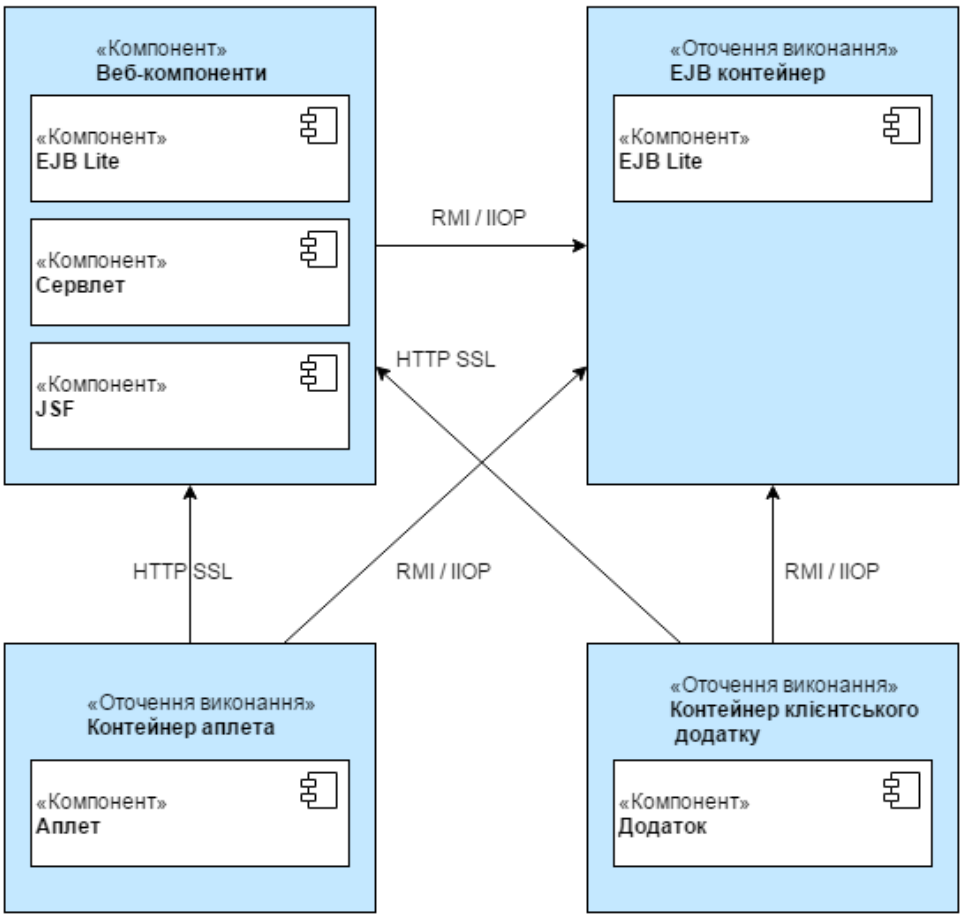

Рисунок 2.6 – Стандартні контейнери Java EE

2.3.2 Компоненти

Наприклад, веб-контейнер розміщує сервлети, інтерфейси для розширення функціоналу сервера, котрі можуть звертатися до компонентів EJB по протоколу віддаленого виклику процедур RMI-IIOP.

У оточенні часу виконання Java EE виділяють чотири типи компонентів, котрі повинна підтримувати реалізація.

- *Аплети* являють собою додатки з графічного користувацького інтерфейсу, виконуваного у браузері. Вони задіюють насичений інтерфейс Swing API для виробництва потужних користувацьких інтерфейсів.
- *Веб-додатки* складаються з сервлетів та їх фільтрів, слухачів подій, статично-динамічних сторінок JSP та фреймворк JSF. Вони виконуються у веб-контейнері і відповідають на HTTP запити від веб-клієнтів. Сервлети також підтримують кінцеві точки доступу SOAP та REST, опис яких було наведено у першій частині.
- *Додатками* називаються програми, виконувані на клієнтській стороні. Як правило, вони відносяться до графічного інтерфейсу користувача і застосовуються для пакетної обробки. Додатки мають доступ до усіх засобів шару домену.
- *Корпоративні додатки*, створені за технологіями EJB, служби обміну повідомленням Java Messaging Servise, Java API для транзакцій, асинхронних викликів, служби часу, протоколів RMI виконуються в контейнері EJB. Керовані контейнером компоненти EJB слугують для обробки транзакційної бізнес-логіки. Доступ до них може бути як локальним так і віддаленим по протоколу RMI (або HTTP для веб-служб SOAP і REST).

2.3.3 Контейнери

Контейнери розгортають свої компоненти, надаючи їм відповідні базові сервіси. Контейнери дозволяють розробнику концентруватися на реалізації бізнес-логіки, а не вирішувати технічні проблеми, присутні в корпоративних додатках. Контейнери Java EE надають багато сервісів, котрі здатні задовольнити будь-які потреби бізнесу, *основними* з них є:

- *Java API для транзакцій* – цей сервіс надає інтерфейс розділення транзакцій, використовуваний контейнером і додатком. Він також надає інтерфейс взаємодії між диспетчером транзакцій і диспетчером ресурсів.
- *Інтерфейс персистентності Java* – стандартний інтерфейс об’єктно-реляційного відображення. За допомогою вбудованої мови запитів JPQL можна звертатися до об’єктів, що зберігаються в основній базі даних.
- *Валідація* – завдяки валідації компонентів констатується обмеження цілісності на рівні класу та мтоду.
- *Інтерфейс Java для доступу до служб повідомлень* – дозволяє компонентам асинхронно обмінюватися даними через повідомлення.
- *Обробка XML* – більшість компонентів Java EE можуть розгортатися з допомогою опціональних дескрипторів розсортування XML, а додаткам часто доводиться маніпулювати XML-документами.
- *Обробка JSON* – інтерфейс для обробки JSON з’явився тільки в Java EE 7. Він дозволяє додаткам синтаксично аналізувати, генерувати, трансформувати та робити запити JSON.

- *Служба безпеки* – служба аутентифікації і авторизації для платформи Java дозволяє сервісам аутентифікуватися та встановлювати права доступу, обов’язкові для користувачів.
- *Веб-служби* – Java EE підтримує веб-служби SOAP і REST. Інтерфейс Java для веб-служб на XML (JAX-WS), замінивший інтерфейс Java з підтримкою виклику віддалених процедур на основі XML (JAX-RPC), забезпечує роботу веб-служб, працюючих по протоколу SOAP/HTTP. Інтерфейс Java для веб-служб RESTful (JAX-RS) підтримує веб-служби, що використовують стиль REST.

До цього списку можна також віднести *ін’єкцію залежностей* – можливість впровадження ресурсів в керовані компоненти. До таких ресурсів відносяться джерела даних, фабрики класів, одиниці персистентності, компоненти EJB тощо.

Підсумовуючи інформацію про архітектуру Java EE, можна сказати, що якщо компанія розробляє Java-додатки з додаванням таких можливостей, як управління транзакціями, безпека, паралельний доступ чи обмін повідомленнями, то варто звернути увагу на цю платформу. Вона стандартизована, працює з різноманітними протоколами, а компоненти розгортаються в контейнери, завдяки чому можна користуватися багатьма сервісами. Java EE, слідуючи трендам, спрощує використання веб-рівня, але суттєвим мінусом є те, що Java EE – це лише набір стандартизованих специфікацій. Для розробника це означає те, що імплементації цих специфікацій він має обирати сам, а це може призвести до додаткових складностей. Іншим мінусом є складність тестування EJB, що призводить до потреби використання спеціальних фреймворків, котрі дещо спрощують тестування, але потребують повторного написання коду у тестах.

2.4 Огляд Spring Framework

Фреймворк Spring має багато функціональних можливостей, однак, якщо розбити їх на складові частини, можна виділити дві його основні особливості: ін’єкція залежностей (DI) і аспектно-орієнтоване програмування (AOP).

Spring – це фреймворк з відкритим вихідним кодом, що вільно розповсюджується, створений з метою усунути складності розробки корпоративних додатків і зробити можливим використання простих компонентів JavaBean для досягнення всього того, що раніше було можливим тільки з використанням EJB. Однак, область використання Spring на обмежується розробкою програмних компонентів, що виконується на стороні сервера. Будь-який Java-додаток може використовувати переваги фреймворка в планы простоти, тестованості і слабого зв’язування.

В основі практично усіх його особливостей лежать декілька фундаментальних ідей, направлених на досягнення головної мети – *спрощення розробки додатків мовою JAVA.* Для досягнення цього Spring використовує чотири стратегії:

- *легковагість* та *ненав’язливість* завдяки застосуванню простих Java-об’єктів (POJO, Plain Old Java Object):
- *слабке зв’язування* посередництвом *ін’єкції залежностей* та орієнтованості на інтерфейси;
- *декларативне програмування* через *аспекти* і загальноприйняті угоди;
- *зменшення об’єму* типового коду через аспекти і шаблони.

Практично усі можливості фреймворка Spring базуються на цих стратегіях. В решті частин даного підпункту будуть розглядатися більш детально.

2.4.1 Використання POJOs

Порівняно з EJB, Spring має велику перевагу – він не примушує, наскільки це можливо, реалізовувати специфічні для нього інтерфейси чи наслідувати свої класи. Навпаки, часто класи додатків взагалі не мають ніяких рис, по котрим можно було б стверджувати. Для порівняння нижче буде наведено код двох синонімічних класів, - один з Java EE, інший – з екосистеми Spring.

Специфікація EJB Java EE примушує реалізовувати непотрібні методи, що видно у лістингу на наступній сторінці:

```
1  package com.habuma.ejb.session;
2  import javax.ejb.SessionBean;
3  import javax.ejb.SessionContext;
4
5  public class HelloWorldBean implements SessionBean {
6          // Методи, котрі є збитковими і зайвими
7      public void ejbActivate() {}
8
9      public void ejbPassivate() {}
10
11     public void ejbRemove() {}
12
13     public void setSessionContext(SessionContext ctx) {}
14
15     public String sayHello() {
16         // Основна логіка компонента EJB
17      return "Hello World";
18     }
19
20     public void ejbCreate() {}
21 }
```

Рисунок 2.7 – Лістинг коду компонента у специфікації EJB 2.1

Фреймворк Spring не вимагає виконання необгрунтованих вимог до аналогічного класу, що суттєво відображається на кількості коду з ростом кількості класів.

Одним із механізмів Spring, що збільшує потужність вищеописаних простих об'єктів, є їх об'єднання з допомогою широковживаного у корпоративних системах патерну ін'єкції залежностей (Dependency Injection).

```
1  package com.habuma.spring;
2  public class HelloWorldBean {
3
4   public String sayHello() {
5       // Це все, що необхідно для
6       // реалізації аналогічного функціоналу
7    return "Hello World";
8   }
9  }
```

Рисунок 2.8 – Лістинг коду компонента у Spring

2.4.2 Ін’єкція залежностей

Ін’єкція залежностей (Dependency Injection) дозволяє спростити програмний код проекту, полегшити його розуміння та тестування. Будь-який нетривіальний додаток складається з багатьох класів. Зазвичай кожен об’єктв *відповідальний за створення власних посилань на об’єкти, з котрими він взаємодіє*, його залежностями.

Це може призвести до сильної зв’язаності і складнощам при тестуванні. Завдяки ін’єкції залежностей об’єкти *отримують* свої залежності від третьої сторони, котра координує роботу кожного об’єкта в системі.

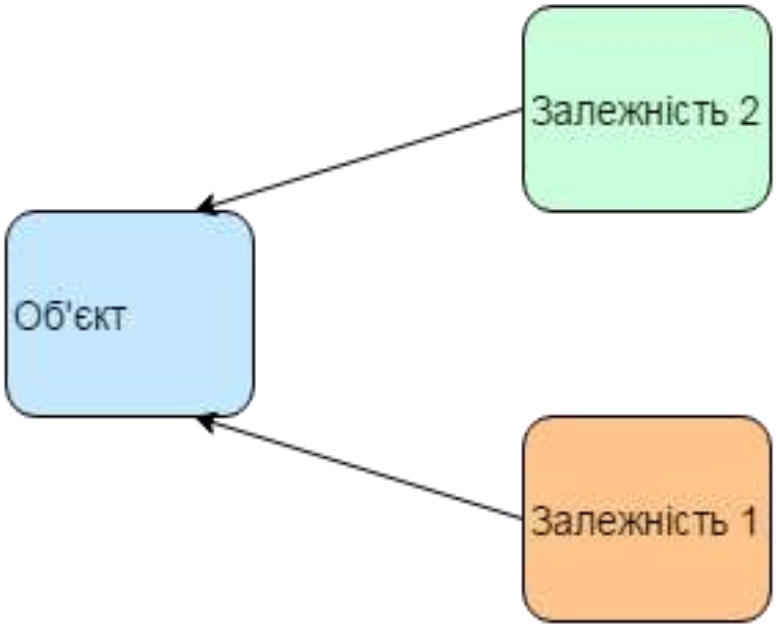

Рисунок 2.9 – Механізм ін’єкції залежностей

Механізм ін’єкції залежностей заснований на наданні об’єкту залежностей ззовні, а не отриманні цих залежностей самим об’єктом.

2.4.3 Аспектно-орієнтоване програмування

Хоча ін’єкція залежностей дозволяє послабити зв’язок між компонентами, *аспектно-орієнтоване програмування* доповнює об’єктно-орієнтоване програмування, доповнюючи його.

Аспекти дозволяють полегшити структуру шарів, котрі проходять наскрізь через інші шари багатошарової архітектури, як-то управління транзакціями, кешування чи безпека. Функції, що стосуються багатьох щарів англійською мовою називаються cross-cutting concerns, що дослівно українською звучить як *наскрізні проблеми,* далі буде використовуватися англомовний варіант визначення. Cross-cutting concerns не стосуються бізнес-логіки додатку.

2.4.4 Опис основних модулів

Spring Framework надає більше 20 модулів, котрі можуть використовуватися залежно від вимог до додатку.

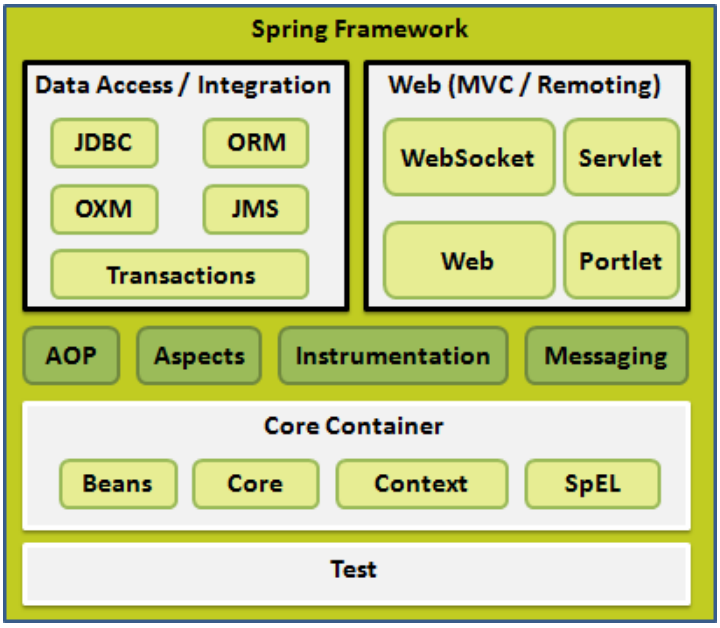

Рисунок 2.10 – Модулі Spring Framework[16]

Короткий опис основних модулів дасть зрозуміти структуру Spring Framework.

- Core – модуль ядра, надає центральний функціонал ін’єкції залежностей;
- Bean – надає складну реалізацію фабрики зіставних об’єктів додатку;
- Context – модуль, побудований поверх двох попередніх, і є оточенням для доступу до будь-яких визначених і налаштованих об’єктів;

Spring Framework надає модуль для доступу до даних, котрий підтримує доступ до сховища даних шляхом як SQL-запитів, об’єктно-реляційного маппінгу, так і підтримує обробку транзакцій.

Основними модулями шару представлення є Web – модуль для базового функціоналу веб-додатків; Web MVC – модуль містить реалізацію патерна

модель-представлення-контроллер для веб; Web Socket – модуль, що надає підтримку двошляхової комунікації між клієнтом та сервером.

2.5 Висновки

Даний розділ присвячений порівнянню існуючих рішень, як з точки зору інструментів розробника для створення багатошарових додатків, так і існуючих систем на ринку електронних систем навчання. Проведено стислий огляд та аналіз обраних систем для навчання. На основі нього можна зробити висновок про наявність потреби створення рішення, яке буде задовольняти потреби користувачів. Реалізація повинна ввібрати максимум найкращих рис з розглянутих додатків, а саме простий, зрозумілий інтерфейс, легкість в опануванні. Передбачається також обов’язкова можливість розгортання додатку на локальному сервері школи, так як потеба в цьому виникає часто.

У цьому розділі було описано дві основні технології на платформі Java для побудови багатошарових архітектур додатків, а саме Java EE та Spring Framework. Spring Framework дозволяє швидше розроблювати багатошарові системи, зменшуючи кількість написаного коду та полегшуючи тестування. До того ж він має широкий спектр модулів, що здатні задовольнити майже будь-які потреби як розробника, так і додатку. Спільнота Spring Framework розробила величезну кількість готових каркасів додатків з різними комбінаціями інструментів та зконфігурованих компонентів, що також вплинуло на ріст популярності фреймворка і на вибір цієї платформи. Ці факти дають підґрунтя для обрання його, як базової технології для реалізації додатку для тестування.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

Для створення серверної частини використовується Spring Framework та модуль Spring Data JPA для об’єктно-реляційного відображення і компонент Spring Data REST для побудови REST-сервісів.

Першим кроком зроблена конфігурація проекту, а саме підключення бази даних PostgreSQL для зберігання сутностей та база даних HSQLDB, котра міститься в пам’яті, для пришвидшення виконання відлагодження проекту, модульного та інтеграційного тестування на етапі розробки.

Веб-контейнером було обрано популярну реалізацію специфікації сервлетів Apache Tomcat 8, що дозволяє запускати створений додаток. Як реалізація ORM (Object Relational Mapping – об’єктно-реляційне відображення) використовується Hibernte Framework, котрий реалізує специфікацію JPA.

Клієнтська частина реалізована з використанням фреймворку AngularJS, каскадних таблиць стилів CSS3 та мови розмітки HTML5.

3.1 Шар моделі

3.1.1 Шар POJO-об’єктів

Першим кроком було створення основних сутностей бази даних додатку, таких як:

- клас користувачів User та похідні від нього класи вчителів Teacher, учнів Pupil, та батьків Parent;
- клас тестів Test;
- клас питань Question;
- відповіді на питання Answer;

- та деякі допоміжні класи, QuestionType, UserRole та інші службові класи.

Проведено їх об’єктно-реляційне зв’язування. Як приклад розглянуто лістинг на рисунку 3.1.

```java
package model.pojos;

import lombok.Data;

import javax.persistence.*;
import java.util.List;

@Data
@Entity
@Table(name = "questions")
public class Question {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(nullable = false)
    private String content;

    // @ManyToOne
    @Enumerated(EnumType.ORDINAL)
    private QuestionType type;

    private String subject;

    private String hint;
    @ManyToOne
    @JoinColumn(name = "author_id", referencedColumnName = "id")
    private User author;

    @ManyToMany
    @JoinTable(name = "questions_possible_answers",
            joinColumns = {@JoinColumn(name = "QUESTION_ID", referencedColumnName = "id")},
            inverseJoinColumns = {@JoinColumn(name = "ANSWER_ID", referencedColumnName = "id")})
    private List<Answer> possibleAnswers;

    @ManyToMany
    @JoinTable(name = "questions_right_answers",
            joinColumns = {@JoinColumn(name = "question_id", referencedColumnName = "id")},
            inverseJoinColumns = {@JoinColumn(name = "answer_id", referencedColumnName = "id")})
    private List<Answer> rightAnswers;

    //private List<URL> mediaResources;
}
```

Рисунок 3.1 – Лістинг класу Questions з використанням ORM-анотацій

Це вже описаний POJO-об’єкт, поля якого анотовані анотаціями об’єктно-реляційного відображення, що дозволяє зв’язати поля об’єктів з таблицями бази даних та їх атрибутами. В даному прикладі добре показані можливості

фреймворку. Основні з них – організація відношень типу *багато-до-одного* (анотація @ManyToOne), *багато-до-багатьох* (анотація @ManyToMany). Також вказується інформаці для створення проміжних таблиць при організації відношення багато-до-багатьох в об'єктній моделі.

Схожим способом імплементовані всі інші сутності додатку, що формують модель даних. Іноді для серіалізації об’єктів в базу даних треба змінити деякі поля, для цього використовуємо прості класи-конвертери, як, наприклад конвертер класу типів питань QuestionTypeConverter.

Методи цього класу будуть автоматично викликатися під час збереження об’єкта даного класу в базу даних.

```java
package model.converters;

import model.pojos.QuestionType;

import javax.persistence.AttributeConverter;
import javax.persistence.Converter;

@Converter
public class QuestionTypeConverter implements AttributeConverter<QuestionType, String> {

    @Override
    public String convertToDatabaseColumn(QuestionType questionType) {
        switch (questionType){
            case SINGLE_ANSWER: return "SINGLE";
            case MULTIPLE_ANSWERS: return "MULTIPLE";
            case COMPREHENSIVE_ANSWER: return "COMPREHENSIVE";
            default: throw new IllegalArgumentException("Unknown question type: " + questionType);
        }
    }

    @Override
    public QuestionType convertToEntityAttribute(String dbData) {
        switch (dbData) {
            case "SINGLE": return QuestionType.SINGLE_ANSWER;
            case "MULTIPLE": return QuestionType.MULTIPLE_ANSWERS;
            case "COMPREHENSIVE": return QuestionType.COMPREHENSIVE_ANSWER;
            default: throw new IllegalArgumentException("Unknow database data: " + dbData);
        }
    }
}
```

Рисунок 3.2 – Лістинг класу QuestionTypeConverter

3.1.2 Шар репозиторіїв

У шарі моделі даних, поверх шару POJO-об’єктів та конвертерів, є шар репозиторіїв. Шар репозиторіїв розташований між шаром домену та шаром даних. Концептуально, репозиторій інкапсулює набір об’єктів, збережених у сховищі і операції, виконані над цими об’єктами. Це надає більш об’єктно-орієнтований вигляд шару даних. Spring Data JPA надає ряд інтерфейсів, які можна наслідувати для спрощення реалізації цього патерну. Одним із основних інтерфейсів репозиторіїв є інтерфейс CrudRepository.

```
package org.springframework.data.repository;

import ...

@NoRepositoryBean
public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {
    <S extends T> S save(S var1);

    <S extends T> Iterable<S> save(Iterable<S> var1);

    T findOne(ID var1);

    boolean exists(ID var1);

    Iterable<T> findAll();

    Iterable<T> findAll(Iterable<ID> var1);

    long count();

    void delete(ID var1);

    void delete(T var1);

    void delete(Iterable<? extends T> var1);

    void deleteAll();
}
```

Рисунок 3.3 – Інтерфейс CrudRepository

Наслідуючи цей інтерфейс з’явилася можливість викликати у класа-наслідника такі методи як:

- save;
- findOne;
- exists;
- findAll;

- count;
- delete;
- deleteAll;

У реалізації додатку для кожної з сутностей створені репозиотрій. Це рішення полегшує подальшу розробку, адже при подальшому проектуванні шару контролерів та логіки можна мислити в звичному об’єктному контексті, не замислюючись над тонкощами реалізації шару даних.

3.1.3 Шар контролерів

HTTP запит, надісланий клієнтом створеному REST-сервісу перш за все обробляється DispathcerServlet-ом.

```
@RestController
@RequestMapping(value = "/tests/{testId}")
class TestsRestController {

    @Autowired
    private final TestRepository testRepository;

    @Autowired
    private final QuestionRepository questionRepository;

    @RequestMapping(method = RequestMethod.POST)
    ResponseEntity<?> add(@PathVariable String testId, @RequestBody Test t) {
        //код додавання тесту
    }

    @RequestMapping(value = "/{testId}", method = RequestMethod.GET)
    Bookmark readTest(@PathVariable String testId) {
        //повернення тесту з питаннями
    }

    @RequestMapping(method = RequestMethod.GET)
    Collection<Test> readTests() {
         // валідація і повернення тестів
    }

    @Autowired
    TestsRestController(TestRepository testRepository, QuestionRepository questionRepository1) {
        this.testRepository = testRepository;
        this.questionRepository = questionRepository1;
    }

    private void validateTest(String testId) {

    }
}

@ResponseStatus(HttpStatus.NOT_FOUND)
class UserNotFoundException extends RuntimeException {
    //відправка помилки пошуку
}
```

Рисунок 3.4 – типовий REST контролер

Після обробки DispathcerServlet знаходить потрібний контролер запитуючи його спеціального класу, котрий містить список відповідності ресурсів та контролерів і викликає його відповідні методи, в залежності від типу запиту.

Контролери та DispatcherServlet-и контролюються Spring Data REST. Після того як контролеру делеговано запит, викликається функціонал, специфічний для контролера.

3.1.4 Шар представлення

Шар представлення при даній архітектурі требя розглядати з точки зору архітектури сервера та клієнта. Для серверя представленням є об’єкт у форматі JSON. Формат JSON – це популярна об'єктна нотація, що широко використовується у REST архітектурах. Для наочності на рисунку 3.5 продемонстровано як створений REST-сервер відповідає на GET запит питання з порядковим номером 4 JSON-об’єктом.

```
evgenii@evgenii-HP-ProBook-4530s:~$ curl -H "Content-Type: application/json" -X
GET localhost:8080/questions/4
{
  "content" : "Question 4",
  "type" : "COMPREHENSIVE_ANSWER",
  "subject" : "History",
  "hint" : "very easy",
  "content" : [ ],
  "links" : [ {
    "rel" : "self",
    "href" : "http://localhost:8080/questions/4"
  }, {
    "rel" : "question",
    "href" : "http://localhost:8080/questions/4{?projection}"
  }, {
    "rel" : "author",
    "href" : "http://localhost:8080/questions/4/author"
  }, {
    "rel" : "possibleAnswers",
    "href" : "http://localhost:8080/questions/4/possibleAnswers"
  }, {
    "rel" : "rightAnswers",
    "href" : "http://localhost:8080/questions/4/rightAnswers"
  } ]
```

Рисунок 3.5 – надсилання GET-запиту до сервера

В той же час об’єкти, що повертаються сервером слугують моделлю даних для клієнтської частини. Так як цей формат заснований на мові JavaScript, то говорити про простоту інтеграції його з JavaScript фреймворком AngularJS, котрый використовується на клієнтській частині є зайвим.

3.2 Функціонал клієнтської частини

Комунікація клієнтської частини з сервером відбувається шляхом надсилання клієнтом HTTP запитів. Дизайн додатку виконаний у мінімалістичному стилі і не перенасичений компонентами, що робить його використання простим, ненав’язливим та водночас функціональним. Наприклад, сторінка вибору тестів у верхній частині містить назву тесту та зображення і кнопку початку тесту.

Рисунок 3.6 – Коротка інформація про тест

У нижній частині сторінки розташований список тестів з обраного предмету та випадне меню, що дозволяє змінити предмет.

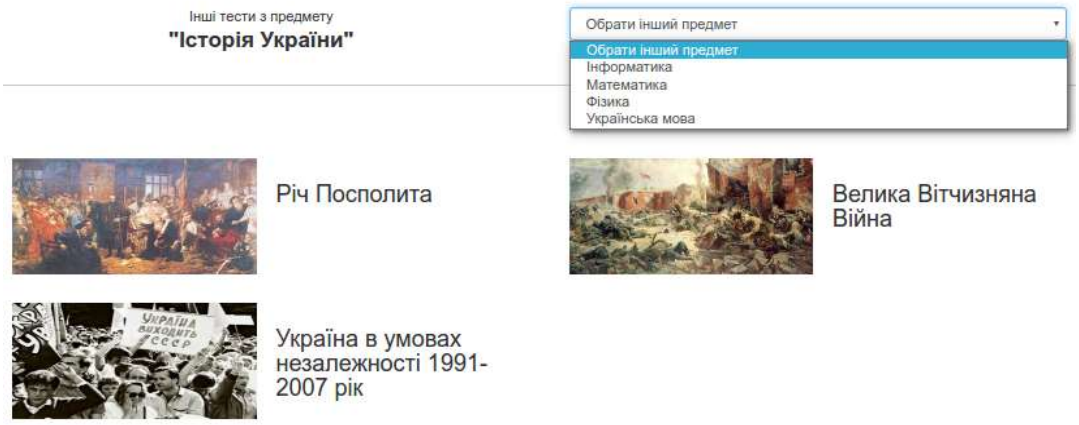

Рисунок 3.7 – Список тестів з обраного предмету у нижній частині сторінки

Натиснувши кнопку «Розпочати тест» користувач потрапляє на сторінку з тестом, де розташовані сторінки з запитанням та варіантами відповідей.

Така конструкція сторінки дає можливість створювати різноманітні тести і завантажувати кожне питання окремо, при потребі. Даний підхід знижує затримку при завантаженні сторінки, так як розмір JSON-відповіді від сервера значно менший у такому випадку. На рисунку 3.8 показано вигляд тестового питання при проходженні тесту. Можливість використати підказку, продемонстрована на рисунку 3.9, робить процес проходження тесту інтерактивним та може знизити нервове напруження учня при навчанні. Нижня частина сторінки зайнята списком питань, що наявні у поточному тесті. Це дає можливість переходити на інше питання, якщо у користувача виникли труднощі з поточним питанням і повернутися до нього пізніше. Таблиця з наявними питаннями має такий вигляд, як показано на рисунку 3.10.

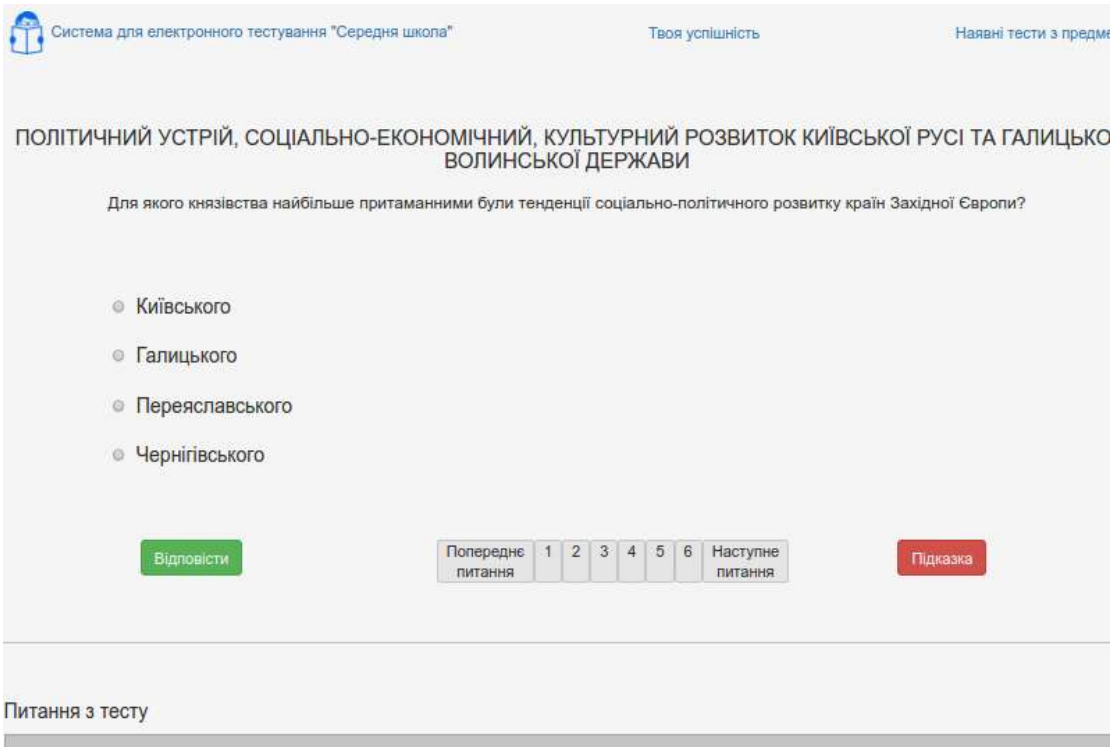

Рисунок 3.8 – Вигляд тестового питання на сторінці

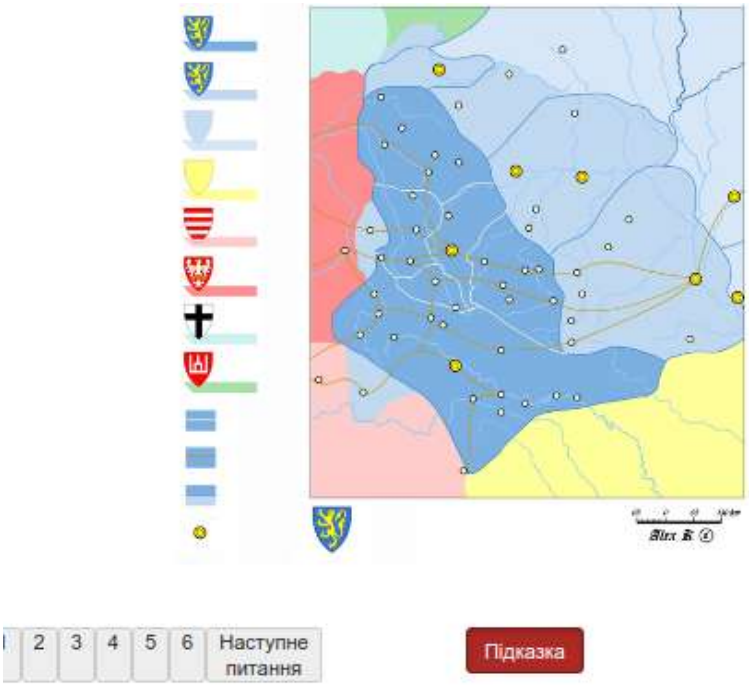

Рисунок 3.9 – Спливаюча підказка у вигляді картинки

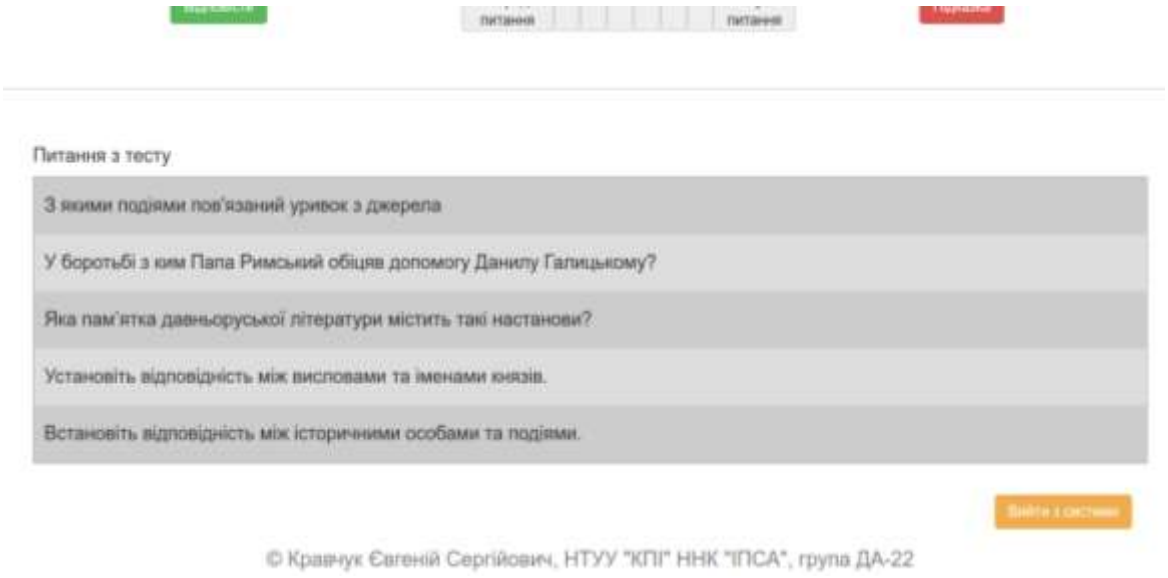

Рисунок 3.10 – Таблиця питань тесту

Вчитель може створити новий тест з питаннями, назначити варіанти відповідей кожному питанню та обрати вид питання з переліку, як показано на рисунку 3.11. Також тест може бути включений у матеріал певного предмету.

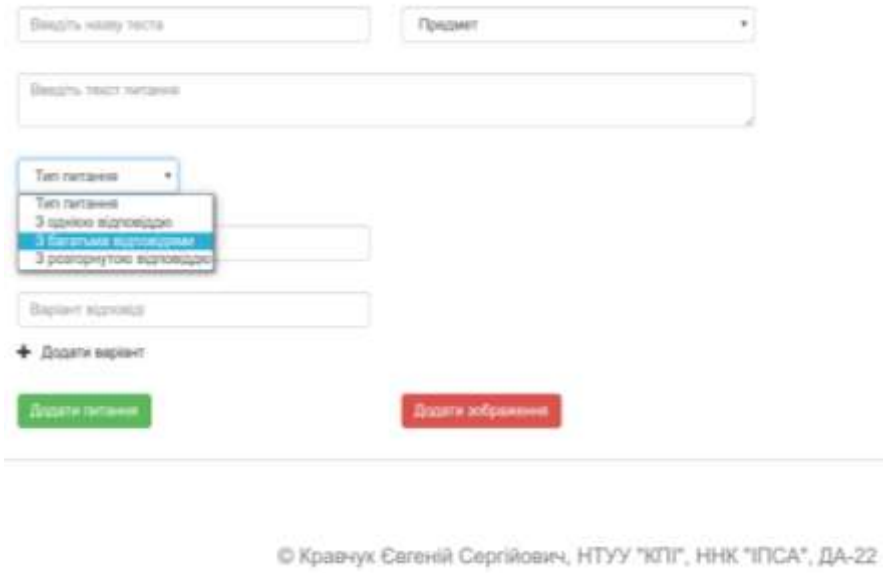

Рисунок 3.11 – Створення тесту

Базовий функціонал розширює можливість перегляду статистики, показаної на рисунку 3.12, з кількістю правильних відповідей, набрану учнями по певному тесту.

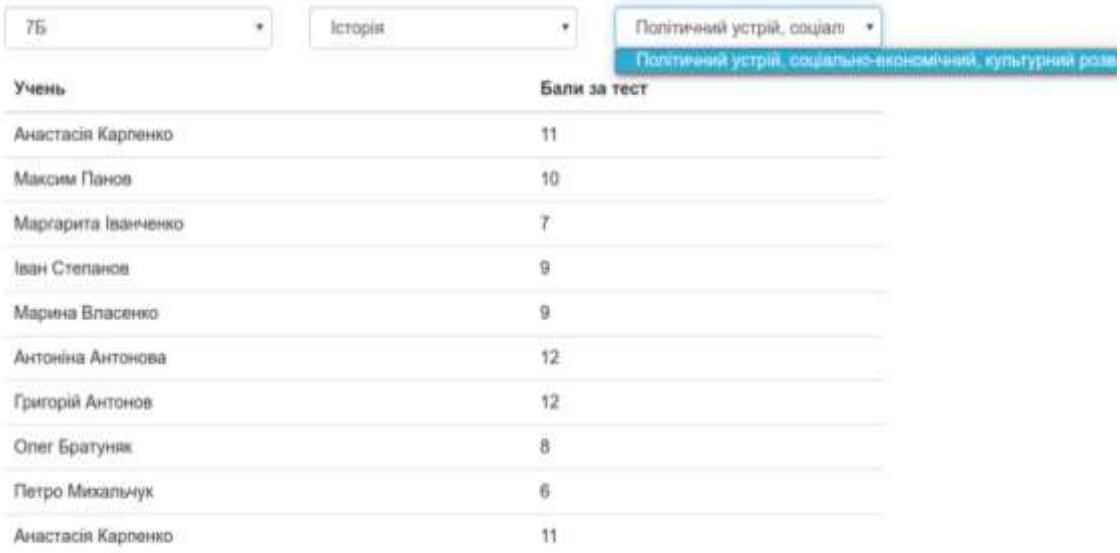

Рисунок 3.12 – Перегляд кількості балів, набраних учнями

3.3 Висновки

Розроблено клієнт-серверну систему. Котра відповідає архітектурному стилю REST, на основі якого побудована взаємодія клієнта і сервера. Показано приклади використання обарногих інструментів розробки та приклади класів з кожного шару створеного багатошарового додатку. В ході розробки було вирішено розділити шар моделі даних та шар логіки на декілька складових шарів. Демонстрація основного функціоналу системи відбулася на прикладі веб-клієнта, побудованого за допомогю фреймворка AngularJS, Також показано працездатність окремо серверної частини.

4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даній роботі проводиться аналіз основних підходів при проектуванні корпоративних багатошарових клієнт-серверних додатківо з поадльшою реалізацією у вигляді додатку для тестування учнів середніх класів з використанням найкращих технік для даної задачі. Під час виконання використовується мова Java та Spring Framework, як каркас для додатку. Графічний інтерфейс користувача виконаний у вигляді веб-інтерфейсу з використанням HTML, CSS та мови JavaScript.

Для техніко-економічного аналізу програмного продукту буде використаний метод функціонально-вартісного аналізу (ФВА). Основою ФВА є функціональний підхід, згідно з яким об'єктом аналізу тобто не сам продукт, а функції, які він виконує. ФВА проводиться в два етапи: функціональний аналіз і вартісний аналіз.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

- визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат;
- для кожної функції визначаються повні річні витрати й кількість робочих часів;

- для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат;
- після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту;

4.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА. Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на смартфонах Apple Iphone із стандартним набором компонент;
- забезпечувати високу швидкість роботи з сервісами у реальному часі;
- забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;
- передбачати мінімальні витрати на впровадження програмного продукту.

4.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0– розробка програмного продукту, який надає можливість контролю та моніторингу якості освіти учнів середніх класів шляхом переходу до електронного тестування. Виходячи з конкретної мети, можна виділити наступні основні функції програмного продукту:

- F1 – вибір типу архітектури клієнт-сервер;
- F2 – вибір стеку технології для серверної частини;
- F3 – вибір JavaScript фреймворків для клієнтської частини;

Кожна з основних функцій може мати кілька варіантів рішення:

для F1:

а) з товстим клієнтом;

б) з тонким клієнтом ;

в) спільна обробка даних;

для F2:

а) на основі модулів Spring Framework;

б) на основі JPA, Jersey, Jetty, Google Guice;

для F3:

а) AngularJS;

б) ReactJS;

Побудуємо позитивно-негативну матрицю, показану в таблиці 1.1.На основі аналізу позитивно-негативної матриці робимо висновок, щопри розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам.

F1: Звертаючи увагу на те, що у навчальних закладах комплектація комп'ютерних класів є мінімальною, обрано варіант клієнт-серверної архітектури зі спільною обробкою даних.

F2: Широкий спектр функціональних модулів та налаштувань, детальні приклади, котрі нівелюють складність та комплексність дають підґрунтя для обрання Spring Framework.

F3: Так як швидкість у вивченні є основним показником і система буде мати невелику кількість представлень прийнятними є обидва варінти.

У зв'язку із оглядом основних функцій програмного продукту наведеним вище, будемо розглядати наступні варіанти реалізації:

А: F1 в) – F2 а) – F3 а)

Б: F1 в) – F2 а) – F3 б)

Далі, у таблиці 4.1 наведено позитивно-негативну матрицю для обраних

функцій та варіантів реалізації з позитивними та негативними наслідками для кожного варіанту реалізації.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	а)	Зменшення навантаження на сервер	Збільшення вимог до клієнтських комп’ютерів
	б)	Має високу швидкодію на клієнті	Вся логіка виконується на сервері, що негативно впливає на швидкодію
	в)	Зручність у підтримці та розподіленні функціоналу;	Дублювання логіки на клієнтах і на сервері
F2	а)	Високий рівень сумісності; велике community розробників; легкість у дотриманні DRY при написанні коду	Високий поріг необхідних для розробки знань, комплексність фреймворку
	б)	Швидкість розробки, простота, можливість писати KISS код	Складність у налагодженні взаємодії на всіх рівнях; необхідність написання збиткового коду для взаємодії модулів
F3	а)	Швидке опанування; велика кількість якісних прикладів	Обмежені можливості при використанні у великих системах
	б)	Легкі для розуміння компоненти;	Робота тільки з представленням; немає AJAX

4.2 Обґрунтування системи параметрів програмного продукту

4.2.2 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, звернемо увагу на таблицю 4.2, будемо використовувати наступні параметри:

Таблиця 4.2 Визначення параметрів продукту

X1 – швидкість виконання	X2 – час виконання	X3 – ресурсоємність	X4 – потенційний об’єм програмного коду
Відображає швидкість виконання по часу в залежності від мови програмування	Відображає час виконання	Відображає навантаженість на систему, спричинену роботою програмного продукту	Відображає розмір коду, який необхідно написати розробнику

4.2.3 Опис параметрів

Гірші, середні і кращі значення параметрів продукту, зведені у таблицю 4.3, вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту:

Таблиця 4.3 – Основні параметри продукту

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметру		
			Гірші	Середні	Кращі
Швидкість виконання	X1	оп/с	2000	11000	19000
Ресурсоємність	X2	МБ	8	16	32
Час виконання	X3	мс	100	400	800
Об'єм програмного коду	X4	строки	1000	1500	2000

Побудуємо графіки залежності бальної оцінки параметра від його основного значення за даними таблиці 4.3. Зображення графіки на рисунках 4.1 – 4.3.

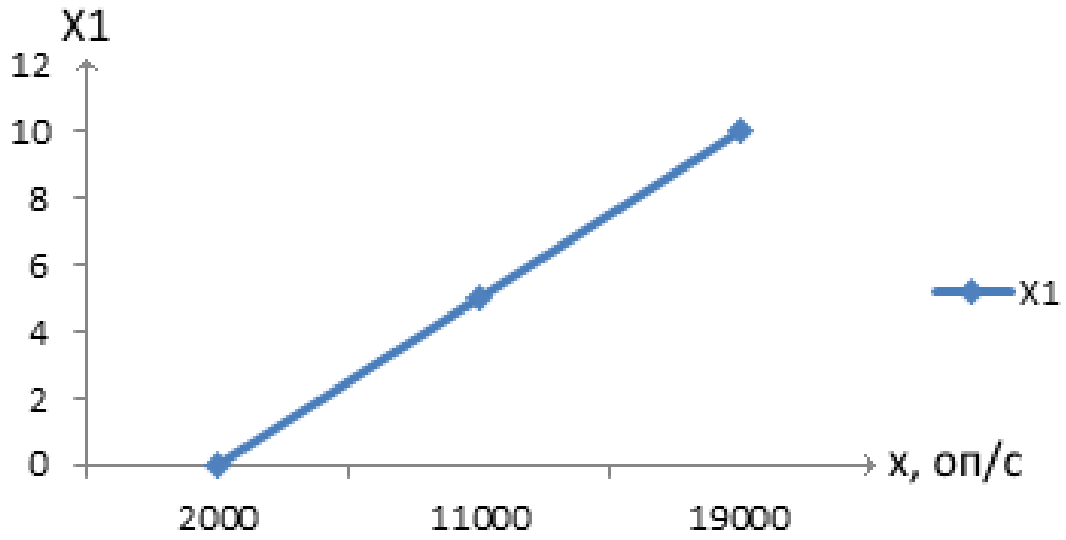

Рисунок 4.1 – Швидкість виконання

Графік залежності бальної оцінки параметра X3, що характеризує час виконання, виглядає наступним чином:

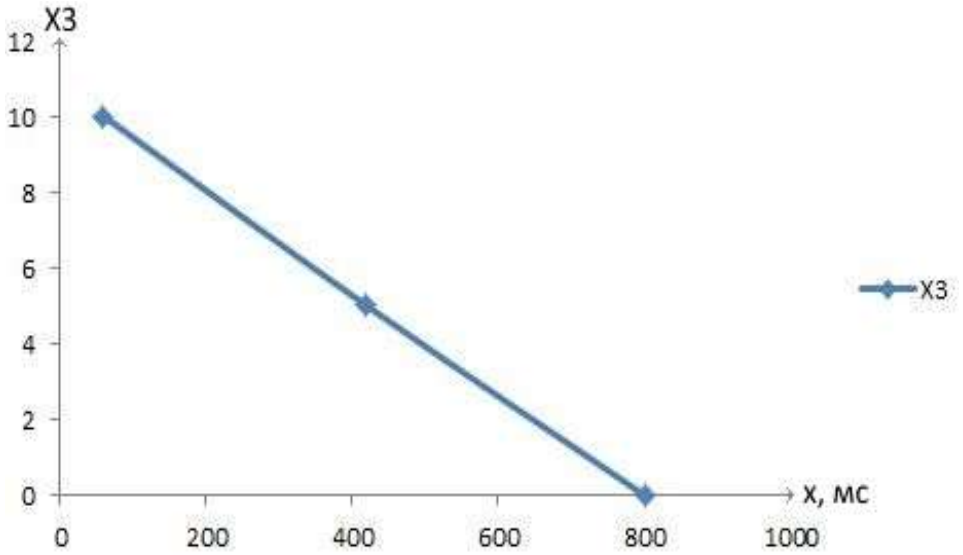

Рисунок 4.2 – Час виконання

Залежність бальної оцінки параметра X2 від кількості необхідних ресурсів демонструє наступний графік:

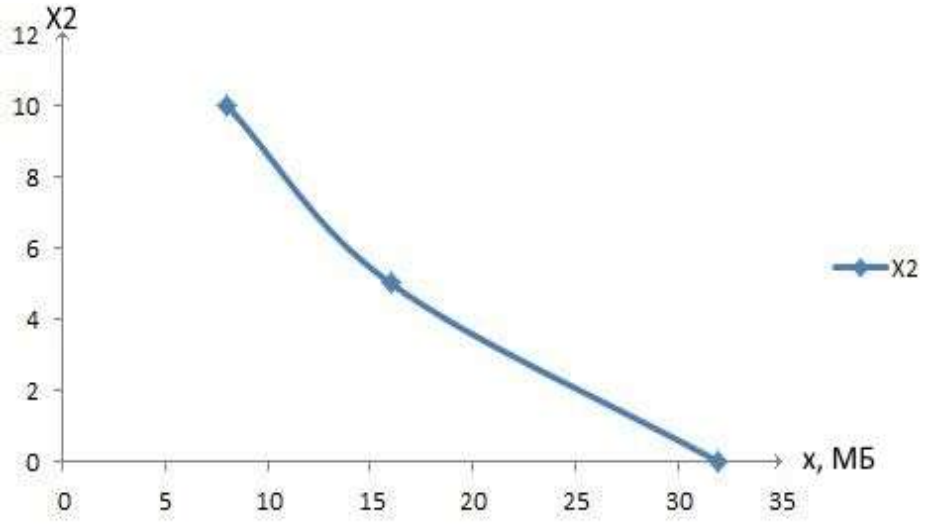

Рисунок 4.3 – Ресурсоємність

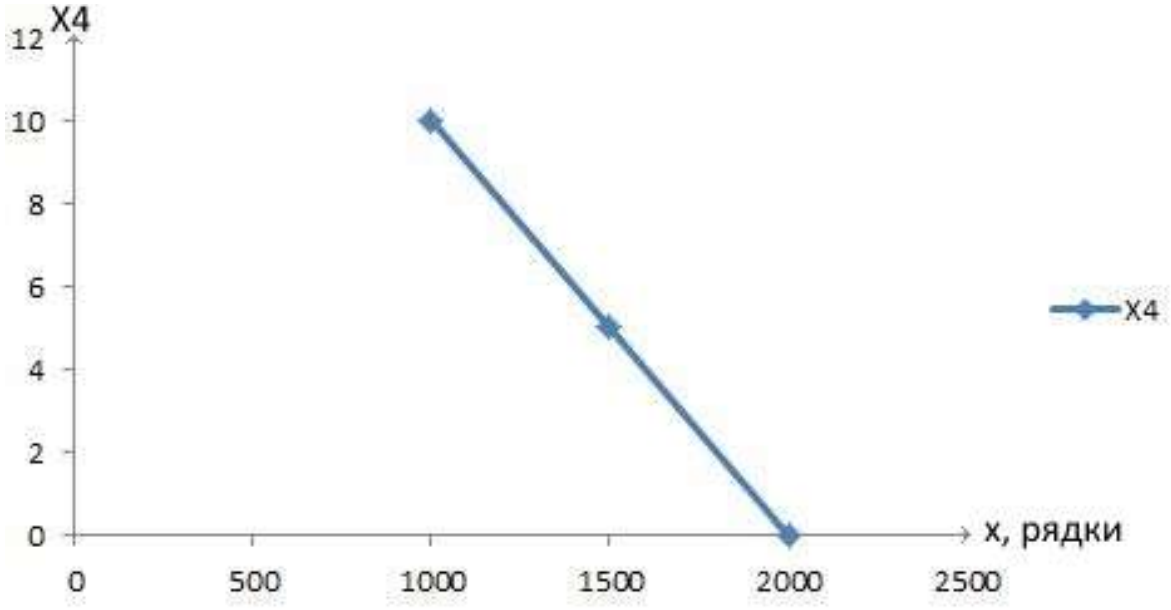

Рисунок 4.4 – Об’єм коду

4.2.4 Аналіз експертного оцінювання параметрів

Важливість кожного параметра в загальній кількості розглянутих під час оцінювання параметрів, знаходять за методом попарного порівняння.

Оцінювання проводить експертна комісія із семи осіб. Визначення коефіцієнта важливості передбачує:

- визначення рівня важливості параметра через присвоєння різних рангів кожному рівню;
- перевірити придатність експертних оцінок у подальшому використанні;
- визначити оцінки попарного пріоритету параметрів;
- обробити результати і знайти коефіцієнт важливості.

Після детального обговорення поставленої проблеми й аналізу кожний експерт оцінює рівень важливості, присвоюючи цим рівням ранг.

Результат експертного ранжування наведено в таблиці 4.4, котра демонструє зв'язок між параметром, одиницею виміру, містить ранги параметра, їх суму та відхилення

Таблиця 4.4 - Результати ранжування параметрів.

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів Ri	Відхилення	Δi2
			1	2	3	4	5	6	7			
X1	Швидкість виконання	мс	3	6	3	4	5	5	5	31	7	49
X2	Час завантаження і обробки даних	с	1	3	5	5	3	3	3	23	-1	1
X3	Ресурсоємність	МБ	2	2	2	3	3	2	1	15	-9	81
X4	Об’єм програмного коду створюваного додатку	строки	4	4	5	5	2	5	2	27	3	9
Разом			10	15	15	17	13	15	11	96	0	140

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

=96, де *N* – число експертів, *n* – кількість параметрів;

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2}$$

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 140$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 420{,}75}{7^2(5^3 - 5)} = 2{,}143 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.5 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	=	>	=	<	=	<	<	>	1,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	>	>	>	>	>	>	>	<	0,5
X2 і X3	<	<	<	<	<	<	<	<	1,5
X2 і X4	>	>	>	>	>	>	>	>	0,5
X3 і X4	>	>	>	>	>	>	>	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.6 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	1,5	0,5	0,5	3,5	0,241	22,25	0,216	100	0,215
X2	1,5	1,0	0,5	1,5	4,5	0,281	27,25	0,282	124,25	0,283
X3	1,5	1,5	1,0	1,5	4,0	0,344	34,25	0,347	156	0,348
X4	0,5	0,5	0,5	1,0	2,5	0,156	14,25	0,155	64,75	0,154
Всього:					14,5	1	54,7	1	213,6	1

Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2*(об'єм пам'яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 1000 мс або варіанту б) 80мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}$$

де n – кількість параметрів;$K_{вi}$– коефіцієнт вагомості i–го параметра; B_i – оцінка i–го параметра в балах.

Таблиця 4.7 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	100	4	0,241	0,964
F2(X2)	А	10	3,1	0,312	0,967
F3(X3,X4)	А	3	2,8	0,275	0,77
	Б	7000	1	0,172	0,172

За даними з таблиці 4.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$K_{K1} = 0,964+0,967+0,77 = 2,701$

КК2 = 0,774 + 0,962 + 0,172 = 2,103

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_P – трудомісткість розробки ПП;$K_П$ – поправочний коефіцієнт; $K_{СК}$ – коефіцієнт на складність вхідної інформації; K_M – коефіцієнт рівня мови програмування; $K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм; $K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =45 людино-днів. Поправочний коефіцієнт, який враховує вид

нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$Т_1$ = 45·1.7·0.8 = 61.2 людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =10 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ = 0.8:

$Т_2$ = 20 · 0.9 · 0.8 = 14.4 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$Т_I$ = (61.2+ 14.4+ 2.4) · 8 = 624 людино-годин;

$Т_{II}$ = (61.2+14.4+8.16) · 8 = 670.08 людино-годин;

Найбільш високу трудомісткість має варіант I.

В розробці бере участь два програмісти з окладом 9000 грн., один фінансовий аналітик з окладом 12000 грн. Визначимо зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників; T_m – кількість робочих днів тиждень; t – кількість робочих годин в день.

$$СЧ = \frac{9000 + 9000 + 12000}{6 \cdot 7 \cdot 8} = 89 \text{ грн}$$

Тоді, розрахуємо заробітну плату за формулою

$$С_{ЗП} = С_Ч \cdot Т_i \cdot К_Д,$$

де $С_ч$– величина погодинної оплати праці програміста;$Т_i$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 89 \cdot 624 \cdot 1.2 = 66643.2$ грн.

II. $С_{ЗП} = 89 \cdot 670.08 \cdot 1.2 = 71564.54$ грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

I. $С_{ВІД} = СЗП \cdot 0,22 = 66643.2 \cdot 0,22 = 14661.5$ грн.

II. $С_{ВІД} = СЗП \cdot 0,22 = 71564.54 \cdot 0,22 = 15744.2$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 9000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 9000 \cdot 0,2 = 21600$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 21600 \cdot (1 + 0.2) = 25920$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 8000 грн.

$С_А = КТМ \cdot КА \cdot ЦПР = 1.15 \cdot 0.25 \cdot 8000 = 2300$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = КТМ \cdot ЦПР \cdot КР = 1.15 \cdot 8000 \cdot 0.05 = 560$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 7 \cdot 0.7 = 1161.3$ годин, де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних

та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1161.3 \cdot 0,6 \cdot 0,4 \cdot 2,0218 = 563.5$ грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 8000 \cdot 0,67 = 5360$ грн.

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$С_{ЕКС} = 20160 + 4435.20 + 2875 + 575 + 523.83 + 6700 = 35269.03$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 35269.03 / 1706,4 = 20,67$ грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М\text{-}Г} \cdot Т$$

I. СМ = 36.64 * 624= 22863.36 грн.;

II. СМ = 36.64 * 670.08 = 24551.73 грн.;

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67$$

I. СН = 66643.2 * 0,67= 44650.94 грн.;

II. СН = 71564.54 * 0,67 = 47948.24 грн.;

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$$

I. $С_{ПП} = = 66643.2 + 22863.36 + 44650.94 = 1134157.5$грн.;

II. СПП = 71564.54+ 24551.73 + 47948.24 = 144064.5 грн.;

Тепер наявні дані для вибору кращого варіанта ПП техніко-економічного рівня.

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$К_{ТЕР1} = 2,701 / 1134157.5 = 0,234 \cdot 10^{-5}$;

$К_{ТЕР2} = 2,103 / 144064.5 = 0,238 \cdot 10^{-5}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 0,238 \cdot 10^{-4}$.

4.5 Висновок

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного

комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0,134 \cdot 10^{-4}$.

Цей варіант реалізації програмного продукту має такі параметри:

- тип архітектури клієнт-сервер – спільна обробка даних;;
- стек технології для серверної частини – на основі модулів Spring Framework;
- вибір JavaScript фреймворків для клієнтської частини – AngularJS

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, базовий функціонал і збалансовану швидкодію.

ВИСНОВКИ

В даній роботі було проведено огляд існуючих архітектур типу клієнт-сервер та багатошарових архітектур додатків. Огляд їх переваг та недоліків дав підстави обрати клієнт-серверну архітектуру з «товстим» клієнтом для реалізації програми для тестування учнів середніх класів.

У роботі обґрунтовані причини широкого застосування багатошарових архітектур серед розробників та постачальників програмних додатків. Використання багатошарової архітектури додатку виявилося більш доцільним, ніж проаналізовані двохшарова та трьохшарова і дана архітектура була обрана як каркас майбутньої програми. У роботі проведено ґрунтовний опис та аналіз як з теоретичної точки зору, так і з точки зору легкості застосування та рівня необхідних знань для використання популярних типів взаємодії в клієнт-серверних оточеннях, таких як, REST, SOAP, RPC з подальшим вибором архітектурного підходу REST, як основи клієнт-серверної взаємодії.

Огляд існуючих систем для організації процесу електронного навчання показав, що наявні на ринку системи часто мають складний, з точки зору недосвіченого у користуванні персональним комп’ютером користувача, графічний інтерфейс, складні в розгортуванні та потребують великої кількості системних ресурсів. Неможливість або надмірна складність використовувати більшість з проаналізованих систем в локальній мережі доповнює список вагомих аргументів, котрі підтверджують доцільність створення нового програмного рішення для задоволення потреб навчальних закладів, де вчаться діти середнього віку.

Впевнившись у потребі створення додатку та обравши архітектурну модель, котрій цей додаток повинен відповідати, було проведено аналіз двох найпопулярніших інструментів для створення багатошарових систем.

В ході порівняння специфікації Java EE, створеної для полегшення організації роботи великого бізнесу та фреймворка Spring Framework, котрий надає аналогічний функціонал, але вже з реалізаціями специфікацій та новаторські, у порівнянні з Java EE, підходи до розробки, було обрано саме Spring Framework. Клієнтська частина створювалася з використанням фреймворка AngularJS.

В ході створення реалізації програмного продукту було показано типові класи для основних шарів архітектури, а саме шару. Шар моделі даних реалізований з використанням технології об’єктно-реляційного відображення на основі простих та водночас потужних POJO-об’єктів. Поверх даного шару створено шар репозиторіїв. В даному випадку шар контролерів використовується для обробки запитів, що надходять від клієнтської частини та формування відповіді у вигляді JSON-об’єкта. Клієнтська частина створена у вигляді веб-інтерфейсу.

У третьому розділі продемонстрований функціонал створеного додатку, показані основні моменти його роботи, як-то проходження тесту, створення тесту, відповідь на запитання, та перегляд успішності учнів по обраному тесту. Реалізація задовольняє потреби користувачів для яких вона створена, в котрий раз підтверджуючи виправданість свого існування.

Суттєвим є також той факт, що серверна частина створеної програми для тестування учнів середніх класів надає можливість розширення свого функціоналу двома основними шляхами - створення клієнтів різних видів завдяки REST, та зміну шару моделі з мінімальними зусиллями.

В рамках роботи було проведено повний функціональний аналіз продукту, на основі якого можна зробити висновок, що підбір архітектури та інструментів для її реалізації був виконаний ефективно з економічної точки зору.

ПЕРЕЛІК ПОСИЛАНЬ

1. René Chevance. Server Architectures: Multiprocessors, Clusters, Parallel Systems, Web Servers, Storage Solutions / René Chevance – Elsevier/Digital Press, 2005. – 690 p.
2. Erich Gamma. Design Patterns: Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnson et al. – Addison-Wesley Professional, 1994. – 458 p.
3. Martin Fowler. Patterns of Enterprise Application Architecture / Martin Fowler. – Addison-Wesley Professional, 2002. – 649 p.
4. Len Bass. Software Architecture in Practice / Len Bass, Paul Clements, Rick Kazman. – Addison-Wesley Professional, 2012. – 334 p.
5. Гонсалвес Э. Изучаем Java EE 7 / Э. Гонсалвес. – Appress, 2016. – 673 p.
6. Peter Herzum. Business Component Factory : A Comprehensive Overview of Component-Based Development for the Enterprise / Peter Herzum, Oliver Sims. - Wiley, 1999 – 257 p.
7. Jose Sandoval. RESTful Java Web Services / Jose Sandovel. – PACT publishing, 2009. – 328 p.
8. Layered Application, Microsoft Developers Network – Режим доступу: https://msdn.microsoft.com/en-us/library/ff650258.aspx - Дата тоступу: 24.05.2016
9. A multi-tier architecture for building RESTful Web services – Режим доступу: http://www.ibm.com/developerworks/library/wa-aj-multitier/ - Дата доступу 27.05.2016
10. Jim Webber. REST in Practice, Hypermedia and Systems Architecture / Jim Webber, Savas Parastatidis,Ian Robbinson. – O`Reilly Media, 2010 – 287 p.

11. W3C, SOAP Messages with Attachments – Режим доступу: https://www.w3.org/TR/SOAP-attachments/ - Дата доступу: 27.05.2016
12. Офіційна сторінка XML-RPC – Режим доступу: http://xmlrpc.scripting.com/ - Дата доступу: 30.05.2016
13. Elearning 101 concepts, trends, applications – Режим доступу: http://www.talentlms.com/elearning/ - Дата доступу: 4.06.2016
14. Understanding the Three-Tier Architecture. Oracle Documentation – Режим доступу: http://docs.oracle.com/undtldev010 – Дата доступу: 15.05.2016.
15. Офіційний сайт Spring Framework – Режим доступу: https://spring.io/ - Дата доступу: 14.06.2016