СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра програмування та математики

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)
спеціалізація „Інженерія програмного забезпечення”

на тему „Реалізація клієнт-серверної архітектури для тестування учнів середніх класів”

Виконав: студент групи ІПЗ-16д _____ Д.С. Фьодоров
(підпис) (ініціали і прізвище)

Керівник _____ Ю.Г. Ковальов
(підпис) (ініціали і прізвище)

Завідувач кафедри _____ В.О. Лифар
(підпис) (ініціали і прізвище)

Рецензент_____

Сєвєродонецьк – 2020

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра програмування та математики

Освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)
спеціалізація „Інженерія програмного забезпечення”
(назва спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПМ,
д.т.н., доцент _____ Лифар В.О.
“____” _____2020 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

Фьодоров Д. С.

1. Тема роботи Реалізація клієнт-серверної архітектури для тестування учнів середніх класів
Керівник роботи: к.ф.-м.н., Ковальов Юрій Григорійович
затверджений наказом університету від “___”_____20__року №___
2. Строк подання студентом роботи 20 травня 2020 р.
3. Вихідні дані до роботи Проведено дослідження архітектур типу клієнт-сервер. Розроблено серверна частина веб-додатку на основі фреймворку Spring Framework та клієнтська частина- на базі фреймворку VueJS для тестування учнів середніх класів.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Зробити огляд існуючих архітектур.
    2. Огляд існуючих програмних рішень та технологій
    3. Створити програмної реалізації додатку
5. Перелік графічного матеріалу немає

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання__30 березеня 2020 року. _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Одержання завдання на виконання роботи	30.03.20	
2	Укладання і погодження з керівником плану і етапів виконання роботи	06.04.20	
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	13.04.20	
4	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху	20.04.20	
5	Укладання та тестування програмного продукту	27.04.20	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	04.05.20	
7	Здача готової пояснювальної записки на кафедру	12.05.20	
8	Укладання доповіді і презентації	30.05.20	

Студент _____ ___Д.С. Фьодоров_____
(підпис) (ініціали і прізвище)

Керівник роботи _____ ____Ю Г. Ковальов_____
(підпис) (ініціали і прізвище)

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

дипломної роботи студента гр. ІПЗ-16Д Фьодоров Д.С..

Науковий керівник

Доцент, к.ф.-м..н. _____ Ковальов Ю.Г.

Оцінка наукового керівника: _____

Рецензент доцент. каф. ПМ СНУ ім.В.Даля Захожай О.І.
місто роботи, посада, ПІБ

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту:
_____

Голова ЕК

_____ Лифар В.О.
підпис

РЕФЕРАТ

Текст – 58 стор., рис. – 23, табл. – 4, літературних джерел – 15

Цифровізація усіх сфер життя не оминула і сферу освіти. Покращення якості навчаня сильно пов’язано з інформаційними технологіями, які полегшують навчання, зокрема, дозволяють навчатися дистанційно, та поліпшують якість навчання, прискорюючи росповсюдження інфорації. Тому, задача автоматичного контролю знань за допомогою програмного продукуту є актуальною задачею, оскілки дає безпристрастність оцінювання та прискорює проведення контролю здобутих знань.

Метою даної роботи є розробка програми для проведення тестування онлайн - перевірки знань учнів середніх класів.

ВЕБ-ДОДАТОК, КЛІЄНТ-СЕРВЕР, БАГАТОШАРОВА АРХІТЕКТУРА, REST, VUEJS

ЗМІСТ

ВСТУП

Актуальність. Застосування інформаційних технологій в усіх сферах життя не оминуло й сферу освіти. Покращення якості навчаня сильно пов’язано з інформаційними технологіями, які полегшують навчання, зокрема, дозволяють навчатися дистанційно, та поліпшують якість навчання, прискорюючи росповсюдження інфорації. Тому, задача контролю знань за допомогою різноманітних комп’ютерних програм є актуальною.

Зміна звичайного тестування комп’ютерним, розширює можливості контролю та оцінювання рівня знань учнів, оскільки надає результат швидко та без суб’єктивних міркувань. Проведення тестування за комп’ютерами, зокрема, через інтернет або локальну мережу зекономе час та дає можливість протестувати більшу кількість людей за обмежений проміжок часу. Онлайн тестування також дозволяє проводити контроль знань серед учнів, що навчаються дома. Автоматичне оцінювання та обробка результатів контроля знань підвищіть ефективність роботи викладача.

Об'єкт дослідження – програмне забезпечення для проведення тестування.

Предмет дослідження – веб-додаток для проведення тестування серед учнів середніх класів

Мета дослідження: розробка веб-додатку для тестування учнів середніх.

Задачі дослідження:

1. дослідити та проаналізувати сайти, що надають можливість проходження тестів онлайн.

2. спираючись на проведений аналіз розробити специфікацію вимог до розроблюваного програмного продукту;

3. розробити архітектуру програми та спроектувати структуру бази даних;

4. зробити вибір мов програмування та технологій для програмної реалізації описаного продукту;

1. АНАЛІЗ АРХІТЕКТУР ВЕБ-ДОДАТКІВ

1.1 Визначення

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервер – програма, що надає деякі послуги іншим програмам (клієнтам). Зв'язок між клієнтом і сервером зазвичай здійснюється за допомогою передачі повідомлень, часто через мережу, і використовує певний протокол для кодування запитів клієнта і відповідей сервера. Серверні програми можуть бути встановлені як на серверному, так і на персональному комп'ютері, щоразу вони забезпечують виконання певних служб (наприклад, сервер баз даних чи веб-сервер).

Клієнт – апаратний або програмний компонент обчислювальної системи, який надсилає запити серверу. Програма-клієнт взаємодіє з сервером, використовуючи певний протокол. Вона може запитувати з сервера будь-які дані, маніпулювати даними безпосередньо на сервері, запускати на сервері нові процеси і т. п. Отримані від сервера дані клієнтська програма може надавати користувачеві або використовувати як-небудь інакше, в залежності від призначення програми. Програма-клієнт і програма-сервер можуть працювати як на одному і тому ж комп'ютері, так і на різних. У другому випадку для обміну інформацією між ними використовується мережеве з'єднання.

1.2 Опис моделі клієнт-сервер

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером, схематичне зображення цього показано на рисунку 1.1.

Логічно можна відокремити три рівні операцій:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів — клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

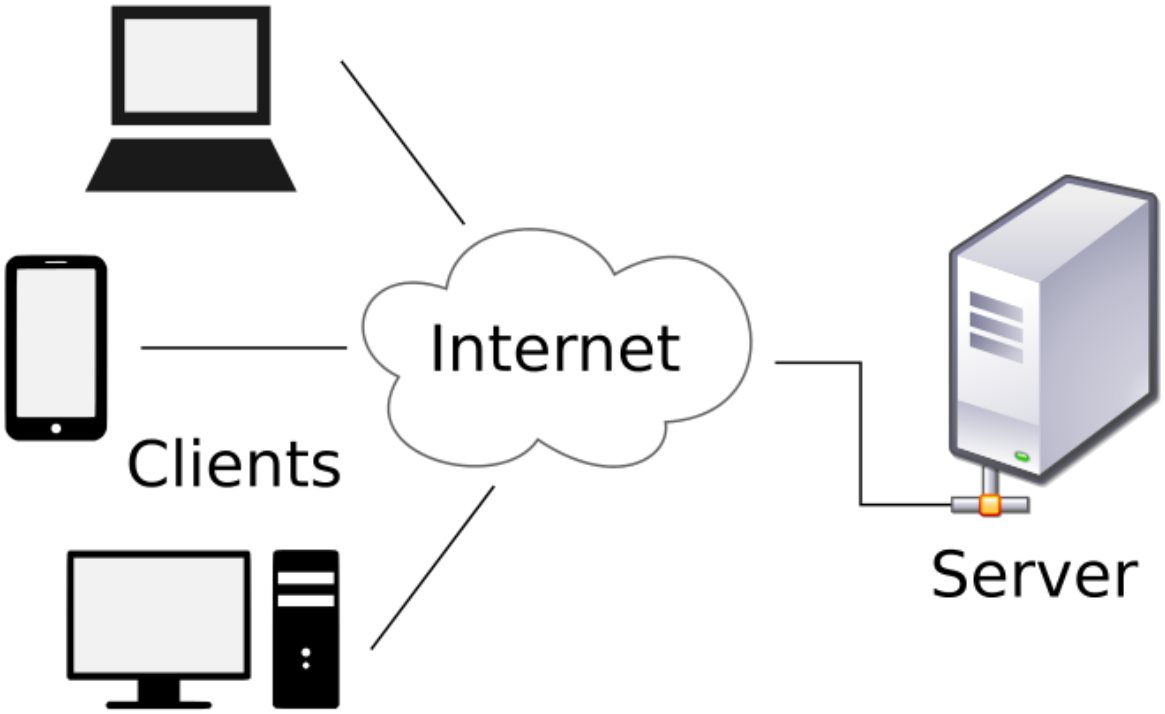

Рисунок 1.1 –Представлення архітектури клієнт-сервер

1.3 Типи архітектур

Обробка даних на хості – ця схема насправді не є додатком клієнт-сервер, але відноситься до оточення mainframe-а, у цьому випадку майже вся обробка даних проводиться на головній обчислювальної машині. Найчастіше в подібній обчислювальній архітектурі інтерфейс користувача має вигляд примітивного терміналу.

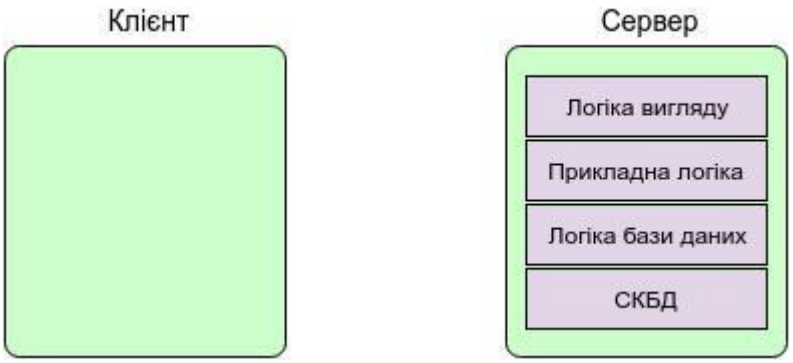

Рисунок 1.2 – Обробка даних на сервері

Розподілена обробка даних - найпростішим типом схеми клієнт-сервер є схема, у якій на стороні клієнта обробка даних зводиться дло надання графічного інтерфейсу користувача, а обробка даних здійснюється на стороні сервера. Ця схема дозволяє дещо зменшити навантаження на сервер, найчастіше коли інтерфейс користувача дуже складний та потребує багато ресурсів.

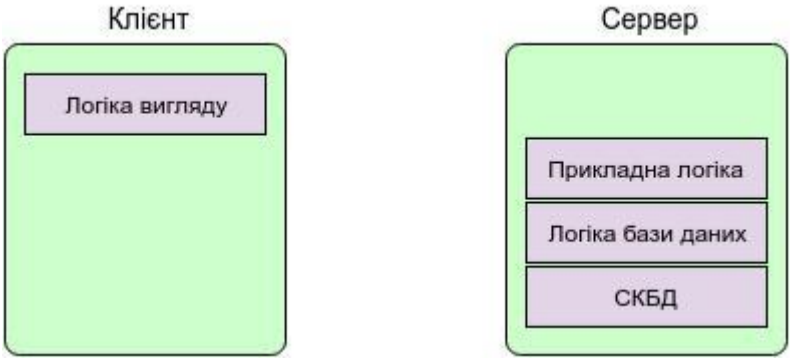

Рисунок 1.3 – Обробка даних на базі сервера

Обробка даних на базі клієнта - майже вся обробка даних проводиться на боці клієнта, за деякими виключенням, такими як: перевірка цілісності даних, логіка обслуговування бази даних, що доречніше проводити на стороні сервера. На даний час ця схема є найбільш поширеною реалізацією архітектури клієнт-сервер. Оскільки вона дозволяє користувачам працювати з програмами, що відповідають їх локальним потребам.

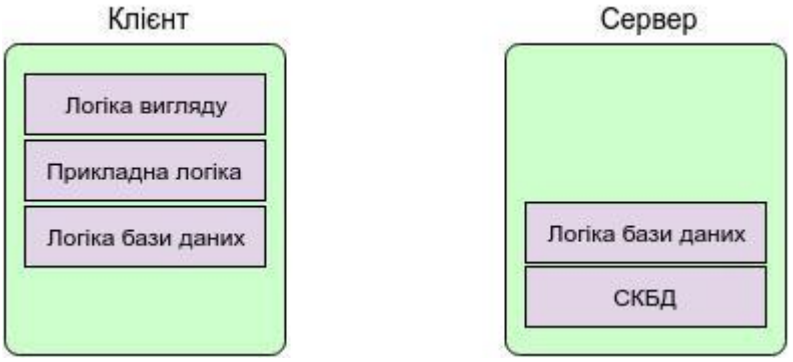

Рисунок 1.4 – Обробка даних на стороні клієнта

Спільна обробка даних – у даному випадку, обробка даних розподілена таким чином, щоб були задіяні сильні сторони клієнта та сервера, а також самого факту розподілу даних.

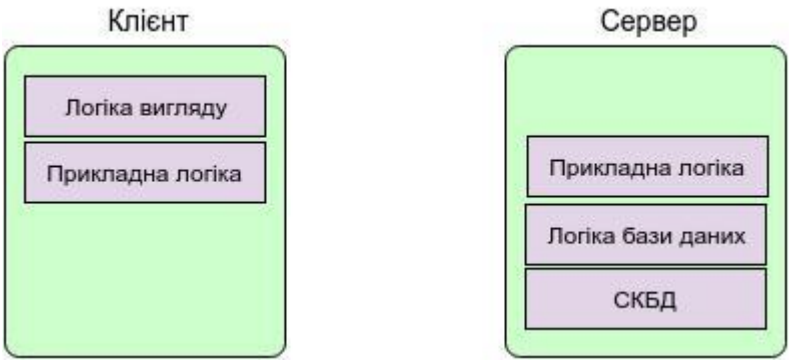

Рисунок 1.5 – Спільна обробка даних

Робити зміни в такій схемі складно при встановленні та 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 сторони сервера

Дволанкова архітектура "клієнт-сервер". Дана архітектура припускає, що СУБД знаходиться на сервері і тільки вона має доступ до файлів БД.

На клієнтських комп'ютерах працюють користувальницькі додатки і клієнтські компоненти СУБД, які здійснюють взаємодію з сервером. Від клієнта на сервер приходять запити, які обробляються СУБД, і результат відправляється на клієнтський комп'ютер. У порівнянні з файл-серверної архітектурою, в цьому випадку мінімізується мережевий трафік: по мережі передаються тільки затребувані дані.

Централізована робота з файлами БД дозволяє більш ефективно вирішувати питання, пов'язані зі спільним доступом до даних і з забезпеченням безпеки. У зв'язку з тим, що СУБД працює на сервері, знижуються і вимоги до обладнання на стороні клієнта. Прикладна програма може бути написана на будь-якій мові, який підтримує взаємодію з використовуваної СУБД через наявні клієнтські компоненти: можуть використовуватися такі технології, як ODBC, ADO або ADO.Net, JDBC і ін.

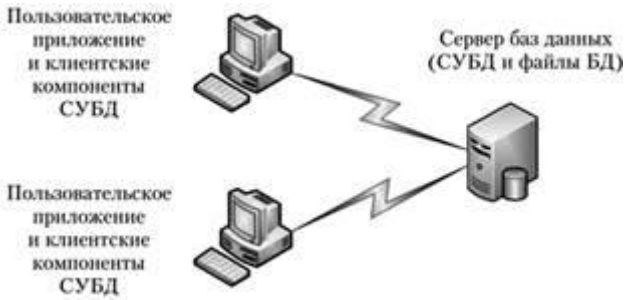

Рис 1.6.. **Дволанква архітектура "клієнт-сервер"**

Двошарова архітектура широко використовується для створення інформаційних систем, а й у неї є ряд недоліків. По-перше, це "логіка" додатка, винесена на сторону клієнта. При великій кількості і географічної віддаленості клієнтських робочих місць виникають проблеми з оновленням і усуненням помилок. Друга проблема - кожен клієнт незалежно від інших може в довільний момент відкрити з'єднання з СУБД. Сервер підтримує відкриті з'єднання з усіма активними клієнтами, навіть якщо ніякої роботи немає. При великому числі клієнтів це може негативно впливати на продуктивність сервера БД.

Триланкового архітектура. Виправити недоліки двухзвенной архітектури дозволяє трехзвенная, також звана трирівневої (рис. 2.4). Дана архітектура припускає наявність додаткового сервера додатків, який проводить попередню обробку запитів клієнтів, формує запити до сервера БД і обробляє отримані результати перед відправкою їх клієнту.

У триланкової архітектурі велика частина логіки додатка перенесена з клієнта на сервер, і завдання клієнтського додатка зводяться, в основному, до реалізації призначеного для користувача інтерфейсу і поданням результатів. З'являється можливість централізовано вносити зміни в алгоритми бізнес-логіки, реалізовані в ПЗ сервера додатків. Також в цій архітектурі тільки сервер додатків може підключатися до сервера БД. Це знімає проблему підтримки невикористовуваних з'єднань і більш переважно з точки зору безпеки. Недоліком багаторівневої архітектури "клієнт-сервер" є складність розробки подібних рішень.

Архітектура інтернет / інтранет-рішень. На закінчення необхідно згадати про архітектуру інтернет / інтранет-додатків (рис. 2.5). У цьому випадку обов'язковим компонентом серверної частини є вебсервер, на якому виконується веб-додаток, що звертається до СУБД для доступу до БД. У ролі універсального клієнтського додатка виступає інтернет-браузер на пристрої клієнта (в даному випадку це може бути персональний комп'ютер, планшетний комп'ютер або якесь мобільний пристрій).

Рис. 2.4. **триланкового архітектура**

Рис. 2.5. **Архітектура інтернет / інтранет-рішень**

У невеликих рішеннях веб-сервер може знаходитися на одному фізичному комп'ютері з СУБД. У великих системах, що зазнають великі навантаження, задіюється безліч веб-серверів і серверів БД.

Переваги і недоліки подібної архітектури пов'язані з використанням браузера в якості універсального клієнта. З одного боку, відпадає необхідність в розробці, розповсюдженні та підтримки клієнтського додатку. З'являється універсальність і часткова незалежність від клієнтської платформи. З іншого боку, в разі обмеженого числа користувачів додатка за допомогою спеціально розробленого клієнтського ПЗ можна домогтися більшої функціональності в порівнянні з використанням веб-технологій.

Багаторівневі архітектури клієнт-сервер

Такі архітектури більш розумно розподіляють модулі обробки даних, які в цьому випадку виконуються на одному або декількох окремих серверах. Ці програмні модулі виконують функції сервера для інтерфейсів з користувачами і клієнта - для серверів баз даних. Крім того, різні сервери додатків можуть взаємодіяти між собою для більш точного поділу системи на функціональні блоки, які виконують певні ролі. Наприклад, можна виділити сервер управління персоналом, який буде виконувати всі необхідні для управління персоналом функції. Зв'язавши з ним окрему базу даних, можна приховати від користувачів всі деталі реалізації цього сервера, дозволивши їм звертатися тільки до його загальнодоступним функцій. Крім того, таку систему дуже просто адаптувати до Web, оскільки простіше розробити html-форми для доступу користувачів до певних функцій бази даних, ніж до всіх даних.

У трирівневої архітектурі "тонкий" клієнт не перевантажений функціями обробки даних, а виконує свою основну роль системи подання інформації, що надходить з сервера додатків. Такий інтерфейс можна реалізувати за допомогою стандартних засобів Web-технології - браузера, CGI і Java. Це зменшує обсяг даних, переданих між клієнтом і сервером додатків, що дозволяє

підключати клієнтські комп'ютери навіть по повільним лініях типу телефонних каналів. Крім того, клієнтська частина може бути настільки простий, що в більшості випадків її реалізують за допомогою універсального браузера. Але якщо міняти її все-таки доведеться, то цю процедуру можна здійснити швидко і безболісно. Трирівнева архітектура клієнт-сервер дозволяє більш точно призначати повноваження користувачів, так як вони отримують права доступу не до самої бази даних, а до певних функцій сервера додатків. Це підвищує захищеність системи (у порівнянні з звичайно архітектурою) не тільки від навмисного нападу, а й від помилкових дій персоналу.

Для прикладу розглянемо систему, різні частини якої працюють на декількох віддалених один від одного серверах. Припустимо, що від розробника надійшла нова версія системи, для установки якої в дворівневої архітектури необхідно одночасно поміняти всі системні модулі. Якщо ж цього не зробити, то взаємодія старих клієнтів з новими серверами може привести до непередбачуваних наслідків, так як розробники зазвичай не розраховують на таке використання системи. У трирівневої архітектурі ситуація спрощується. Справа в тому, що помінявши сервер додатків і сервер зберігання даних (це легко зробити одночасно, так як обидва вони зазвичай знаходяться поруч), ми відразу міняємо набір доступних сервісів. Таким чином, ймовірність помилки через невідповідність версій серверної і клієнтської частин різко скорочується. Якщо в новій версії будь-якої сервіс зник, то елементи інтерфейсу, які обслуговували його в старій системі, просто не будуть працювати. Якщо ж змінився алгоритм роботи сервісу, то він буде коректно працювати навіть зі старим інтерфейсом.

Багаторівневі клієнт-серверні системи досить легко можна перевести на Web-технологію - для цього достатньо замінити клієнтську частину універсальним або спеціалізованим браузером, а сервер додатків доповнити Web-сервером і невеликими програмами виклику процедур сервера. Для розробки цих програм

можна використовувати як Common Gateway Interface (CGI), так і більш сучасну технологію Java.

Слід зазначити і той факт, що в трирівневої системи по каналу зв'язку між сервером додатків і базою даних передається досить багато інформації. Однак це не уповільнює обчислень, так як для зв'язку зазначених елементів можна використовувати більш швидкісні лінії. Це зажадає мінімальних витрат, оскільки обидва сервера зазвичай знаходяться в одному приміщенні. Таким чином, збільшується сумарна продуктивність системи - над одним завданням тепер працюють два різних сервера, а зв'язок між ними можна здійснювати за найбільш швидкісним лініях з мінімальними витратами коштів. Правда, виникає проблема узгодженості спільних обчислень, яку покликані вирішувати менеджери транзакцій - нові елементи багаторівневих систем.

Менеджери транзакцій

Особливістю багаторівневих архітектур є використання менеджерів транзакцій (МТ), які дозволяють одного сервера додатків одночасно обмінюватися даними з декількома серверами баз даних. Хоча сервери Oracle мають механізм виконання розподілених транзакцій, але якщо користувач зберігає частину інформації в БД Oracle, частина в БД Informix, а частина в текстових файлах, то без менеджера транзакцій не обійтися. МТ використовується для управління розподіленими різнорідними операціями і узгодження дій різних компонентів інформаційної системи. Слід зазначити, що практично будь-яке складне ПО вимагає використання менеджера транзакцій. Наприклад, банківські системи повинні здійснювати різні перетворення уявлень документів, т. Е. Працювати

1.5 Обмін даними між клієнтом та сервером: SOAP та REST

SOAP (англ. Simple Object Access Protocol) — протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, базується на форматі XML.

Спочатку SOAP призначався, в основному, для реалізації віддаленого виклику процедур (RPC), а назва була абревіатурою: Simple Object Access Protocol — простий протокол доступу до об'єктів. Зараз протокол використовується для обміну повідомленнями в форматі XML, а не тільки для виклику процедур. SOAP є розширенням мови XML-RPC.

SOAP можна використовувати з будь-яким протоколом прикладного рівня: SMTP, FTP, HTTP та інші. Проте його взаємодія з кожним із цих протоколів має свої особливості, які потрібно відзначити окремо. Найчастіше SOAP використовується разом з HTTP.

SOAP є одним зі стандартів, на яких ґрунтується технологія веб-сервісів.

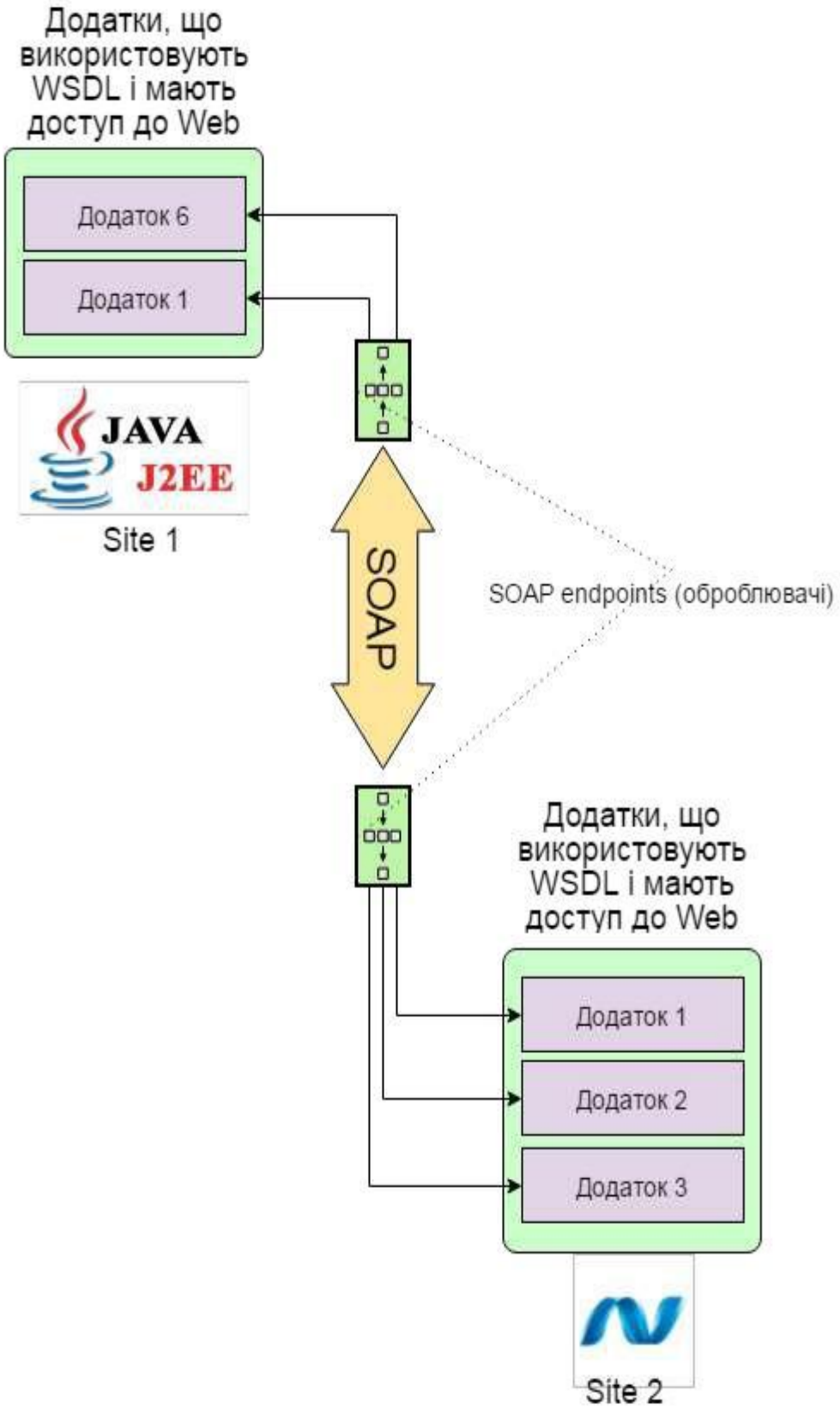

Рисунок 1.11 – Механізм SOAP

Переваги

SOAP є достатньо гнучким, що дозволяє використовувати різні транспортні протоколи. Стандартні реалізації використовують HTTP як транспортний протокол, однак також можливо використовувати JMS чи SMTP.

Позаяк модель SOAP каналів(з'єднань) прекрасно працює в контексті моделі HTTP запитів-відповідей, то можливо легко встановлювати з'єднання на базі існуючих фаєрволів чи проксі-серверів без жодних модифікацій самого SOAP-протоколу.

Недоліки

Застосування SOAP для передачі повідомлень збільшує їхній обсяг і зменшує швидкість обробки. У системах, де швидкість важлива, часто надсилають XML документи безпосередньо через HTTP як звичайні HTTP параметри або застосовують протокол REST.

Попри те, що для SOAP є стандарт, різні програми часто генерують повідомлення в несумісному форматі. Наприклад, запит згенерований Apache Axis[en]-клієнтом, не завжди розпізнається сервером WebLogic.

REST (скор. англ. Representational State Transfer, «передача репрезентативного стану») — підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP. В основі REST закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0.

Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабовність системи і дозволяє їй еволюціонувати з новими вимогами.

Антиподом REST є підхід, заснований на виклику віддалених процедур (Remote Procedure Call, RPC). Підхід RPC дозволяє використовувати невелику кількість мережевих ресурсів з великою кількістю методів і складним протоколом. При підході REST кількість методів і складність протоколу суворо обмежені, що призводить до того, що кількість окремих ресурсів має бути великою.

REST — це архітектурний стиль для розподілених гіпертекстових систем
Архітектурні обмеження
REST, як і кожен архітектурний стиль відповідає ряду архітектурних обмежень (англ. architectural constraints). Це гібридний стиль який успадковує обмеження з інших архітектурних стилів.[1]

Клієнт-сервер
Перша архітектура від якої він успадковує обмеження — це клієнт-серверна архітектура. Її обмеження вимагає розділення відповідальності[en] між компонентами, які займаються зберіганням та оновленням даних (сервером), і тими компонентами, які займаються відображенням даних на інтерфейсі користувача та реагування на дії з цим інтерфейсом (клієнтом). Таке розділення дозволяє компонентам еволюціонувати незалежно.

Відсутність стану
Наступним обмеженням є те, що взаємодії між сервером та клієнтом не мають стану, тобто кожен запит містить всю необхідну інформацію для його обробки, і не покладається на те, що сервер знає щось з попереднього запиту.

Обмеження "відсутність стану" не означає, що архітектура REST дозволяє будувати лише системи, в яких немає стану[en]. Відсутність стану означає, що сервер не знає про стан клієнта і не повинен запам'ятовувати послідовність

здійснених до нього запитів, тому що кожен з них є незалежним. Коли клієнт, наприклад, запитує головну сторінку сайту, сервер відповідає на запитання і забуває про клієнта. Клієнт може залишити сторінку відкритою протягом кількох років, перш ніж натиснути посилання, і тоді сервер відповість на інший запит. Тим часом сервер може відповідати на запити інших клієнтів, або нічого не робити — для клієнта це не має значення.

Таким чином, наприклад дані про стан сесії (користувача, який автентифікувався) зберігаються на клієнті і передаються з кожним запитом. Це покращує масштабовність, бо сервер після закінчення обробки запиту може звільнити всі ресурси, задіяні для цієї операції, без жодного ризику втратити цінну інформацію. Також спрощується моніторинг і зневадження, бо для того аби розібратись у тому, що відбувається в певному запиті, досить подивитись лише на цей запит. Збільшується надійність, бо помилка в одному запиті не зачіпає інші.[2]

Мінусом цього обмеження є те, що знижується продуктивність через те, що в кожен запит тепер доводиться додавати дані сесії з клієнта. Також збереження стану на різних клієнтах важче підтримувати, бо реалізації клієнтів можуть відрізнятись, тоді як середовище сервера повністю під контролем розробника.[3]

Кешування

Додатковим обмеженням стилю REST є те, що системи, написані в цьому стилі, повинні підтримувати кешування, тобто дані, які передаються сервером, повинні містити інформацію про те, чи можна їх кешувати, і якщо можна, то як довго. Це дозволяє збільшувати продуктивність, уникаючи зайвих запитів, але також зменшує надійність системи через те, що дані в кеші можуть застаріти.

Рання архітектура веб, створена Тімом Бернерсом-Лі відповідала цим трьом обмеженням — клієнт-сервер без стану з підтримкою кешування.[4] Проте, стиль REST додає ще додаткові обмеження.

Однорідний інтерфейс

Всі компоненти в архітектурі REST підтримують однорідний інтерфейс. Це зменшує зв'язність між компонентами і сервісами які вони надають і дозволяє нескладно змінювати компоненти при потребі.[5] Це досягається кількома точнішими обмеженнями:

ідентифікація ресурсів: URI запит повинен точно визначати ресурс, що очікується та якого формату повинна бути відповідь. У відповідь на запит клієнту надається представлення ресурсу. Ресурси концептуально відділені від представлення. Наприклад, серевер може надсилати дані із бази даних в форматі HTML, XML або JSON, жоден з яких не являється типом даних, що зберігаються всередині сервера.

маніпуляція ресурсами через представлення: якщо клієнт має представлення ресурсу, включаючи метадані, він має достатньо інформації, щоб модифікувати або видаляти цей ресурс.

самоописові повідомлення: кожне повідомлення має включати достатньо інформації, щоб обробити повідомлення. Наприклад, обробник (parser), що використовується для розшифровки даних, може бути вказаний в списку MIME типів.

гіпермедіа як рушій стану застосунку (HATEOAS): Клієнт, що має доступ до REST сервісу, повинен отримати всі наявні сервіси та методи за допомогою гіперпосилань.

Шари абстракції. Наступним обмеженням для REST є поділ на шари абстракції. Кожен компонент потрапляє в якийсь шар, і спілкується лише з компонентами в шарі під ним або в шарі над ним. Обмежнення знання системи одним шаром зменшує складність компонентів.[6]

Запитування коду. Останнім архітектурним обмеженням в REST є те що клієнти повинні дозволяти розширювати свою функціональність дозволяючи завантаження додаткового коду (code on demand[en]) в формі аплетів чи скриптів. Це спрощує клієнти, дозволяючи не реалізовувати всі необхідні функції попередньо. Щоправда це необов'язкове обмеження, і якщо воно не дає переваг для конкретного застосування, то його не обов'язково реалізовувати. Наприклад, дозвіл завантаження стороннього коду може бути не бажаним з точки зору безпеки.[7]

Типове використання HTTP методів:

HTTP методи

URL GET PUT PATCH POST DELETE

Колекція https://api.example.com/resources/ Повертає URI та можливо інші деталі ресурсів колекції. Замінює одну колекцію іншою. Зазвичай не використовують Створює новий ресурс в колекції. URI для нового ресурсу застосунок призначає автоматично та зазвичай повертає у відповіді. Видаляє всю колекцію.

Ресурс https://api.example.com/resources/item17 Повертає представлення ресурсу колекції. Замінює певний ресурс колекції, і якщо він не існує, створює його.

Оновлює певний ресурс колекції. Зазвичай не використовують Видаляє певний ресурс колекції.

Хоча ресурс може бути яким завгодно, дії, які можна виконувати над ресурсом, визначаються повідомленнями, які визначено стандартним протоколом. В системі WWW цей протокол — HTTP, але існують REST-архітектури на основі й інших протоколів.

Стандарт HTTP визначає 8 типів повідомлень.

Найчастіше використовують 4 з них:

GET — отримати представлення ресурсу
DELETE — знищити ресурс
POST — створити новий ресурс на місці даного використавши передане представлення
PUT — замінити стан поточного ресурсу станом, що описується переданим представленням
Ці використовуються, щоб дослідити API:

HEAD — отримати заголовки, які б відсилались разом з представленням цього ресурсу, але не саме представлення.
OPTIONS — визначити список методів, на які цей ресурс відповідає.
Інші два методи, CONNECT та TRACE, використовуються лише для HTTP-проксі.

Існує також дев'ятий метод, щоправда, описаний не в HTTP, а в додатку RFC 5789:

PATCH — змінити лише частину цього ресурсу на основі даного представлення. Якщо якась частина ресурсу не згадується в переданому представленні — не чіпати її. Це знижує кількість інформації, яку потрібно передавати.

Ще два методи описуються в пропозиції до стандарту «Internet-Draft „snell-link-method“»:

LINK — прив'язати певний ресурс до цього
UNLINK — відв'язати ресурс від цього

Методи GET, PUT та DELETE — ідемпотентні, що означає, що незалежно від того, скільки разів ви виконаєте операцію, яку вони просять, — ви отримаєте той самий результат. Звісно, спершу DELETE поверне 204 No Content, а потім 404 Not Found, але ресурсу не буде, що після одного видалення, що після десяти. Ідемпотентність є дуже важливою в мережі, де ви не знаєте, чи досяг запит успіху, і, не отримавши відповіді, надсилаєте його ще раз. POST — не ідемпотентний, тобто, відправивши POST на створення повідомлення кілька разів, ви отримаєте кілька повідомлень.

Висновки

В цьому розділі було розглянуто основні типи клієнт-серверних архітектур, типи архітектур клієнт-серверних додатків, їх переваги та недоліки.
Проаналізувавши переваги та недоліки розгялнутих архітектур, зроблено висновок, що для розробки веб-додатка для проведення тестування доцільно обрата триланкову архітектуру сервера. Також було обрано в якості підходу до архітектури мережевих протоколів - підхід REST, як один з найпоширеніших у веб-додатках.

2 АНАЛІЗ ПРОГРАМНИХ РІШЕНЬ ТА ТЕХНОЛОГІЙ

2.1 Аналіз програмних рішень систем електронного навчання

MOODLE

Модульна об'єктно орієнтована навчальна система (MOODLE) є пакетом програмного забезпечення для створення курсів дистанційного навчання. Цей проект був створений для підтримки та досліджень теорії «socialconstructionist framework of education» в Curtin University of Technology, Австралія. До переваг слід віднести те, що платформа MOODLE - це повністю відкритий і вільно поширюваний проект. Основними особливостями MOODLE є - система спроектована з акцентом на взаємодію між студентами, учнями; може використовуватися для дистанційного та очного навчання; має простий і ефективний web-інтерфейс; дизайн має модульну структуру і легко модифікується; багатий набір модулів-складових для курсів; можливе налаштування e-mail (розсилки новин, форумів, оцінок та коментарів викладачів). Прикладом застосування платформи MOODLE для організації навчання школярів є система E-school (https://eschool.dn.ua/), яка об’єднала десять шкіл Донецької області для організації дистанційного навчання учнів з окупованих територій. За допомогою платформи розміщено теоретичний матеріал до навчальних дисциплін, учні мають змогу виконувати практичні роботи, спілкуватися з викладачами, отримувати оцінки, складати тести та контрольні роботи. Але платформа MOODLE – система, орієнтована на підтримку саме дистанційного навчання, взаємодії між учнем та вчителем, без урахування потреб розв’язання завдань науково-методичної діяльності вчителів та кадрових питань.

спільною назвою «соціальний конструктивізм»: − у сучасному навчальному середовищі ми всі одночасно є потенційними вчителями та учнями; − ми успішні у навчанні, особливо тоді, коли намагаємось створити щось, чи пояснювати щось людям; − розуміння інших дає змогу вивчити їх більш індивідуально; − великий внесок у навчання робить спостереження за роботою більш досвідчених

педагогів; − навчальне середовище має бути гнучким, забезпечувати учасникам навчального процесу простий інструмент для реалізації їхніх навчальних потреб [6]. При проектуванні Moodle особлива увага приділялася таким принципам: − просування педагогіки соціального конструкціонізму (співпраця, активне навчання, критична рефлексія тощо); − підтримка різних підходів до навчання: дистанційне, змішане, очне; − простий, інтуїтивно зрозумілий, ефективний, кросплатформенний інтерфейс у вікні браузера; − проста установка на більшість платформ, що підтримують PHP; − сумісність із більшістю широко використовуваних баз даних; − список курсів містить описи і є доступним будь-якому користувачеві; − курси структуруються за категоріями; − істотна увага приділяється питанням безпеки; − для більшості текстових областей (ресурси, повідомлення форумів тощо) використовується вбудований WYSIWYG HTMLредактор.

Lotus Learning Space

Lotus Learning Space розроблена компанією IBM, надає можливість учитися й викладати в асинхронному режимі, звертаючись до матеріалів курсів у зручний час, брати участь в онлайн-заняттях у режимі реального часу. Викладач може створювати зміст курсу в будьяких програмах і потім розміщувати створений матеріал у Learning Space [7]. Програма має гнучку систему редагування й адміністрування курсу, дозволяє обирати різні режими викладання й відстежувати поточні результати студентів. Курси організовані у вигляді послідовних занять, які можуть бути самостійними, інтерактивними або колективними. Самостійні заняття зазвичай містять матеріали для читання і тести, які необхідно виконати після вивчення матеріалу. Інтерактивні заняття включають лекції у віртуальному класі, участь в онлайновій дискусії або чаті, роботу з віртуальною дошкою (Whiteboard) і системою сумісного перегляду Web-сайтів (Follow me). Інтерактивні заняття плануються на певну дату і часто проводяться викладачем у віртуальному класі в режимі реального часу. Поточні результати студентів (етап проходження курсу, оцінки, витрачений час, кількість звернень та ін.) зберігаються в базі даних. Ця інформація доступна викладачеві у

будь-який час у вигляді звітів різної форми. Колективні заняття передбачають заняття в офлайновій та онлайновій дискусіях, чаті [7].

ATutor

ATutor – модульна система дистанційного керування навчанням із в ідкритим кодом. Поширюється на основі GNU General Public License. Для установки необхідно мати комп'ютер із веб-сервером Apache 1.3.x, PHP версії 4.2.0 та MySQL версій 3.23.x і 4.0.12 (версії 4.1.x і 5.x офіційно не підтримуються). Система розроблена з урахуванням доступності та можливістю адаптації за бажанням користувача. Щодо операційної системи сервера обмежень немає – система є кросплатформеною [5]. Використання системи ATutor дозволяє викладачам легко організовувати різні курси навчання. Студенти ж отримують адаптивне і просте середовище навчання. Адміністратору ця система також особливого клопоту не завдасть. Зовнішній вигляд можна змінити буквально за кілька клацань мишки, доступність вихідного коду й відкриті інструменти, застосовувані для побудови сервера курсів, дозволяють у разі крайньої необхідності внести і більш серйозні зміни. Серед необхідного для створення курсів та управління ними і процесом навчання є й засоби обміну повідомленнями. Особлива увага приділяється й безпеці. За допомогою додаткових модулів можна наростити функціональність. Отже, описані системи дистанційного навчання – програмне забезпечення процесу дистанційного навчання, проте мають разні параметри й можливості. Наведемо основні відомості про описані системи управління у таблиці. Спільними зусиллями програмістів і педагогів розроблено достатню кількість систем дистанційного навчання для організації дистанційного навчання у навчальних закладах різного типу. Передбачити, що буде в майбутньому, складно, але вже зараз можна підібрати необхідне програмне забезпечення для автоматизації навчання, і не тільки дистанційного, але також очного й заочного

Net Школа Україна

Net Школа Україна - комплексна інформаційна система, області застосування якої включають побудову єдиного інформаційного середовища освітньої

установи: школи, гімназії, ліцею, коледжу і тощо; дистанційне навчання в рамках шкільного навчального процесу. Net Школа підтримує різні типи користувачів. Для кожного з типів користувачів гнучко визначаються права доступу до різних частин бази даних. Особливість Net Школи: вона є інтегрованою комплексною системою в масштабі освітньої установи. Система має інтуїтивно зрозумілий інтерфейс і проста в освоєнні. Платформа поширюється на платній основі. Ця система з 2008 року успішно функціонує у НВК № 157 Оболонського району м. Києва (http://new.lyceum157.kiev.ua:81/login.asp), приватній школі «Фортуна» (http://fortunaschool.com.ua/netschool/), ЗОШ № 7 м. Прилуки (http://school7.org/dnetschoolua.html). Перевагою є те, що система дає змогу організувати не лише підтримку навчання школярів за допомого розміщення вчителями матеріалів навчальних дисциплін, доступу до розкладу уроків, журналу оцінок, а й запровадити повноцінне адміністрування кадрової та методичної роботи. До недоліків слід віднести те, що система потребує наявності сервера та професійного обслуговування і адміністрування. Щоденник.ua - Всеукраїнська безкоштовна освітня мережа, яка формує електронне середовище для вчителів, учнів та їх батьків. Проект працює за підтримки Міністерства освіти і науки, Інституту Інноваційних технологій і змісту освіти, регіональних адміністрацій, управлінь освіти. Перевага в тому, що освітня мережа «Щоденник.ua» формує електронне середовище для вчителів, учнів та їх батьків. Серед функцій продукту – розклад уроків, електронний журнал, електронний щоденник, шкільний сайт, шкільна медіатека. Кожна з цих функцій може використовуватись як елемент підтримки електронного навчання. Цю мережу використовують сьогодні, за даними розробників, 10620 шкіл з різних міст та областей України. Серед них Херсонська гімназія № 1 (http://schools.shodennik.ua/school.aspx?school=11697), НВК № 27 м. Маріуполь (http://schools.shodennik.ua/school.aspx?school=2899), СЗШ № 7 м. Стрий (http://schools.shodennik.ua/school.aspx?school=1000000174094) та інші.

Наявність електронних щоденників, розкладу уроків, можливості розміщення навчальних матеріалів, створення персональних сторінок вчителів та учнів орієнтоване саме на підтримку навчального інформаційного простору, не охоплюючи решту, що слід віднести до недоліків.

G Suite for Education

G Suite for Education – це пакет хмарних сервісів для інформаційнотехнологічного підтримування спільної роботи, таких як Gmail, Календар и Клас і додаткових - YouTube, Карты и Blogger, що пропонується компанією Google Inc., як за підпискою (на платній основі), такі і безкоштовно. До переваг G Suite for Education слід віднести те, що пакет є хмарним рішенням, всі дані користувачів зберігаються розподілено в надійно захищених дата-центрах, а не на локальних серверах клієнта. Можливості використання продуктів компанії Google у школі докладно розглянуті у статті А. Шевченко [11]. Зокрема автор наводить приклади використання таких сервісів, як Google календар, Google форми, Google таблиці, Google документи, Blogger для організації навчання школярів, планування розкладу уроків, організації науковометодичної роботи вчителя, керування освітнім процесом. Серед недоліків слід відмітити, що згадані ресурси не є однією цілою системою, хоча й інтегруються один з одним. Кожне нове завдання, яке виникає в процесі створення інформаційного середовища в закладі освіти, потребує пошуку найбільш оптимальних для його розв’язання програмних рішень.

Microsoft Office 365 Education

Microsoft Office 365 Education - це безпечний портал, який виступає в якості базової структури для об'єднання через Інтернет різноманітних рішень в галузі освіти в єдину, повністю керовану середу. Набір програм, що базується на хмарних сервісах і включає в себе безкоштовну електронну пошту, службу обміну миттєвими повідомленнями, засіб проведення відеоконференцій і здійснення голосових викликів. Office 365 для освітніх установ поєднує можливості знайомих додатків Office для настільних систем з Інтернет-версіями нового покоління служб Microsoft для зв’язку і спільної роботи.

Перевагою є те, що, об'єднуючи цілий ряд серверних продуктів Microsoft, таких як поштовий сервіс, базові продукти Office 365, One Note, Skype, SharePoint, Sway та інших, Microsoft Office 365 Education представляє собою масштабований Інтернет-портал, який може бути розгорнутий в освітньому середовищі практично будь-якого розміру. Зокрема на основі використання програмного середовища Microsoft Office 365 був побудований дослідно-експериментальний проект «Хмарні сервіси в освіті» (2014-2017 н.р.) , який охопив 18 шкіл з різних областей України. Запровадження хмарних сервісів Microsoft допомогло підняти на новий рівень організацію роботи і навчання у цих закладах [5, с. 26]. Програмні продукти Microsoft Office 365 дозволили створити в цих школах середовища для комунікації та колаборації учасників освітнього процесу, підтримки дистанційного навчання. Але до недоліків згаданої системи слід віднести ті ж самі, що виникають при застосуванні G Suite – кожен новий потік інформації потребує окремого проектування

2.2 Вибір мови програмування

Мова Java має велику кількість технологій та програмних платформ, фреймворків. На данний час найпопулярнішими технологіями створення Java-додатків є набір специфікацій Java EE та фреймворк Spring.

Jakarta EE (раніше - Java Platform, Enterprise Edition, скор. Java EE, до версії 5.0 - Java 2 Enterprise Edition або J2EE). У 2018 Eclipse Foundation перейменувала Java EE в Jakarta EE - набір специфікацій і відповідної документації для мови Java, яка описує архітектуру серверної платформи для задач середніх і великих підприємств.

Специфікації деталізовані настільки, щоб забезпечити переносимість програм з однієї реалізації платформи на іншу. Основна мета специфікацій - забезпечити масштабованість додатків і цілісність даних під час роботи системи. Java EE багато в чому орієнтована на використання її через веб, як в інтернеті, так і в

локальних мережах. Вся специфікація створюється та затверджується через JCP (Java Community Process) в рамках ініціативи Sun Microsystems Inc.

Java EE є промисловою технологією і в основному використовується в високопродуктивних проектах, в яких необхідна надійність, масштабованість, гнучкість.

Популярності Java EE також сприяє те, що Sun пропонує безкоштовний комплект розробки, SDK, що дозволяє підприємствам розробляти свої системи, не витрачаючи великих коштів. У цей комплект входить сервер додатків GlassFish з ліцензією для розробки.
Spring Framework (або коротко Spring) - універсальний фреймворк з відкритим вихідним кодом для Java-платформи. Також існує форк для платформи .NET Framework, названий Spring.NET [4].

Перша версія була написана Родом Джонсоном, який вперше опублікував її разом з виданням своєї книги «Expert One-on-One Java EE Design and Development» [5] (Wrox Press, жовтень 2002 року).

Фреймворк був вперше випущений під ліцензією Apache 2.0 license в червні 2003 року. Перша стабільна версія 1.0 була випущена в березні 2004. Spring 2.0 був випущений в жовтні 2006, Spring 2.5 - в листопаді 2007, Spring 3.0 в грудні 2009, і Spring 3.1 в грудні 2011. Поточна версія - 5.2.4.

Незважаючи на те, що Spring не забезпечував якусь конкретну модель програмування, він став широко поширеним в Java-співтоваристві головним чином як альтернатива і заміна моделі Enterprise JavaBeans. Spring надає велику свободу Java-розробникам в проектуванні; крім того, він надає добре документовані і легкі у використанні засоби вирішення проблем, що виникають при створенні додатків корпоративного масштабу.

Тим часом, особливості ядра Spring застосовні в будь-якому Java-додатку, і існує безліч розширень і удосконалень для побудови веб-додатків на Java Enterprise платформі. З цих причин Spring набув великої популярності і визнається розробниками як стратегічно важливий фреймворк.

Висновки

У даному розділі зроблено порівняння деяких існуючих програмних продуктів для онайлан навчання та, зокрема, проведення онлайн тестування. Проведено стислий огляд та аналіз обраних систем для навчання. На основі нього зроблено висновок що є потреба у створенні додатка, що буде задовольняти потребам користувачів. Додаток буде мати простий, зрозумілий інтерфейс та легкий в опануванні. Додаток можна буде розмістити на хостинги та на локальному сервері навчального закладу, оскільки потреба в цьому виникає часто.
Для розробки було обрано та розглянуто дві основні технології Java: Jakarta EE та Spring Framework. Spring Framework дає можливість швидко розробляти багатошарові системи, писати менше коду та полегшувати тестування.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

На першому етапі було розроблено конфігурацію проекту: підключення бази даних PostgreSQL для зберігання таблиць даних та база даних HSQLDB, що міститься в пам’яті, для прискорення виконання відлагодження проекту та тестування на етапі розробки.
Клієнтська частина реалізована з використанням джаваскріпт фреймворку Vue JS, каскадних таблиць стилів CSS3 та мови розмітки HTML5.

3.1 Моделі та відповідні класи

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

- клас користувачів User та дочерні 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.

Далі проведено їх об’єктно-реляційне зв’язування. Як приклад розглянуто лістинг:

```
package model.pojos;
import ombok.Data;
import javax.persistence.*;
import java.util.List;
@Data
@Entity
@Table(name = "questions")
        public class Question {
```

```
@Id
(BGeneratedValue(strategy - GenerationType.Auto)
        private long id;
        @Column(nullable = false) private String content;
        @Enumerated(EnumType.ORDINAL) private QuestionType type;
        private String subject;
        private String hint;
        @МапуТоОпе
        @JoinColumn(name = "author_id", referencedColumnName = "id")
                private User author;
        @ManyToMany
        @JoinTable(name = "questions_possible_answers",
                joinColumns = {@JoinColumn(name = "QUESTIONED",
referencedColumnName = "id")},
                inverseJoinColumns = {@JoinColumn(name = "ANSWERID",
referencedColumnName = "id")})
        private List<Answer> possibleAnswers;
        @ManyToHany
        @JoinTable(name = "questionsrightanswers",
                joinColumns = {@JoinColumn(name = "questionid",
referencedColumnName = "id")},
                inverseJoinColumns = {@JoinColumn(name = "answer_id",
referencedColumnName = "id")})
        private List<Answer> rightAnswers;
}
```

Вище наведено описаний POJO-клас, атрібути якого описані анотаціями об’єктно-реляційного відображення, що дозволяє зв’язати атрібути об’єктів з сутностями, тобто, таблицями бази даних та їх полями. У наведеному листингу продемонстровано можливості фреймворку. Головні з них – опис відношень типу *багато – до - одного* (опис Many To One), *багато – до - багатьох* (опис

Many To Many). Також вказано інформацію для побудови проміжних таблиць для створення відношення багато – до - багатьох.

Подібним способом втілено всі інші таблиці бази даних додатку, що формують шар моделі даних. Іноді для перетворення об’єктів в запис бази даних треба змінити деякі поля, для цього використовуються прості класи, так звані, конвертери, наприклад конвертер класу типів питань QuestionTypeConverter. Методи цього класу конвертеру будуть автоматично виконуватися під час перетворення об’єкта даного класу в запис бази даних.

Лістинг класу QuestionTypeConverter

```
package model.converters; inport model.pojos.QuestionType;
import javax.persistence.AttributeConverter;
import javax.persistence.Converter;
@Converter
public class OuestionTypeConverter implements AttributeConverter<QuestionType,
String> {
@Override
public String convertToDatabaseColunin( QuestionType questionType) { switch
(questionType){
case SINGLE_ANSWER: return "SINGLE";
case MULTIPLEANSWERS: return "MULTIPLE";
case COMPREHENSIVE_ANSWER: return "COMPREHENSIVE";
default: throw new IllegalArgumentException("Unknown question type: " +
questionType);
}
}
@Override
public QuestionType convertToEntityAttribute(String dbOata) { switch (dbOata) {
case "SINGLE": return QuestionType.SINGLE ANSWER,
```

```
case "MULTIPLE": return QuestionType.MULTIPLEJNSWERS;
case "COMPREHENSIVE": return QuestionType.COMPREHENSIVE_ANSWER;
default: throw new IllegalArgumentException("Unknow database data: " + dbData);
}
}
```

На рівні моделі даних, поверх рівня класу POJO, знаходиться рівень сховища. Рівень сховища розташований між рівнем домену та рівнем даних. Концептуально, сховище инкапсулює набір об'єктів, що зберігаються в сховищі, і операції, що виконуються з цими об'єктами. Це дає більш об'єктно-орієнтоване уявлення рівня даних. Spring Data JPA надає ряд інтерфейсів, яким можна слідувати, щоб спростити реалізацію цього шаблону. Одним з основних інтерфейсів репозиторіїв є інтерфейс CrudRepository.

Інтерфейс CrudRepository

```
package org.springframework.data.repository;
import ...
@NoRepository8ean
public interface CrudRepositorycT, ID extends Serializable> extends RepositorycT,
ID> {
        <S extends T> S save(S varl);
        <S extends T> Iterable<S> save(Iterable<S> varl);
        T findOne(ID varl);
        boolean exists(ID varl);
        Iterable<T> findAllO;
        Iterable<T> findAll(Iterable<ID> varl);
        long countO;
        void deleted( varl);
        void deleted( varl);
        void delete(Iterable<? extends T> varl); void deleteAl();
```

}

Після цього інтерфейсу стало можливим викликати методи класу-спадкоємця, такі як: save; findOne; існує; знайти всі; розраховувати; видаляти; Видалити все; При реалізації програми для кожної з сутностей створений репозиторій. Це рішення полегшує подальшу розробку, оскільки при подальшому проектуванні рівня контролерів і логіки ви можете думати в звичайному об'єктному контексті, не замислюючись про тонкощі реалізації рівня даних.

HTTP запит, що надіслається клієнтом до REST-API спочатку обробляється DispathcerServlet-ом.

REST контролер
(aftestCont roller
@RequestMappmg (value = '/tests/{testld}')
class TestsRestController {
@Autowired
private final TestRepository testRepository;
@Autowired
private final OuestionRepository questionRepository;
@RequestMapping(method = RequestMethod.POST)
ResponseEntity<?> add(@PathVariable String testld, @RequestBody Test t) {}
@RequestHapping(value = "/{testld}", method = RequestMethod.GET)
Bookmark readiest(@PathVanable String testld) {
@RequestMapping(method = RequestMethod.GET)
Collection<Test> readTests() {
QAutowired
TestsRestController(TestRepository testRepository, QuestionRepository
questionRepositoryl) {
    this.testRepository = testRepository;

```
        this.questionRepository = questionRepositoryl;
>
private void validateTest(String testld) {
>
}
@ResponseStatus(HttpStatus.NOT_FOUND)
class NotFoundExceptio extends RuntimeException {
}
```

Після обробки Dispatcer Servlet знаходить необхідний контролер запитуючи його в спеціального класу, який містить список відповідних ресурсів та контролерів і викликає відповідні методи, в залежності від типу запиту.

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

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.

З іншого боку, об’єкти, які повертає сервер є моделлю даних для клієнтської частини.

3.2 Клієнтська частина додатка

Спілкування клієнтської частини та сервера проходить за допомогою HTTP запитів. Дизайн додатку виконано у стилі мінімалізму та не перенасичено компонентами, це робить використання додака простим, ненав’язливим та водночас функціональним. Наприклад, сторінка з вибором тестів містить назву тесту, тематичну картинку та кнопку для початка тесту.

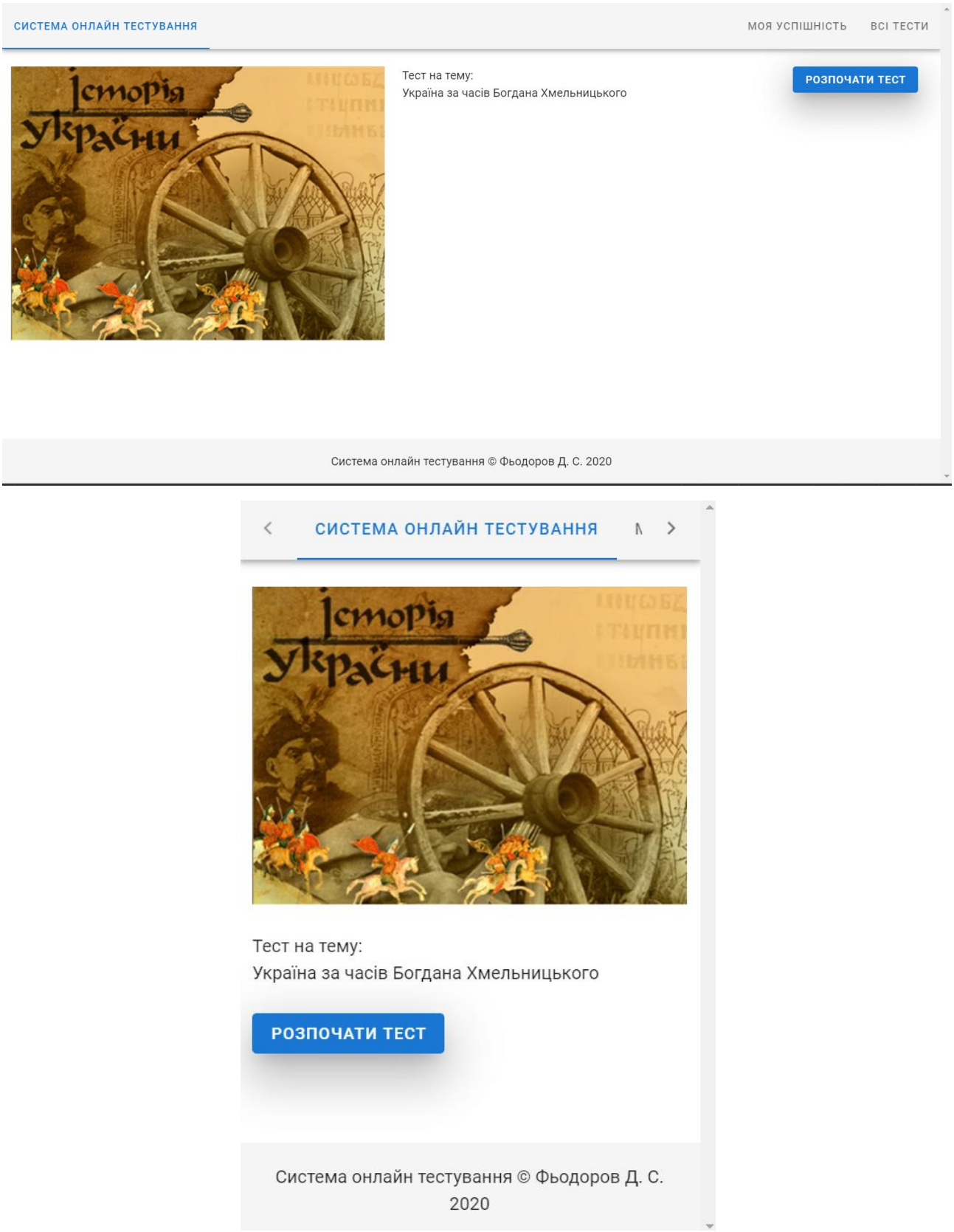

Рисунок 3.6 – Інформація про тест (Демонстрація адаптивного дизайну)

Знизу сторінки розташовано список тестів за обраною дисципліною та випадне меню, через яке можна змінити дисципліну.

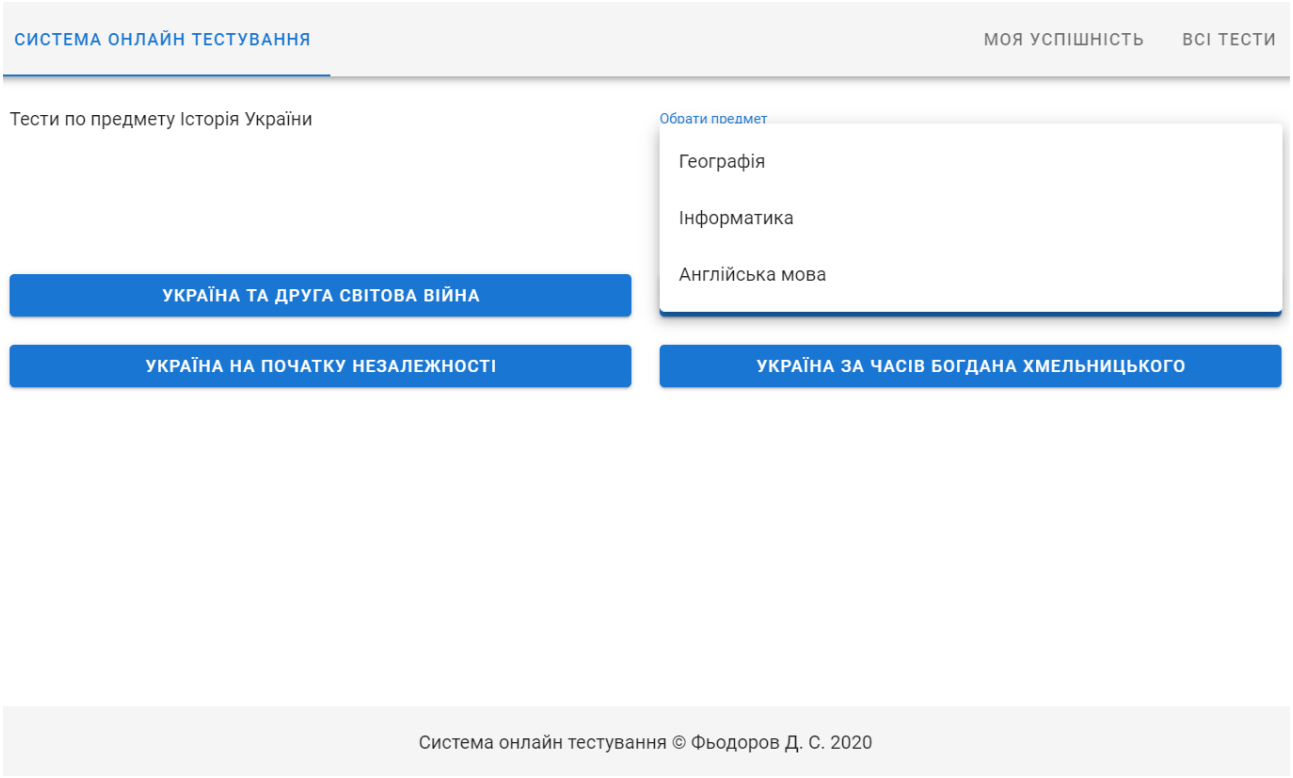

Рисунок 3.7 – Список тестів з обраного предмету

Для розпочинання тесту, користувач натискає кнопку «Розпочати тест», далі він потрапляє на сторінку з тестом, де розташовано сторінки з питанням та списком варіантів відповідей. Описана конструкція сторінки надає можливість розробляти різноманітні тести, відображати кожне питання окремо, при потребі. Описаний підхід знижує час на завантаження сторінки, оскільки розмір JSON-відповіді від сервера значно менший за розмір відповіді з повною HTML-сторінкою. На рисунку 3.8 показано приклад тестового питання при проходженні тесту. Можливість використання підказки, продемонстровано на рисунку 3.9, це робить проходження тесту більш інтерактивним та може знизити нервове напруження учня при навчанні.

У низу сторінки відображено список номерів питань, які є у поточному тесті. Завдяки цьому можна переходити на будь-яке інше питання, якщо у користувача виникли труднощі з поточним питанням і повернутися до нього пізніше.

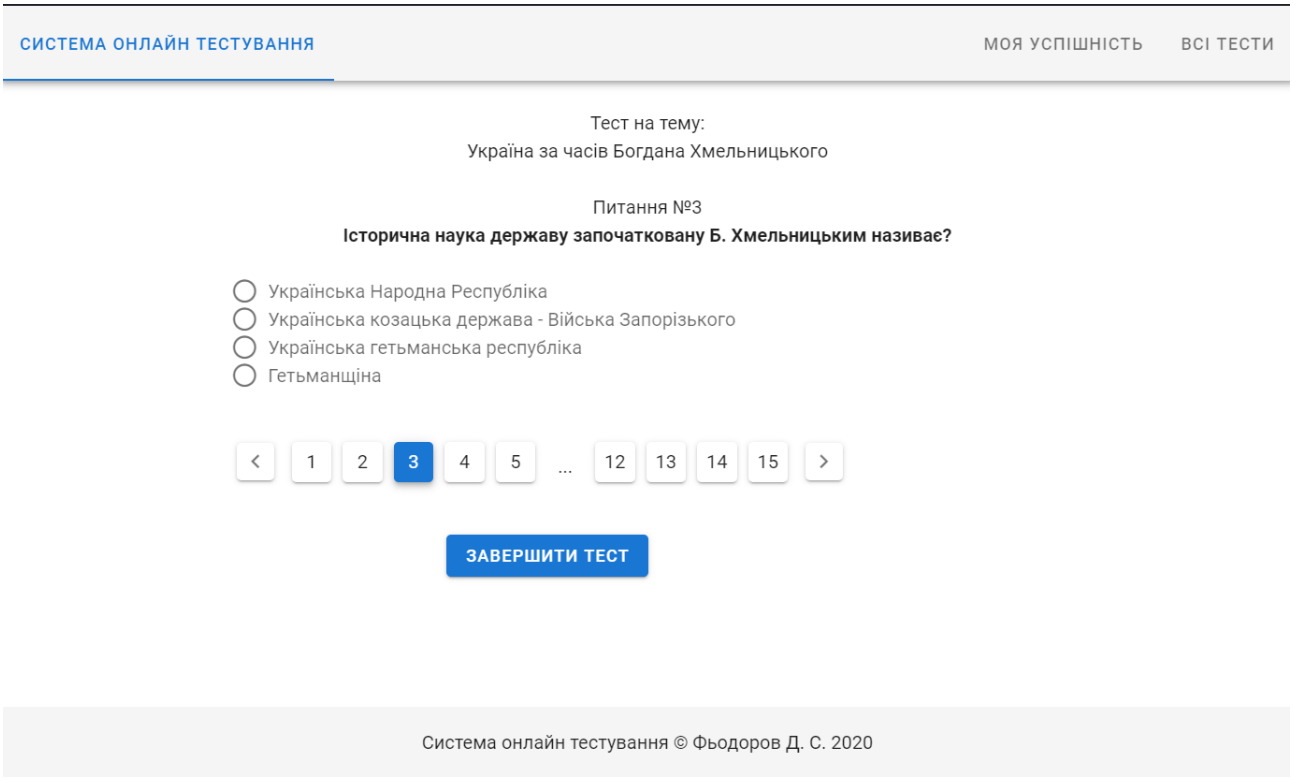

Рисунок 3.8 – Вигляд тестового питання на сторінці

Викладач може створити новий тест, додати питання з варіантами відповідей та обрати тип питання з наведеного переліку, як показано на рисунку 3.11.

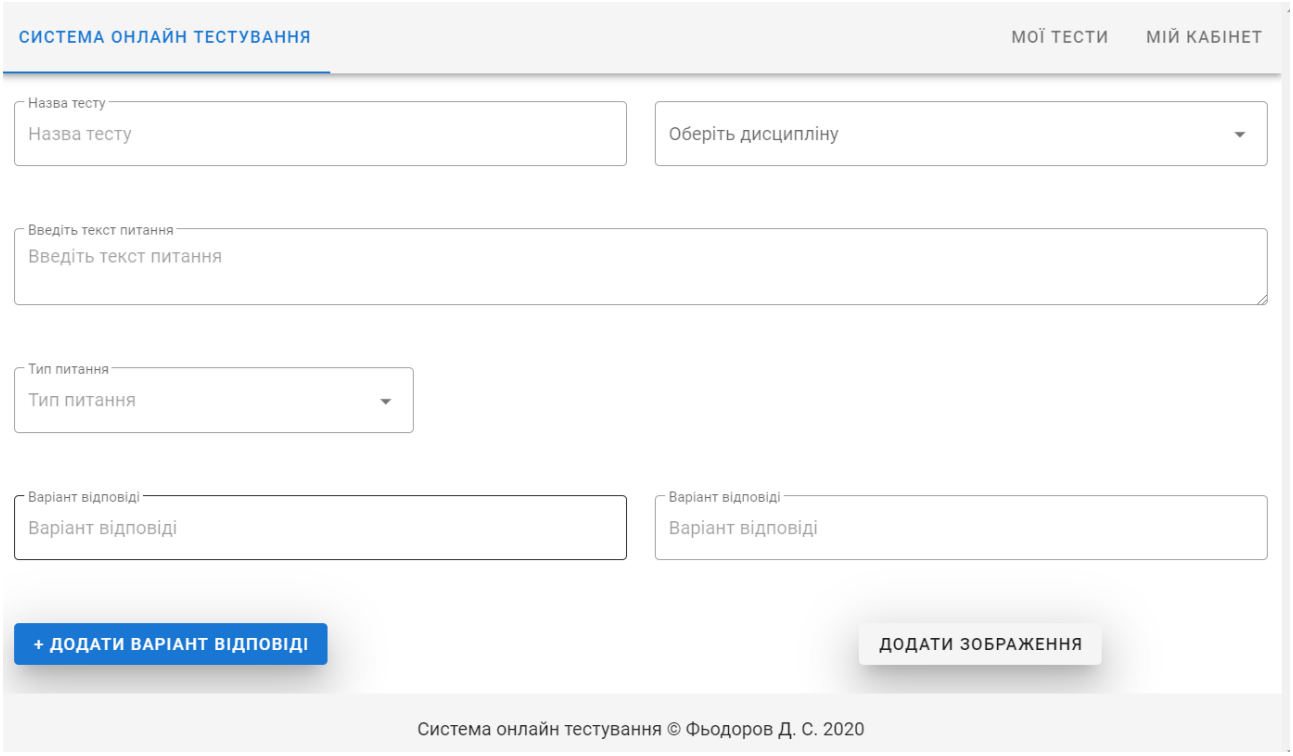

Рисунок 3.11 – Створення тесту

Викладач має можливість переглядати статистику з кількістю правильних відповідей, набраних учнями по обраному тесту.

Висновки

Розроблено веб-додаток у вигляді клієнт-серверної системи. Додаток втілює архітектурний стиль REST, на базі якого побудовано обмін даними між клієнтом та сервером. Показано приклади використання обраних інструментів розробки. Продемонстровано основний функціонал додатка на прикладі клієнта, який побудовано за допомогю JavaScript фреймворка VueJS.

ВИСНОВКИ

В даній роботі було проведено огляд архітектур типу клієнт-сервер та багатошарових архітектур програмних додатків. Проведено аналіз переваг та недоліків архітектур, що дало підстави обрати клієнт-серверну архітектуру з «товстим» клієнтом для розробки програми для тестування.

Обґрунтовано причини широкого застосування багатошарових архітектур програмних додатків. Використання багатошарової архітектури додатку виявилося більш доречним, ніж проаналізовані двошарова та тришарова, і дана архітектура була обрана як модель майбутньої програми. У роботі дано опис та проведено аналіз популярних типів клієнт-серверних взаємодій, таких як, REST та SOAP, та обрано підхід REST.

Проведено огляд та аналіз систем для організації процесу дистанційного онлайн навчання. Аналіз показав, що розгялнуті системи часто мають складний, з точки зору малодосвіченого у користуванні комп'ютером користувача, графічний інтерфейс, складні в розгортанні та потребують великої кількості системних ресурсів.

В якості мови програмування було обрано мову Java та її фреймворк Spring Framework. Оскільки було обрано тип клієнт-серверної взаємоідї REST та технологію AJAX, то для написання клієнтської частини додатка було обрано один з найпопулярніших JavaScript фреймворк VueJS.

Під час створення веб-додатка було продемонстровано типові класи для основних шарів архітектури. Шар моделей даних реалізовано з використанням технології об'єктно-реляційного відображення на основі простих та водночас потужних класів POJO. Поверх даного шару було створено шар хранилищ. Шар контролерів використовується для обробки HTTP 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.

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.

ПЕРЕЛІК ПОСИЛАНЬ

1 Ren Chevance. Server Architectures: Multiprocessors, Clusters, Parallel Systems, Web Servers, Storage Solutions / Ren 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: 24.05.2020

9 A multi-tier architecture for building RESTful Web services – Режим доступу: http://www.ibm.com/developerworks/library/wa-aj-multitier/ - Дата доступу 27.05.2020

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: 27.05.2020

12 Офіційна сторінка XML-RPC – Режим доступу: http://xmlrpc.scripting.com/ - Дата доступу: 30.05.2020

13 Elearning 101 concepts, trends, applications –Режим доступу: http://www.talentlms.com/elearning/ - Дата доступу: 4.05.2020

14 Understanding the Three-Tier Architecture. Oracle Documentation – Режим доступу: http://docs.oracle.com/undtldev010 – Дата доступу: 15.05.2020

15 Офіційний сайт Spring Framework – Режим доступу: https://spring.io/ -Дата доступу: 14.05.2020