МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.

“_____”_____2020 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
“МАГІСТР”

Тема: «Вебсистема організації розкладу та прокторингу екзаменів в навчальному закладі»

Виконавець: студентка, СП-235М, Михайловська Н.В.

Керівник: к.ф.-м.н. Кучерява О.М.

Нормоконтролер: Тупота Є.В.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Напрям (Спеціальність) 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Литвиненко О. Є.
(підпис) (ПІБ)
" " 2020 р.

ЗАВДАННЯ
на виконання дипломної роботи
Михайловської Наталії Володимирівни
(прізвище, ім'я, по батькові)

1.Тема роботи: «Вебсистема організації розкладу та прокторингу екзаменів в навчальному закладі»
затверджена наказом ректора від "27" серпня 2020 року № 1203 /ст.
2.Термін виконання роботи: з 05.10.2020 до 13.12.2020
3.Вихідні дані до роботи: мова програмування *JavaScript*, *C#*, платформа розробки *ASP.Net*, технологія для створення вебсистем *Web Real-Time Communications*
4.Зміст пояснювальної записки (перелік питань, що підлягають розробці):
1. Особливості розроблення вебсистем.
2. Технології для розробки системи прокторінгу.
3. Технологічні засоби для розробки вебсистеми.
4. Розробка вебсистеми організації розкладу та прокторингу екзаменів.

5.Перелік обов'язкового графічного матеріалу:
1. Діаграма діяльності додатку для написання онлайн-іспиту студентом.
2. Діаграма діяльності додатку для перевірки проктором екзамену.
3. Діаграма класів вебсистеми для управління екзаменами.
4. Форма входу до вебсистеми.
5. Форма відображення інформації про екзамен.
6. Робота вебсистеми (схема алгоритму).

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Отримання завдання на дипломну роботу	05.10.2020	
2.	Дослідження та аналіз процесу веброзробки	06.10.2020 – 10.10.2020	
4.	Написання розділу 1	11.10.2020 – 15.10.2020	
5.	Розгляд технології для передачі відео та аудіо файлів	16.10.2020 – 26.10.2020	
6.	Написання розділу 2	27.10.2020 – 01.11.2020	
7.	Вибір інструментів, технологій розробки системи та їх аналіз	02.11.2020 – 07.11.2020	
8.	Написання розділу 3	08.11.2020 – 12.11.2020	
9.	Розробка вебсистеми	13.11.2020 – 30.11.2020	
10.	Написання розділу 4	01.12.2020 – 05.12.2020	
11.	Оформлення пояснювальної записки	06.11.2020 – 10.11.2020	
12.	Підготовка презентації та доповіді	13.12.2020	

7. Дата видачі завдання: "05" жовтня 2020 р.

Керівник дипломної роботи _____ Кучерява О.М.
(підпис керівника) (ПІБ.)

Завдання прийняв до виконання _____ Михайловська Н.В.
(підпис випускника) (ПІБ

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Вебсистема організації розкладу та прокторингу екзаменів в навчальному закладі»: 92 сторінки, 35 рисунків, 1 таблиця, 30 літературних джерел, 1 додаток.

ВЕБСИТЕМА, ПЛАТФОРМА *ASP.NET, WebRTC*, ПРОКТОРИНГ, СЕРВЕР, ЕКЗАМЕН.

Об’єкт дипломної роботи – процес створення вебдодатків.

Предмет дипломної роботи – вебсистема для організації розкладу та прокторингу екзаменів.

Мета дипломної роботи – розробка вебсистеми організації розкладу та прокторингу екзаменів в навчальному закладі.

Методи дослідження – набір інструкцій та команд мови програмування *JavaScript, C#* , мова розмітки даних *HTML*, мова стилю вебсторінок *CSS*, технології для створення вебсистем *ASP.NET* та *Web Real-Time Communications*.

Результатом дипломної роботи є централізована системи для ефективного управління та прокторингу екзаменів студентів. За допомогою системи студенти мають можливість переглядати та змінювати свої особисті дані та деталі курсу, а також переглядати графіки іспитів та оцінювати їх загальну оцінку на основі раніше накопичених оцінок, отримавши доступ до системи за допомогою реєстраційних даних, наданих університетом під час вступу.

Значущість виконаної роботи полягає у наявності безкоштовного, простого у використанні та кросплатформенного вебдодатку для управління екзаменам, перегляду розкладу та прокторингу.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

HTML – HyperText Markup Language (стандартна мова розмітки веб-сторінок в Інтернеті).

CSS – Cascading Style Sheets (спеціальна мова, що використовується для опису зовнішнього вигляду сторінок, написаних мовами розмітки даних).

JS – JavaScript – динамічна, об'єктно-орієнтована прототипна мова програмування.

DOM – об'єктна модель документа (*Document Object Model*).

AJAX (Asynchronous JavaScript And XML) – підхід до побудови користувацьких інтерфейсів веб-додатків, за яких веб-сторінка, не перезавантажуючись, у фоновому режимі надсилає запити на сервер і сама звідти довантажує потрібні користувачу дані.

URL (Uniform Resource Locator) – стандартизована адреса певного ресурсу (такого як документ, чи зображення) в інтернеті (чи деінде).

WebRTC (Web Real-Time Communications) – технологія для зручного працювання з аудіо та відео ресурсами.

ASP (*Active Server Pages*) – технологія для створення вебсистем.

ВСТУП

Актуальність теми. Інтернет став потужною платформою для створення систем, а також змінив спосіб розробки додатків. Цей здвиг у розвитку від розроблення десктопних додатків до вебсистем, які доступні звичайним веббраузерам, призвів до появи безлічі веборієнтованих фреймворків та бібліотек, які дозволяють розробникам використати потужність Інтернет-технологій для досягнення цілей різного роду.

Веброзробка – це побудова та обслуговування вебсистем; це робота спрямована на те, щоб вебсайт мав привабливий вигляд, працював швидко і надійно, тим самим забезпечуючи зручність для користувачів. Процес веброзробки, як правило, можна поділити на дві частини: *front-end* – сторона, що спрямована на користувача – інтерфейс, і *back-end* сторона сервера.

Мета і завдання дипломної роботи. Головною метою дипломної роботи є розробка вебсистеми організації розкладу та прокторингу екзаменів в навчальному закладі. За допомогою системи студенти мають можливість переглядати та змінювати свої особисті дані та деталі курсу, а також переглядати графіки іспитів та оцінювати їх загальну оцінку на основі раніше накопичених оцінок, отримавши доступ до системи за допомогою реєстраційних даних, наданих університетом під час вступу.

Основні завдання та цілі дипломної роботи:

− сформулювати вимоги до створюваної вебсистеми організації розкладу та прокторингу екзаменів;

− організувати у вебсистемі збереження даних, таких як особисті дані студентів, позначки їх модулів, деталі курсу та дані адміністратора, ефективно та безпечно, створюючи захищену базу даних та застосовуючи відповідні методи шифрування;

− описати функціональні характеристики вебсистеми за допомогою таблиць та діаграм *UML*;

− розробити прототип та каркас для вебінтерфейсу для забезпечення включення та досягнення функцій, визначених на етапі вимог користувача, а також для визначення логічної навігації вебсистеми;

− визначити та вибрати технології, платформу та середовище розробки для впровадження вебсистеми шляхом дослідження найбільш часто використовуваних технологій та середовища розробки для створення вебсистеми;

− реалізувати вебсистему із використанням макету, створеного на більш ранньому етапі розробки з урахуванням бажаних вимог користувача (вимірювання цієї мети здійснюватиметься шляхом ретельного тестування вебпрограми в кінці кожного кроку, а також на завершальній стадії розробки, це допоможе викорінити існуючі помилки в системі).

Об’єкт і предмет дослідження. Об’єктом дослідження даної роботи є процес створення вебдодатків. Предметом даної роботи є вебсистема для організації розкладу та прокторингу екзаменів.

Методи дослідження. *Front-end* розробник відповідає за макет, дизайн та інтерактивність за допомогою *HTML, CSS* та *JavaScript*. Частина веб-сайту, з якою користувач безпосередньо взаємодіє, називається інтерфейсом. Його також називають «стороною клієнта» програми. Він включає все, що користувачі відчувають безпосередньо: кольори та стилі тексту, зображення, графіки та таблиці, кнопки, кольори та меню навігації.

HTML розшифровується як мова розмітки гіпертексту. Він використовується для оформлення інтерфейсної частини веб-сторінок за допомогою мови розмітки. *HTML* – це комбінація гіпертексту та мови розмітки. Гіпертекст визначає зв'язок між вебсторінками. Мова розмітки використовується для визначення текстової документації всередині тегу, яка визначає структуру вебсторінок.

Каскадні таблиці стилів, які називають *CSS* – це просто розроблена мова для того, щоб спростити процес створення вебсторінок презентабельними. *CSS* дозволяє застосовувати стилі до вебсторінок.

JavaScript – це відома мова сценаріїв, яка використовується для створення «магії» на сайтах, щоб зробити сайт інтерактивним для користувача. Він використовується для розширення функціональних можливостей.

Розробник *back-end* розробляє те, що відбувається за лаштунками. Тут зберігаються дані, і без цих даних не було б інтерфейсу. *Back-end* Інтернету складається з сервера, на якому розміщена вебсистема, програми для його запуску та бази даних, що містить дані. *Back-end* розробник використовує комп'ютерні програми, щоб забезпечити безперебійну роботу сервера, та бази даних. Для *back-end* розробки використовуються різноманітні мови на стороні сервера, такі як *PHP, C#, Ruby, Python* та *Java*.

ASP.NET – це ще одна технологія для створення вебсистем. Однією з найкращих речей *.NET* є те, що вона базується на об'єктно-орієнтованому програмуванні (ООП). Тут програмне забезпечення ділиться на менші фрагменти, що дозволяє розробникам працювати над ними по черзі. Після закінчення роботи над однією частиною вони можуть перейти до наступної. Коли всі дрібні частини завершені, їх можна поєднувати та управляти ефективніше. Також є чудова система кешування. Простота системи кешування *.NET* робить надійним та простим тимчасове зберігання даних.

.NET постачається з інтегрованим середовищем розробки *Visual Studio* (*IDE*). Цей інструмент дозволяє розробникам створювати програми без особливої «суєти», а також налагоджувати та публікувати їх на різних платформах та ОС. Крос-платформний розвиток – ще одна перевага.

Розгортання програм та їх обслуговування не може бути простіше за допомогою засобів розробки *.NET*. Завдяки своєму модульному дизайну розробники можуть буквально розбирати додатки, а потім виправляти ті, що потребують виправлення або оновлення, а потім складати їх назад. Немає

необхідності пробиратись через океани сценаріїв, аби просто знайти той єдиний рядок, який змушує все збиватися з місця.

Завдяки стандарту *.NET*, розробка додатків з його використанням означає відсутність необхідності переробляти однакові програми для кожної нової платформи. Це завдяки величезній бібліотеці класів, яка складається з функцій, а розробники можуть їх викликати. Це допомагає у таких проектах, які вимагають візуалізації графіки та взаємодії з базами даних. Маніпулювати *XML*-документами також стало набагато простіше.

Для розробки вебсистем також може використоуватись *WebRTC* (*Web Real-Time Communications)*. Це технологія, яка дозволяє вебдодаткам та сайтам отримувати доступ та передавати аудіо та/або відео медіа-потоки, а також обмінюватись власними даними між браузерами, без обов'язкового використання посередників. Набір стандартів, що включає в себе технологію *WebRTC*, дозволяє змінювати дані та проводити пірингові телеконференції, без необхідності користувачеві встановлювати плагіни або будь-яке інше стороннє програмне забезпечення.

Робота з *WebRTC* створена з декількох взаємопов'язаних програмних інтерфейсів (*API*) та протоколів, які працюють разом.

Окрім великого вибору технологій для розробки вебсистем, існує велика кількість напрямків у різних сферах, де вони можуть використовуватись. В дипломній роботі було розглянуто напрямок освіти. В даний час управління іспитами включає багато ручних розрахунків і в основному відбувається на паперовій основі. На приклад, графіки іспитів розсилаються студентам електронною поштою, але вони не є загальнодоступними на вебсайті університету чи в будь-якій вебсистемі, або ці графіки можна лише подивитися в університеті на відповідних інформуючих дошках. Дипломна робота спрямована на запровадження централізованої вебсистеми, яка забезпечить ефективне управління діяльністю в контексті іспитів. Ця динамічна вебсистема забезпечить студентам можливість

перегляду та модифікації своїх особистих даних та деталей курсу, а також перегляду графіків іспитів, отримавши доступ до системи за допомогою облікових даних для входу, наданих адміністратором навчального закладу, а також буде надавати можливість проведенню онлайн-іспитів. Інструмент для оцінки загальної оцінки курсу на основі раніше накопичених оцінок на різних модулях також будуть надані студентам через вебсистему, яка створена.

Наукова новизна отриманих результатів. Система управління екзаменами студентів створена з метою зменшити кількість ручних робіт для управління академічними даними студента, включаючи курс, модулі та результати, лише надавши доступ адміністрації. Вебсистема спрямована на включення інформаційних технологій у сферу управління іспитами під час екзаменаційного «сезону». В ході виконання дипломної роботи отримано програмний засіб, який забезпечує функції онлайн-керування екзаменами. Завдяки широкому функціоналу стало можливим не тільки розміщувати та керувати розкладом екзаменів, а й проводити їх онлайн.

Практичне значення отриманих результатів. Розроблена у даній дипломній роботі вебсистема дозволяє здійснювати контроль за проведенням онлайн-екзаменів, використовуючи будь-який пристрій (комп’ютер, ноутбук, планшет, смартфон). Користувачі з правами адміністратора мають можливість реєструвати студентів на курсах і в вебсистемі, а також додавати та публікувати розклади екзаменів. В вебсистемі реалізована можливість відправки студентам інформацію про їх курс навчання та реєстраційні дані, включаючи облікові дані для входу в вебсистему. Функції та можливості, що запропоновані в розробленій системі, відіграють важливу роль в поліпшенні планування, управління та зручності в проведенні екзаменів онлайн.

РОЗДІЛ 1
ОСОБЛИВОСТІ РОЗРОБЛЕННЯ ВЕБСИСТЕМ

1.1. Поняття та класифікація вебсистем

Вебсистема – це комп'ютерна програма, яка використовує веббраузер для виконання певних функції. Вебсистема – це клієнт-серверна програма. Це означає, що він має сторону клієнта та сторону сервера.

Вебсистеми можуть розроблятися з різних причин і використовуватися компаніями чи приватними особами. Людям це потрібно для полегшення спілкування або, наприклад, придбання речей через Інтернет. Також люди можуть співпрацювати над проектами та працювати над спільними документами за допомогою вебдодатків. Вони можуть створювати звіти, файли та обмінюватися інформацією з будь-якого місця та з будь-якого пристрою.

Вебсистеми розвивалися з моменту їх винаходу. Один з перших додатків, *Perl*, популярна мова сценаріїв на стороні сервера, була розроблена в 1987 році. Це було до того, як Інтернет справді став популярним поза академічними та технологічними колами. Перші вебсистеми були відносно простими і стали більш досконалими наприкінці 90-х. Сьогодні вони є частиною повсякденного життя мільйонів осіб.

Переваги використання вебситсем:

– не потрібно встановлювати на жорсткий диск додаткових програм;

– потребує меншої підтримки та обслуговування з боку розробників та нижчих технічних вимог до комп'ютера користувача;

– вебсистема зменшує витрати як для кінцевого користувача, так і для розробників;

– усі користувачі можуть отримати доступ до однієї версії, що усуває будь-які проблеми сумісності;

– можна отримати доступ до вебсистем де завгодно за допомогою веббраузера;

– кросплатформенність – вебпрограми можуть працювати на декількох платформах, незалежно від операційної системи або пристрою;

– вебсистеми звільняють розробника від відповідальності за створення клієнта, який буде сумісним з певним типом комп'ютера або певноїю операційною системою;

– вебсистеми зменшують піратство програмного забезпечення, оскільки вони знаходяться у відкритому доступі або включать в собі платний функціонал.

Для доступу до вебсистеми потрібне лише підключення до Інтернету. Можна використати веббраузер, такий як *Safari*, *Mozilla Firefox* або *Google Chrome*, щоб підключитися до вебсистеми. Існує три елементи, необхідні для функціонування вебсистеми: вебсервер для обробки запитів від клієнта, сервер додатків для виконання задач, які викликались через запит та база даних для зберігання інформації.

Вебпрограми створюються двома типами мов. Як правило, вони використовують комбінацію сценарію на стороні сервера та сценарію на стороні клієнта. Серверний скрипт займається збереженням та отриманням інформації. Розробники програмують на сервері для створення сценаріїв, які використовуватиме вебпрограма. Для клієнтського сценарію потрібні такі мови, як *JavaScript*, каскадні таблиці стилів (*CSS*) та *HTML*. Ці мови покладаються на браузер для запуску програми. Вони підтримуються браузерами. Клієнтський сценарій займається поданням інформації користувачеві.

Більшість вебсистем мають короткий цикл розробки та можуть створюватися невеликими командами. Деякі програми вимагають обробки на стороні сервера. Їх називають «динамічними». Деякі не потребують обробки на стороні сервера і є статичними.

Розробка вебсистем не обмежується смартфонами або планшетами, швидше цей тип додатків призначений для роботи в будь-якому браузері, як на стаціонарних комп'ютерах, ноутбуках, так і на мобільних пристроях. Існують різні види веб систем.

Найпростіші – це статичні вебсистеми Якщо створюється статична система, перше, що потрібно знати, це те, що програма такого типу відображає дуже мало вмісту і не особливо гнучка. Статичні вебсистеми зазвичай розробляються на *HTML* та *CSS*, але це не єдині платформи для розробки статичного додатка; можна використовувати *jQuery* та *Ajax*. Також можна зручно включати або відображати об'єкти з анімацією, такі як банери, *GIF*-файли, відео тощо. На жаль, змінити вміст статичних вебсистем непросто. Для цього спочатку потрібно завантажити *HTML*-код, потім змінити його і, нарешті, відправити назад на сервер. Ці зміни може вносити лише веброзробник.

Прикладами розробки статичних вебдодатків є професійні портфоліо або цифрові резюме. Подібним чином, сторінка, що представляє компанію, також може використовувати вебдодаток цього типу для відображення їх контактної інформації.

Динамічні вебсистеми набагато складніші на технічному рівні. Вони використовують бази даних для завантаження даних, а їх вміст оновлюється кожного разу, коли користувач отримує до них доступ. Зазвичай вони мають адміністративну панель (так звану *CMS*), де адміністратори можуть виправляти або змінювати вміст програми, будь то текст чи зображення. Різні мови програмування можна використовувати для розробки динамічних систем. *PHP* та *ASP* – найпоширеніші мови, що використовуються для цієї мети, оскільки вони дозволяють структурувати вміст.

У цьому типі додатків оновлення вмісту дуже просте, і для внесення змін не потрібно навіть мати доступ до сервера. Крім того, це дозволяє застосувати безліч функцій, таких як форуми або бази даних.

Якщо вебсистема є Інтернет-магазином, його розробка, швидше за все, буде нагадувати розробку вебсайту мобільної комерції або електронної комерції. Цей тип розробки додатків є більш складним, оскільки він повинен дозволяти електронні платежі, які можна здійснювати за допомогою кредитних карток, *PayPal* або інших способів оплати. Розробник також повинен створити панель

управління для адміністратора; які будуть використовуватися для переліку нових продуктів, оновлення їх, видалення записів та управління додатком та платіжками.

Вебсистема портал – мається на увазі тип системи, яка отримує доступ до різних розділів або категорій через домашню сторінку. Ці програми можуть включати багато речей: форуми, чати, електронну пошту, браузери, області, доступ до яких здійснюється через реєстрацію, найновіший вміст тощо.

Анімація неминуче пов'язана з технологією *Flash*. Цей підхід програмування дозволяє відображати вміст з анімованими ефектами. Цей тип додатків забезпечує більш креативний та сучасний дизайн і є однією з ключових технологій, що використовуються дизайнерами та креативними директорами.

Недоліком, властивим розробці анімованих вебсистем, є те, що цей тип технології не підходить для цілей вебпозіціонування та оптимізації *SEO*, оскільки пошукові системи не можуть належним чином читати інформацію, яку вони містять.

Коли процес доходить до розробки вебсистеми, контент повинен постійно оновлюватись, тому встановлення системи керування контентом (*CMS*) є серйозним варіантом для розгляду. Адміністратор може використовувати цю систему управління контентом тільки для внесення змін та оновлень. Ці вебсистеми з управлінням контенту зрозумілі та дуже прості в управлінні.

Деякі приклади систем управління контентом: *WordPress*: безперечно, є найпоширенішим прикладом таких систем та в Інтернеті є безліч інформації, навчальних посібників та путівників, які допоможуть його налаштувати та зрозуміти, як працює, окрім крім усього, це безкоштовна система; *Joomla* – ця система управління контентом посідає друге місце після *WordPress*; *Drupal* – це безкоштовне програмне забезпечення *CSM* – дуже адаптоване і особливо рекомендується для створення спільнот.

Цей тип вебсистем дуже поширений серед тематичних сторінок: особисті щоденники, корпоративні щоденники, професійні блоги, сторінки новин, статті, засоби масової інформації тощо.

1.2. Типи архітектур вебсистем

Будь-яка вебсистема складається з клієнтської та серверної сторони.

Клієнт – це зручне представлення функціональних можливостей вебпрограми, з якими взаємодіє користувач. Написаний *HTML, JavaScript* та *CSS,* існують у веббраузері користувача та не потребують певних налаштувань, пов'язаних з ОС/пристроєм.

Для створення серверної частини необхідна розробка за допомогою таких мов: *PHP*, *Java*, *.NET, Python, Ruby on Rails* або *Node.js* та інших. Зазвичай ця сторона складається ще щонайменше з двох частин: вебсервера з логікою системи (або головного центру управління) та бази даних (зберігання всіх постійних даних). Якщо масштабувати цю сторону, це означає, що збільшується кількість вебсерверів та баз даних, що підвищує продуктивність та стабільність вебпрограми.

Архітектура вебсистем – це схема взаємодії між компонентами вебпрограми. Спосіб планування цієї взаємодії визначає стійкість, продуктивність та безпеку майбутнього вебдодатку, а отже вибір правильної архітектури компонентів сприяє підвищенню якості роботи майбутньої вебсистеми. Розглянемо плюси і мінуси можливих моделей.

Можна виділити 3 основні типи архітектури вебсистем та описати їх переваги та недоліки. Можна оцінити їх з трьох різних точок зору: власник програмного забезпечення, розробник програмного забезпечення (член спеціальної команди, яка розробляє проекту) та кінцевий користувач.

Критерії користувача до вебсистем це по-перше зручність використання: оновлення даних на сторінках, перемикання між сторінками (час відгуку). Такі якості користувальницького інтерфейсу, як насиченість та інтуїтивність. Інший критерій це лінкування – можливість зберігати закладки та посилання на різні розділи вебсайту. І ще один критерій це робота в автономному режимі

Критерії розробника: швидкість розвитку (впровадження нових функцій, рефакторинг, розпаралелювання процесу розробки програмного забезпечення); продуктивність (максимальна швидкість реакції сервера з мінімальним

споживанням обчислювальної потужності); масштабованість (можливість збільшити обчислювальну потужність або простір на диску при збільшенні обсягу інформації та/або кількості користувачів); тестування (можливість і простота автоматизованого модульного тестування).

Якщо говорити про критерії власника програмного продукту, то важливим є функціональна розширюваність – нова функціональність за мінімальний час та бюджет. Також важливим є *SEO* – користувачі повинні мати можливість знайти програму через будь-яку пошукову систему. Ще один критерій – підтримка, оскільки окрім розробки програмного забезпечення, існують додаткові витрати на обладнання, мережеву інфраструктура, обслуговування. Не менш важливою є безпека. Власник програмного забезпечення повинен бути впевнений, що як дані, так і інформація про користувачів захищаються.

Клієнт-серверна архітектура (рис. 1.1) найпоширеніша архітектура вебсистем. Сервер генерує *HTML*-контент і відправляє його клієнту як повноцінну *HTML*-сторінку. Іноді цю архітектуру називають "*Web* 1.0", оскільки вона з'явилася першою і в даний час домінує у сфері веброзробки.

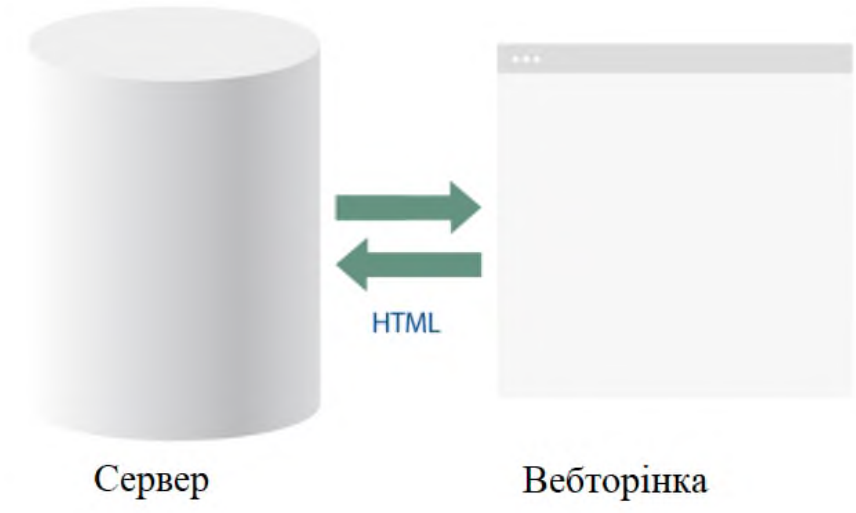

Рис. 1.1. Клієнт-серверна архітектура

У даній архітектурі величезна кількість даних передається між сервером і клієнтом. Користувач повинен почекати, поки вся сторінка не перезавантажиться,

реагуючи на тривіальні дії, наприклад, коли потрібно перезавантажити лише частину сторінки. Шаблони інтерфейсу на клієнті залежать безпосередньо від фреймворків, застосованих на сервері.

Неможливо надіслати миттєві оновлення даних або зміни в режимі реального часу. Якщо розглянемо можливість оновлення в режимі реального часу за допомогою генерації готових фрагментів контенту на стороні сервера та оновлень клієнта (через *AJAX*, *WebSockets*), а також дизайну з частковими змінами на сторінці, ми вийдемо за рамки цієї архітектури.

У даній архітектурі одна *URL*-адреса отримує певний *HTML*-вміст на сервері, а *SEO* легко реалізується, подібно до попереднього критерію – зміст відомий заздалегідь. Також це найстаріша архітектура веброзробки, тому можна вибрати будь-яку мову сервера та структуру для певних потреб.

Якщо подивимось на генерацію *HTML*, під збільшенні навантаження настає момент, коли потрібен баланс навантаження. Ситуація з масштабуванням баз даних набагато складніша, але це завдання є однаковим для цих трьох прикладів архітектури програмного забезпечення.

Продуктивність цієї архітектури порівняно низька, оскільки потрібно передати велику кількість даних, що містить *HTML*, дизайн та бізнес-дані. Тому необхідно генерувати дані для цілої сторінки (не тільки для змінених бізнес-даних), а також усієї супровідної інформації (наприклад, дизайну).

Хороша річ полягає в тому, що немає необхідності в спеціальних інструментах, які підтримують інтерпретацію *JavaScript*, для тестування інтерфейсу, а вміст статичний.

Логіка поведінки додатків знаходиться на стороні сервера. Однак дані передаються відкрито, тому може знадобитися захищений канал (що в основному є історією будь-якої архітектури, якщо це стосується сервера). Вся функціональність захисту знаходиться на стороні сервера.

Архітектура на основі віджетів (рис. 1.2) згенерованих *JS* (*AJAX*). Це еволюціонувала архітектура першого типу. Різниця полягає в тому, що сторінка, яка відображається у браузері, складається з віджетів (функціонально незалежних

блоків). Дані завантажуються до цих віджетів через запит *AJAX* із сервера: або як повноцінний фрагмент *HTML*, або як *JSON*, і перетворюється (за допомогою шаблону/прив'язки *JavaScript*) у контент сторінки. Варіант завантаження фрагментів *HTML* виключає необхідність використання фреймворків *JavaScript-MV** на стороні клієнта; в цьому випадку можна використовувати щось простіше (наприклад, *jQuery*). Знижуючи інтерактивність, збільшуємо швидкість розробки та робимо функціонал дешевшим та надійнішим.

Найголовніша перевага полягає в тому, що оновлення з сервера надходять лише для тієї частини сторінки, яку вимагає клієнт. Також добре, що віджети розділені функціонально. Певний віджет відповідає за частину сторінки; часткові зміни не вплинуть на всю сторінку.

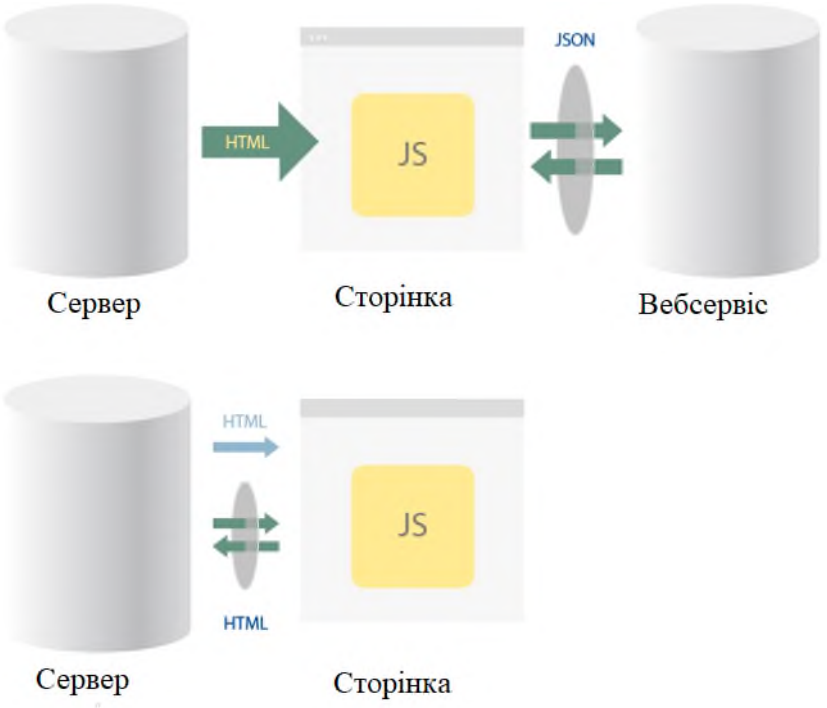

Рис. 1.2. Структура на основі віджетів згенерованих *JS*

Обсяг переданих даних для частини сторінки менший, ніж для цілої сторінки, тому швидкість реагування вища. Але оскільки сторінка є набором віджетів, застосовні шаблони інтерфейсу користувача у вебпрограмі обмежені вибраною структурою інтерфейсу. Холодний запуск (перше повне завантаження) такої сторінки займе трохи більше часу. Вміст, який повністю генерується та кешований

на сервері, може негайно відображатися на клієнті; тут витрачається час на отримання даних для віджету і, як правило, на шаблонування. Під час першого відвідування вебсайту завантажуватиметься не так швидко, але надалі він буде набагато приємнішим у використанні, якщо порівнювати з сайтами, заснованими на архітектурі першого типу. Також варто згадати про можливість здійснення "часткового".

Для *SEO* задач існують спеціальні механізми. Наприклад, для просування вебсистем, заснованих на цій архітектурі, можна заздалегідь визначити список сторінок для просування і створити для них статичні *URL*-адреси без параметрів та модифікаторів.

Щодо швидкості розробки таких систем, то потрібно необхідно серверні технології веброзробки та використовувати *JavaScript* на стороні клієнта. Також потрібно реалізувати вебсервіси на стороні сервера.

Час і ресурси, витрачені на створення вмісту *HTML*, відносно незначні, якщо порівнювати з часом, витраченим програмою на отримання даних із баз даних та на їх обробку перед створенням шаблону. Використання розширеного типу цієї архітектури (коли дані передаються як *JSON*) зменшує трафік між клієнтом та сервером, але додає рівень абстракції додатку: отримання з бази даних, потім обробка даних, серіалізація в *JSON* потім передача на *API*:

Тестувати такі системи більш складно. Потрібно протестувати сторону сервера, сторону клієнта та вебслужбу, яка повертає дані для оновлення віджетів.

Останній тип – орієнтовані на сервіс односторінкові вебпрограми: програми *Web* 2.0, *HTML5* (рис.1.3). Термін "*Web* 2.0" тут не зовсім правильний. Однією з особливостей *Web* 2.0 є принцип залучення користувачів до наповнення та повторних налаштувань вмісту. В основному термін "*Web* 2.0" означає проекти та послуги, які активно розробляються та вдосконалюються самими користувачами: блоги, вікі, соціальні мережі. Це означає, що *Web* 2.0 не пов'язаний з однією технологією або набором технологій.

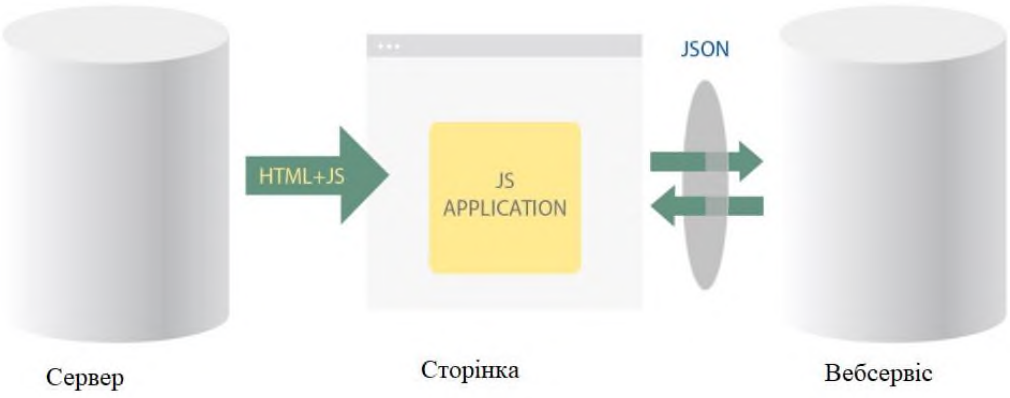

Рис. 1.3. Структрура орієнтованих на сервіс вебпрограми

Розглянемо суть цієї архітектури. *HTML*-сторінка завантажується з сервера. Ця сторінка є контейнером для коду *JavaScript*, який звертається до певної вебслужби та отримує лише дані. Дані використовуються додатком *JavaScript*, який генерує *HTML*-контету сторінки. Ця архітектура є самодостатнім і досить складним додатком *JavaScript*, де частина функціональних можливостей переходить на сторону клієнта. Для порівняння, дана архітектура не може показувати велику кількість взаємопов'язаних та структурованих функцій.

У сучасній веброзробці програми, що працюють у режимі офлайн, є рідкістю (за невеликими винятками, наприклад, *rad-js.com*). Цей підхід дозволяє легко здійснити зворотне перетворення: опублікувати існуючу програму в Інтернеті.

Обсяг даних, що передаються для оновлень, мінімальний. Інтерфейс користувача створюється за допомогою *JavaScript*, можна реалізувати будь-які необхідні варіанти. Є проблема з багатопоточністю в *JavaScript*: у цьому конкретному випадку обробка великих обсягів бізнес-даних повинна бути перенесена на вебслужбу.

Щодо розробки систем з такою архітектурою, то потрібно розробити вебсервіс і застосувати більш спеціалізовані фреймворки *JavaScript*, які будують архітектуру додатків. Оскільки архітектура є відносно новою, не так багато фахівців, здатних створити якісний сайт/систему на основі цього підходу. Існує не так багато перевірених часом інструментів, основ та підходів.

На критерій продуктивності найменш впливає сторона сервера. Сервер повинен лише передати браузеру код. З боку клієнта, продуктивність та тип браузера мають найбільше значення.

Веблогіка стоїть на стороні клієнта. На сервері немає генерування вмісту. Коли кількість користувачів збільшується, потрібно масштабувати лише вебслужби, які надають дані про бізнес.

Розглянемо безпеку таких систем. Логіка перенесена на клієнтський *JavaScript*, який зловмисник може відносно легко змінити. Для захищених систем потрібно розробити превентивну архітектуру, яка враховує особливості програм з відкритим кодом.

Ця архітектура є повноцінним додатком; можна зберігати окремі дані, а також частини програми за допомогою будь-якого сховища (наприклад, локального сховища). Ще однією перевагою є можливість перевести зберігання даних та керування ними в автономний режим. Для порівняння, дві вищезазначені архітектури функціонують лише частково в автономному режимі. Тут відсутні дані можна замінити макетами, можна показати вікна попереджень або використовувати дані з локальної пам’яті, а синхронізацію можна залишити на потім.

1.3. Етапи створення вебсистем

Процес створення вебсистеми складається з декількох етапів (рис. 1.4). Існує багато додатків з хорошим вмістом, але поганим дизайном. Крім того, сайт з хорошим дизайном не потребує обслуговування. Веброзробка – це не впровадження коду на вебсайті; вебдизайн також відіграє важливу роль у процесі розробки. Кожній системі потрібна система управління контентом (*CMS*).

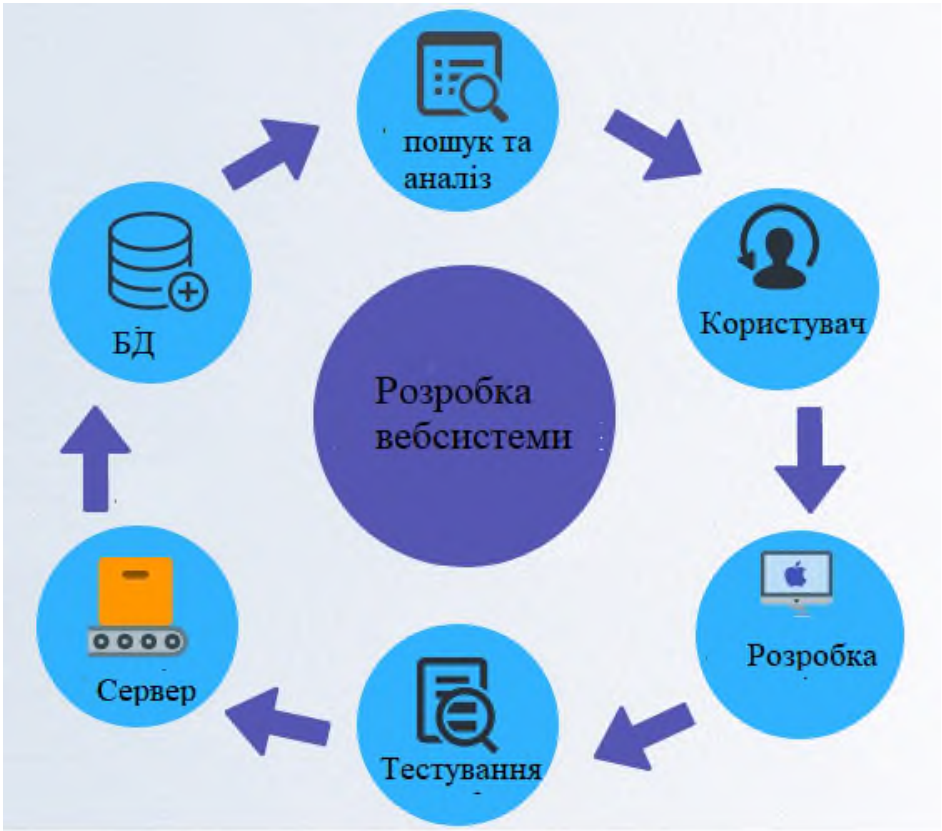

Рис. 1.4. Процес веброзробки

Перший етап розробки – збір інформації. На цьому початковому етапі дизайнеру потрібно визначити кінцеву мету дизайну системи, як правило, у тісній співпраці з клієнтом або іншими зацікавленими сторонами. Питання для вивчення та відповіді на цьому етапі процесу проектування та розробки веб-сайтів включають: для кого призначений сайт, що очікується там знайти чи зробити, чи основна мета цього додатку – інформувати, продавати (електронна комерція) чи розважати та інші питання.

Це найважливіша частина будь-якого процесу веброзробки. Якщо ці короткі запитання не дають чіткої відповіді у брифі, цілий проект може піти в неправильному напрямку.

Може бути корисним виписати одну або кілька чітко визначених цілей або короткий зміст очікуваних цілей в одному абзаці. Це допоможе поставити дизайн на правильний шлях. Потрібно переконатися, що є розуміння цільової аудиторії.

Після того, як зібрано достатньо інформації про бізнес, настав час для створення мап сайтів та каркасних карток. Карта сайту складається з інформацією,

зібраною на першому етапі. Основним мотивом мапи сайту є створення зручного для користувача вебсайту та створення структури сайту. Каркас забезпечує візуальний опис сайту.

Вебдизайн – це ключова частина успіху Інтернет-системи. Вебдизайн створюється відповідно до цільової аудиторії. Вебсистема, яка розробляється для школи, абсолютно відрізняється від того, що розробляється для продажу речей. Інші речі, про які слід пам’ятати, - це тема, кольорова гама, де розмістити текст, зображення, відео тощо. Дизайн-макет структурує вашу сторінку систематично, щоб вона виглядала привабливою.

Перший етап розробки – реалізація систем, тобто розробники вебсистеми пишуть код. Веброзробник повинен забезпечити щоб система працювала безперебійно. Це важливий крок у реалізації, оскільки графічний дизайн з попереднього етапу оживає. Відповідно до мап сайтів, домашня сторінка розробляється спочатку, ніж інші сторінки.

Третый етап – заповнення контенту. Після процесу розробки настав час заповнити контент у вебсистемі. Відмінний та привабливий вміст необхідний, щоб привернути увагу людей. Вміст слід модифікувати для з заголовками, підзаголовками, тегами тощо, щоб люди могли знайти те, що вони шукають.

Тестування – це ще один етап веброзробки. Кожні сторінки та посилання перевіряються перед запуском системи, щоб переконатися, що нічого не порушено. Необхідно перевірити кожну форму, сценарій та запустити програмне забезпечення для перевірки правопису, щоб знайти можливі помилки друку.

Слід використовувати валідатори коду, щоб переконатися, що ваш код відповідає чинним стандартам веброзробки. На цьому етапі ваш сайт перевіряється на кілька речей, включаючи швидкість вебсистеми, сумісність між браузерами.

Після вдалого тестування вебсистема завантажується на сервер і готова до використання.

1.4. Вебсервер

Вебсервер – це комп’ютер, на якому зберігається вебвміст. В основному вебсервер використовується для розміщення вебсистем, але існують і інші вебсервери.

Вебресурс – це колекція вебсторінок, а вебсервер – це програмне забезпечення, яке відповідає на запит вебресурсів.

Вебсервер відповідає на запит клієнта одним із наступних двох способів: надсилання файлу клієнту, пов’язаному із запитуваною *URL*-адресою або генерування відповіді шляхом виклику сценарію та спілкування з базою даних

Коли клієнт надсилає запит на вебсторінку, вебсервер шукає сторінку,на яку було створено запит, якщо дану сторінку знайдено, тоді він надішле її клієнту з відповіддю *HTTP*. Якщо запитувану вебсторінку не знайдено, вебсервер надішле відповідь *HTTP*: Помилка 404 Не знайдено.

Якщо клієнт запитував деякі інші ресурси, вебсервер зв'яжеться із сервером додатків та сховищем даних для побудови відповіді *HTTP*.

Архітектура вебсервера використовує наступні два підходи: паралельний підхід та підхід, керований одним процесом та подією.

Паралельний підхід дозволяє вебсерверу одночасно обробляти кілька запитів клієнтів. Цього можна досягти наступними методами: багатопроцесний, багатопоточний та гібридний метод.

У багатопроцесному методі єдиний процес (батьківський процес) ініціює кілька однопоточних дочірніх процесів і розподіляє вхідні запити до цих дочірніх процесів. Кожен з дочірніх процесів відповідає за обробку одного запиту. Багатопоточний метод, на відміну від багатопроцесорного, створюється безліч однопоточних процесів. А гібридний – це поєднання вищезазначених двох підходів. При цьому підході створюється кілька процесів, і кожен процес ініціює кілька потоків. Кожен з потоків обробляє одне з'єднання. Використання кількох потоків в одному процесі призводить до меншого навантаження на системні ресурси.

1.5. Кросплатформеність

У світі розробки додатків існують різні платформи пристроїв, для яких потрібно розробляти системи. Традиційно програми для кожної з цих платформ повинні будуватися окремо, оскільки кожна операційна система використовує різну мову програмування, не визнану іншою.

Існує, різниця між нативними та міжплатформенними програмами і вона полягає в наступному. Нативні програми написані різною мовою для кожної операційної системи. Це означає, що кожна система (*Android, iOS* тощо) вимагає окремого процесу розробки.

Кросплатформні програми мають єдиний код, який інтерпретується та коректується на всіх операційних системах.

Переваги використання кросплатформної розробки:

1. Зменшення витрат, оскільки немає необхідності виконувати кодування двома-трьома окремими мовами. Крім того, помилки, які присутні в загальній кодовій базі, потрібно виправити лише один раз (здебільшого).

2. Скорочення часу виходу на ринок, оскільки розробка однієї програми займає набагато менше часу, ніж створення двох-трьох програм для кожної платформи пристрою.

3. Зростання рівномірності між різними операційними системами, створюючи більш зручний інтерфейс користувача.

4. Охоплення більшої аудиторії клієнтів, що збільшить майбутню рентабельність інвестицій.

5. Використання сучасних мов програмування та засобів розробки

6. Створення прототипів, дозволяє швидше вийти на ринок більш ніж на одній платформі та дозволяє власнику збирати відгуки користувачів, займати позицію на ринку та захищати будь-які патенти, якщо це необхідно. Є кілька недоліків використання кросплатформної розробки. Програми можуть бути менш ефективними. Це відбувається через відсутність апаратного забезпечення для виконання плавної анімації *HTML5* на пристроях низького або середнього рівня,

надлишковість процесів для різних платформ або роботи в операційних системах, яким більше 3 років.

Використання вдосконалених функцій при кросплатформенній розробці може зайняти більше енергії акумулятора та набагато більше ресурсів, ніж у нативному додатку.

1.6. Технологія *AJAX*

AJAX – це техніка веброзробки для створення інтерактивних вебдодатків та систем. *AJAX* розшифровується як асинхронний *JavaScript* та *XML*. *AJAX* – це нова техніка для створення кращих, швидших та інтерактивніших вебдодатків за допомогою *XML, HTML, CSS* та *JavaScript*.

Ajax використовує *XHTML* для вмісту, *CSS* для презентації, а також об'єктну модель документа та *JavaScript* для динамічного відображення вмісту.

Звичайні вебпрограми передають інформацію на сервер і з сервера за допомогою синхронних запитів (рис. 1.5). Це означає, що коли заповнюється форма, натискається кнопка «Відправити» та в результаті відбувається перехід на нову сторінку з новою інформацією з сервера. За допомогою *AJAX*, коли натискається кнопка «Відправити» *JavaScript* зробить запит на сервер, інтерпретує результати та оновить поточний екран. Користувач ніколи не дізнається, що щось навіть передається на сервер.

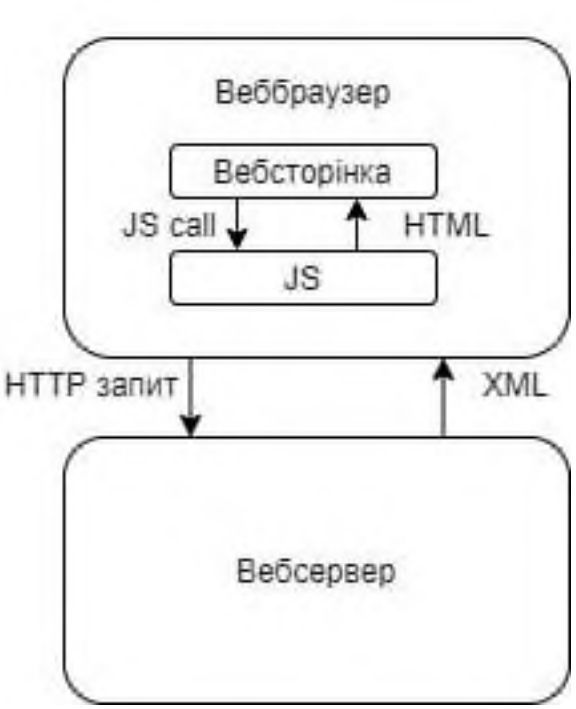

Рис. 1.5. Принцип роботи *AJAX*

XML зазвичай використовується як формат отримання даних сервера, хоча можна використовувати будь-який формат, включаючи звичайний текст.

AJAX – це технологія веббраузера, яка не залежить від програмного забезпечення вебсервера. Користувач може продовжувати використовувати програму, поки клієнтська програма запитує інформацію від сервера у фоновому режимі.

AJAX базується на наступних відкритих стандартах – презентація на основі браузера з використанням *HTML* та каскадних таблиць стилів (*CSS*). Дані зберігаються у форматі *XML* і отримуються з сервера. Закулісні дані отримують за допомогою об’єктів *XMLHttpRequest* у браузері. *JavaScript* для реалізації всього.

AJAX не може працювати самостійно. Він використовується у поєднанні з іншими технологіями для створення інтерактивних вебсторінок: *JavaScript* – мова сценаріїв, функція *JavaScript* викликається, коли подія відбувається на сторінці, також він з’єднує всі операції *AJAX*; *DOM – API* для доступу та маніпулювання структурованими документами, також представляє структуру *XML* та *HTML* документів; *CSS* – дозволяє чітко відокремити стиль відобаження від вмісту і може

бути програмно змінений за допомогою *JavaScript*; *XMLHttpRequest* – об'єкт *JavaScript*, який виконує асинхронну взаємодію з сервером.

Етапи роботи *AJAX*:

– відбувається подія клієнта;

– створюється об'єкт *XMLHttpRequest*;

– об'єкт *XMLHttpRequest* налаштовується;

– об'єкт *XMLHttpRequest* робить асинхронний запит до вебсервера;

– вебсервер повертає результат, що містить *XML*-документ;

– об'єкт *XMLHttpRequest* викликає функцію зворотного виклику і обробляє результат;

– *HTML DOM* оновлено.

Щодо безпеки *AJAX* на стороні сервера, то вебпрограми на базі *AJAX* використовують ті самі схеми захисту на стороні сервера, що і звичайні вебпрограми. Визначаються вимоги щодо автентифікації, авторизації та захисту даних у файлі *web.xml* або у своїй програмі. Вебпрограми на базі *AJAX* зазнають тих самих загроз небезпеки, що і звичайні програми.

Безпека *AJAX*: на стороні клієнта: код *JavaScript* видно користувачеві/хакеру. Хакер може використовувати код *JavaScript* для знаходження слабких сторін сервера. Код *JavaScript* завантажується з сервера і виконується у клієнта і може скомпрометувати клієнта за допомогою «поганого» коду.

1.7. Висновки до розділу

В першому розділі було розглянуто основи веброзробки, що собою являє вебсистема, її особливості. Отже, вебсистема – це клієнт-серверний додаток, за допомогою якого відбувається взаємодія клієнта з сервером через веббраузер. Вони мають багато переваг, по-перше, не потрібно додаткового програмного забезпечення, щоб відкрити будь-яку вебсистему, окрім браузера, нелазено від операційної системи яка встановлена на пристрої. Також вебсистеми потребують

менше програмної підтримки зі сторони розробника, а зі сторони користувача – нижчих технічних вимог до пристрою на яких буде відкрита система.

Щодо класифікації вебсистем, то виділяють статичні та динамічні вебсистеми, інтернет-магазин, вебсистема-портал, вебсистеми управління контентом та інші.

Також у даному розділі було розглянуто різні типи архітектур вебсистем, їх переваги та недоліки. Можна виділити основні: клієнт-серверна, архітектура на основі віджетів згенерованих *JS* та архітектура, яка орієнтована на сервіс вебпрограми.

У даному розділі було розглянуто етапи створення вебсистем. Для створення вебпрограми спочатку необхідно провести аналіз та пошук, щоб зробити висновки про те, для чого ця система буде створена, наскільки вона буду актуальною. Слід звертати увагу на потреби користувача, оскільки будь-яка вебсистема є клієнтно-орієнтованою. Після впровадження системи її потрібно протестувати, а потім завантажувати на сервер та налаштовувати базу даних.

Будь-яка вебсистема не може існувати без вебсерверу. З точки зору «заліза» вебсервер – це комп’ютер, на якому зберігаються ресурси вебсистеми. Це сервер що приймає запити від клієнта, зазвичай через веббразуер, та видає їм відповідь на даний запит.

Одні з ключових моментів веброзробки це є кросплатформенність та використання технології *AJAX*. Кросплатформенність – це зручність як і для веброзробника, так і для користувача системою. Зі сторони розробника, то непотрібно створювати декілька версій своєї системи, щоб вона підтримувалась на різних пристроях і різних операційних системах. Також це полегшує підтримку таких систем, та зменшує витрати на їх реалізацію. Зі сторони користувача не потрібно встановлювати додаткових програм, щоб користуватися функціоналом системи. Також не обов’язково мати від рукою комп’ютер, щоб скористатися системою, оскільки вона є доступною на будь-якому пристрою.

РОЗДІЛ 2
ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ СИСТЕМИ ПРОКТОРИНГУ

2.1. Поняття системи прокторингу

Онлайн-прокторинг – це онлайн-екзамен, прив’язаний до часу або поза часом. Під час його проходження віддалений проктор та/або програмне забезпечення спостерігають за людиною та її комп’ютером за допомогою демонстрації робочого столу, відео з вебкамери та аудіо (рис. 2.1).

Як і традиційний прокторинг, онлайновий залучає проктора, який спостерігає за учасником тестування, щоб підтвердити свою особу, відповісти на будь-які запитання, що виникають у них, а також запобігти, виявити та/або повідомити про шахрайство і зловживання. Використовуючи безпечні технології, онлайн-прокторинг є альтернативою тестуванню в аудиторіях.

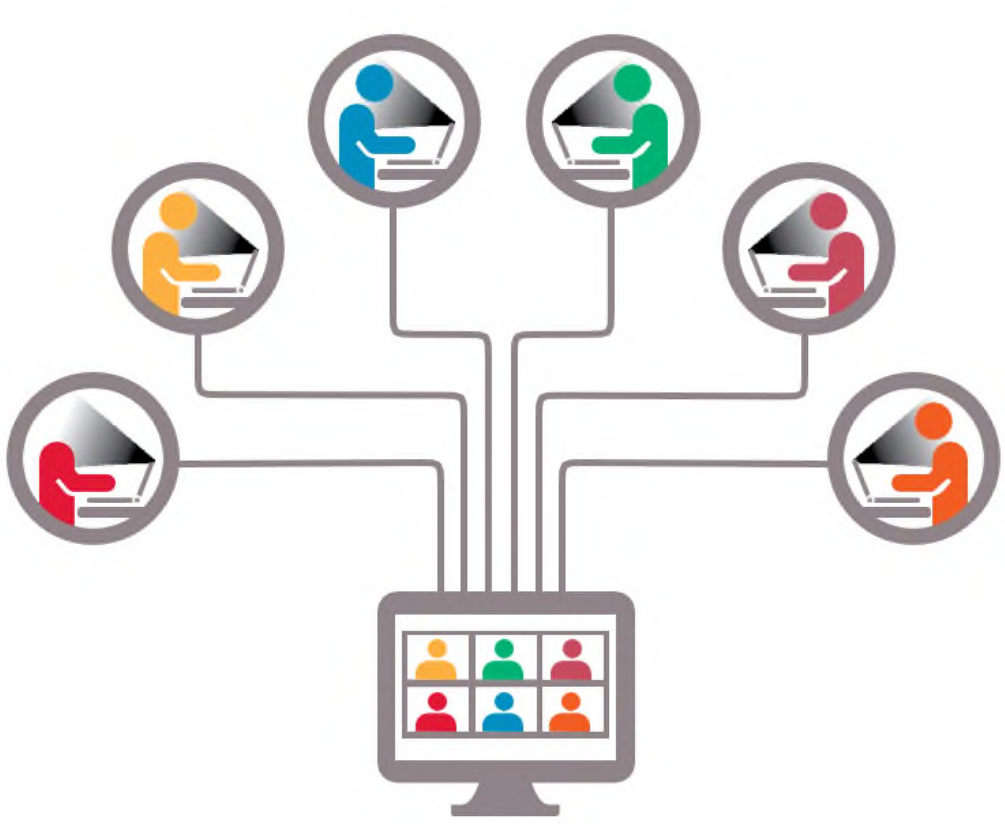

Рис. 2.1. Проведення онлайн-прокторингу

Можна виділити три основні види прокторингу:

1. Прокторинг у реальному часі, в якому спостереження за процесом відбувається під час іспиту.

2. Іспити, які перевіряються пізніше на основі записів відеокамери та екрану користувача.

3. Автоматизований прокторинг, де програмне забезпечення контролює процес написання екзамену.

Онлайн-прокторинг – це найдавніша і найвідоміша форма інтернет-прокторингу. Ця форма найбільше нагадує реальну аудиторію, в якій проходить екзамен, при цьому проктор віддалено контролює іспит.

Кількість іспитів, які може прийняти один проктор, залежить від обраного методу. Чим більше екранів повинен стежити проктор, тим менше іспитів можна контролювати одночасно. Проктор може втрутитися під час іспиту, як і у залі для іспитів. Наприклад, під час іспиту з відкритою книгою вони можуть попросити студента витруситись або показати свою книгу, щоб довести це всередині немає прихованих записок або шпаргалок.

Найбільшими недоліками цієї форми є обмежена масштабованість та необхідність планувати іспит заздалегідь. Студент не може просто увійти в систему і розпочати роботу, як тільки відчує готовність; натомість їм потрібно запланувати час за кілька днів до того, щоб проктор мав бути доступним. Потужність системи визначається кількістю доступних виконавців.

Подальше зберігання та перевірка. Ця загальновживана форма онлайн-прокторингу зберігає зображення камери, щоб пізніше можна було переглянути (прискорене) відео. На основі зображень оцінюється, чи було вчинено шахрайство під час іспиту.

Найбільша перевага цієї форми полягає в тому, що студенти можуть складати іспит, коли вони готові. Вони можуть увійти відразу і розпочати іспит, не плануючи його заздалегідь. Ще одна перевага полягає в тому, що ця форма легко масштабована і може впоратися з одночасними іспитами. Велика кількість студентів може складати іспити одночасно, а викладачі можуть оцінювати їх протягом більш тривалого періоду.

Недоліком є те, що викладач не може втрутитися під час іспиту, тобто він не може сказати студенту, що певна дія є незаконною або якщо камера неправильно розміщене, і проктор не має повного огляду столу.

В умовах автоматизованого прокторингу, що стає все популярнішим, проктори більше не стежать за процесом іспиту. Натомість програмне забезпечення визначає моменти, коли існує шахрайство. Наприклад, щоразу, коли відкривається інше програмне забезпечення, студент відводить погляд або в кімнаті виявляється інша людина. Проктор попереджається про події такого роду, і він може потім переглянути конкретні моменти, щоб оцінити, чи насправді вчинено шахрайство.

Автоматизований прокторинг робить процес написання екзаменів набагато ефективнішим та економить багато часу, оскільки не всі відеозаписи потрібно переглядати.

Одним з недоліків є те, що якщо студенти знають, як працює програмне забезпечення, вони зможуть легше уникнути заходів щодо виявлення шахрайства. А навпаки, проктор залишається непередбачуваним для студента, оскільки неможливо бути впевненим, що вони контролюють у будь-який момент часу. Іншим недоліком є те, що програмне забезпечення легко виробляє помилкові спрацьовування (тобто повідомляє про невинні події як потенційне шахрайство).

Особливості Інтернет-системи прокторингу:

1. Аутентифікація кандидата.

Інструмент онлайн-пошуку забезпечує належну автентифікацію кандидата. Це перший крок для реєстрації кандидата, в якому ідентифікація кандидата перевіряється через *OTP*, *Aadhaar* або *IP*-адресу. Цей крок полягає у забезпеченні того, що учасник тестування є оригінальним кандидатом, а не імітатором.

2. Обмеження перегляду.

При цьому адміністратор може заздалегідь визначити кількість спроб залишити тестовий екран. Якщо кандидат покидає сторінку тестування занадто багато разів, тест автоматично закінчується.

3. Розширені функції управління.

Якщо кандидата підозрюють у шахрайстві чи введенні в оману діяльності, тоді піднімаються червоні прапори. У разі постійного підняття червоних прапорів для кандидата, проктор може легко призупинити або закінчити тест. Крім того, функція копіювання-вставлення вимкнена на пристрої під час тесту.

4. Детальний звіт.

В кінці тесту інструмент віддаленого пошуку генерує детальний звіт як для учасника тесту, так і для адміністратора тесту. Звіт містить поглиблений аналіз результатів роботи кандидата під час тестування. У ньому підкреслюється кількість підняття прапорів та будь-який інший вид незвичної діяльності, яка мала місце під час тесту.

Розглянемо переваги прокторингу для учасників. Гнучкість – кандидати можуть складати іспит, в день або певний час коли їм зручно. Оскільки організатор іспиту не обмежується одним конкретним іспитовим днем, а може запропонувати варіанти з кількох днів, протягом якого можна скласти іспит, та можна вибрати час, який підходить.

Кандидати мають можливість скласти іспит в комфорті власного будинку чи офісу. Це більше розслабляє, ніж перебування в тестовому центрі, і знайоме та тихе середовище допомагає їм зосередитися під час іспиту.

Подібно до інших іспитів в Інтернеті, кандидати можуть працювати за комп'ютером, як і в звичайному робочому житті. Це дозволяє уникнути болючої руки при спробах писати годинами і дозволяє використовувати Інтернет-інструменти, такі як електронні таблиці та текстові редактори.

Також можна заначити переваги для організацій, які проводять іспити за допомогою онлайн-прокторингу. Іспити можна проводити частіше – замість того, щоб концентрувати всі іспити на кілька великих іспитових днів, їх можна проводити частіше і ближче до періоду навчання.

Також є перевага підвищеного захисту, оскільки дуже важко обдурити, коли невидимий оглядач постійно переглядає вас через великий відеоекран, який перевіряє ваше посвідчення особи та ваше оточення та спостерігає за будь-якими порушеннями. Якщо трапляються серйозні порушення, вони можуть скасувати

доступ до іспиту, якщо це потрібно. За певних типів онлайн-прокторингу іспит записується, і якщо виникають якісь проблеми, орган, що перевіряє, може переглянути відео на будь-якому етапі після цього.

2.2. Технологія *WebRTC*

Для передачі відео з різних браузерів і з мінімальною затримкою є технологія *WebRTC*.

WebRTC (*Web Real Time Communications)* – це безкоштовний проект з відкритим вихідним кодом, який надає веббраузерам і мобільним додаткам можливості взаємодії в режимі реального часу через прості інтерфейси прикладного програмування (*API*) . Ця технологія дозволяє аудіо і відеозв'язку працювати всередині вебсторінок, дозволяючи встановлювати пряме однорангове з'єднання, усуваючи необхідність установки плагінів або завантаження додатків (рис. 2.2).

WebRTC дозволяє швидко і легко налаштувати однорангові з'єднання з іншими веббраузерами. Щоб побудувати таку систему з нуля, вам знадобиться безліч фреймворків та бібліотек, що займаються типовими проблемами, такими як втрата даних, переривання з'єднання та обхід *NAT*. З *WebRTC* все це вбудовується в браузер. Ця технологія не потребує плагінів або сторонніх програм. Він відкритий, а його вихідний код знаходиться у вільному доступі.

Рис. 2.2. Компоненти *WebRTC* системи

API WebRTC включає захоплення медіа, кодування та декодування аудіо та відео, транспортний рівень та управління сесіями.

Першим кроком є отримання доступу до камери та мікрофона пристрою користувача. Визначається тип доступних пристроїв, отримується дозвіл користувача на доступ до цих пристроїв та керуємо потоком.

Надсилати потік аудіо- та відеоданих через Інтернет непросте завдання. Тут використовується кодування та декодування. Це процес розділення відеокадрів та звукових хвиль на менші шматки та стиснення їх. Цей алгоритм називається кодеком. Існує величезна кількість різних кодеків, які підтримуються різними компаніями з різними бізнес-цілями. У *WebRTC* також є багато кодеків, таких як *H.264, iSAC, Opus* та *VP8*. Коли два браузери з'єднуються разом, вони вибирають найбільш оптимальний кодек, що підтримується між двома користувачами. На щастя, *WebRTC* виконує більшу частину кодування за лаштунками.

Транспортний рівень управляє порядком пакетів, має справу з втратою пакетів та підключенням до інших користувачів. Знову *WebRTC API* надає легкий доступ до подій, які повідомляють, коли виникають проблеми з підключенням.

Управління сесією займається управлінням, відкриттям та організацією зв'язків. Це зазвичай називають сигналізацією. Якщо передається аудіо- та відеопотоки користувачеві, також має сенс передати додаткові дані. Це робиться за допомогою *API RTCDataChannel*.

Інженери таких компаній, як *Google, Mozilla, Opera* та інші, проробили чудову роботу, щоб перенести цей досвід у реальному часі в Інтернет.

Етапи роботи *WebRTC*:

1) Відкриття користувачем сторінки, що містить тег <*video*> .
2) Запит доступу до вебкамери і мікрофону користувача браузером.
3) Параметри з'єднання для обходу *NAT* і *Firewall*, такі як *IP*-адреси і порти сервера *WebRTC* і *WebRTC* клієнтів, контролюються на сторінці користувача *JavaScript* кодом.
4) Початок процесу кодування і передачі потоків даних між *WebRTC* клієнтами.

Переваги стандарту *WebRTC*.:

1) Відсутність вимог встановлення додаткового ПО.
2) Використання сучасних відео та аудіо-кодеків, вбудованої системи ехо- і шумозаглушення, а також автоматичного регулювання рівня чутливості мікрофонів учасників.
3) Всі з'єднання захищені і зашифровані згідно з протоколами, що говорить про високий рівень безпеки.
4) Наявність можливості використання вбудованого механізму захоплення контенту.
5) Наявність можливості реалізації різних інтерфейсів керування на основі *HTML5* і *JavaScript*.
6) Наявність можливості впровадження в власний продукт, так як проект має відкритий вихідний код.

7) Являється кросплатформенним стандартом, так як необхідна лише підтримка стандарту в браузері, і не важливо яка операційна система встановлена на пристрої. Це значно економить ресурси на розробку програмного забезпечення.

Оскільки *WebRTC* дозволяє спілкуватися в режимі реального часу або обмінюватися даними, для підключення потрібен фактичний локальний *IP* кінцевої точки.

Для спілкування з іншим браузером технологія потребує *RTCPeerConnection*. Це ядро однорангового з'єднання між кожним із браузерів. Для створення об'єктів *RTCPeerConnection* необхідно написати: *var pc = RTCPeerConnection (config).*

Потім технологія проходить через протокол *SDP. SDP* – це скорочення для протоколу опису сеансу. Протокол опису сеансу визначає стандарт для визначення параметрів для обміну медіа (часто потокового мультимедіа) між двома (як правило) кінцевими точками. Протокол несе інформацію про яка *IP*-адреса готова приймати вхідний медіапотік, який номер порту прослуховує вхідний медіапотік, який тип носія очікує кінцева точка (як правило, аудіо), в якому протоколі очікує обмін інформацією кінцева точка, яке стиснення, що кодує кінцеву точку, здатне декодувати (кодек)

Існує велика кількість випадків використання *WebRTC*, які можуть варіюватися від простого аудіо- або відеозв’язку до спільного використання файлів або навіть спільного використання екрану тощо.

Для спілкування використання *WebRTC* дозволяє користувачам спілкуватися один з одним через свій веббраузер (наприклад, за допомогою *Skype* або *WhatsApp*), не встановлюючи жодного додаткового плагіна або доповнення.

Для підтримки клієнтів технологія *WebRTC* актуальна, оскільки вона забезпечує функцію взаємодії в режимі реального часу між браузерами, служба підтримки клієнтів може використовувати таку можливість, щоб вони могли відповідати на запити в режимі реального часу.

Для спільного використання файлів – *WebRTC* не обмежується лише спілкуванням в Інтернеті, але ця технологія поширює його використання навіть на спільний доступ до файлів, а також спільний доступ до екрану.

Також, може виникнути питання: «Чи технлогія *WebRTC* безпечна?». Так, *WebRTC* захищена, оскільки використовує наскрізне шифрування. Як результат, жодна стороння особа не може з’явитися, щоб переглянути ваші дані. Але якщо використовується *VPN*, поганою новиною є те, що *WebRTC* може просочити вашу адресу *IP*.

При розробці чогось, що дозволяло б здійснювати голосові та відеодзвінки кілька років тому, швидше за все, для цього використовували *C/C++*. Це означає тривалі цикли розробки та більші витрати на розробку.

WebRTC має верхній шар *API Javascript*, який можна використовувати всередині браузера. Це значно полегшує розробку та інтеграцію комунікацій у режимі реального часу в будь-якому місці. Внутрішньо *WebRTC* все ще в основному реалізується за допомогою *C/C++*, але більшості випадків, під час використання *WebRTC*, не потрібно буде копатись глибоко в цих шарах, щоб розробляти свої програми.

API WebRTC складається з кількох основних об'єктів *javascript* - *RTCPeerConnection, MediaStream, RTCDataChannel*.

Об'єкт *RTCPeerConnection* є основною точкою входу в *API WebRTC*. Це допомагає з'єднуватися з одноранговими вузлами, ініціалізувати з'єднання та приєднувати медіапотоки. Він також управляє з'єднанням *UDP* з іншим користувачем.

Основним завданням об'єкта *RTCPeerConnection* є налаштування та створення однорангового з'єднання (рис. 2.3). Ми можемо легко перехопити точки з'єднання, оскільки цей об'єкт запускає набір подій, коли вони з'являються. Ці події дають вам доступ до конфігурації нашого з'єднання.

```
[code]
var conn = new RTCPeerConnection(conf);

conn.onaddstream = function(stream) {
   // use stream here
};

[/code]
```

Рис. 2.3. Створення об`єкту *RTCPeerConnection*

Сучасні браузери надають розробнику доступ до *API getUserMedia*, також відомого як *API MediaStream*. Є три ключові моменти функціональності. Він надає розробнику доступ до об'єкту потоку, який представляє відео та аудіопотоки. Він управляє вибором вхідних користувацьких пристроїв на випадок, якщо користувач має на своєму пристрої кілька камер або мікрофонів. Він забезпечує рівень безпеки, який запитує користувача весь час, коли він хоче отримати потік.

Окрім надсилання медіапотоків між одноранговими вузлами, також можна надсилати додаткові дані за допомогою *API DataChannel*. Цей *API* такий же простий, як *API MediaStream*. Основна робота – створити канал, що надходить із існуючого об'єкта *RTCPeerConnection*.

Щодо доступності *WebRTC*, сьогодні у всіх сучасних браузерах. *Google Chrome*, *Mozilla Firefox*, *Apple Safari* та *Microsoft Edge* підтримують *WebRTC*. Також можна *WebRTC* та інтегрувати його в додаток або вбудований пристрій, не потребуючи браузера.

Головне, що *WebRTC* робить, це дозволяє доступ до пристроїв, тобто можна отримати доступ до мікрофона вашого пристрою, камери, яка є у вас на телефоні чи ноутбуці – або це може бути сам екран, можна зафіксувати відображення користувача, а потім дозволити спільний доступ до цього екрану або його запис. Також *WebRTC* не обмежується лише голосом та відео. Це дозволяє надсилати будь-які типи довільних даних.

2.3 Сервер *Janus WebRTC Server*.

Для того щоб вручну не налаштовувати медіа-зв'язок з браузером можна використаний *WebRTC* сервер загального призначення – *Janus WebRTC Server*.

Сервер *Janus WebRTC* був задуманий як сервер загального призначення. Як такий, він не надає ніяких функціональних можливостей, крім реалізації засобів для налаштування медіа-зв'язку *WebRTC* з браузером, обміну повідомленнями *JSON* з ним і передачі повідомлень між браузерами і логікою додатка на стороні сервера, з якою вони пов'язані. Будь-яка конкретна функція або додаток повинні бути реалізовані в плагінах на стороні сервера, щоб потім браузери могли зв'язуватися через ядро *Janus*, щоб скористатися перевагами, цих функцій. Прикладом таких плагінів можуть бути реалізації програм, таких як ехо-тести, конференц-мости, медіа-рекордери, *SIP*-шлюзи тощо.

Взаємодія комп'ютерів при використанні *Janus-gateway*, які знаходяться в мережі і знаходяться за *NAT* представлено на рисунках (рис. 2.3, рис. 2.4). Із малюнків зрозуміло, що *Janus-gateway* являє собою *STUN*, сигнальний і медіа-сервер, і користувачі вже взаємодіють не між собою, а через *Janus*.

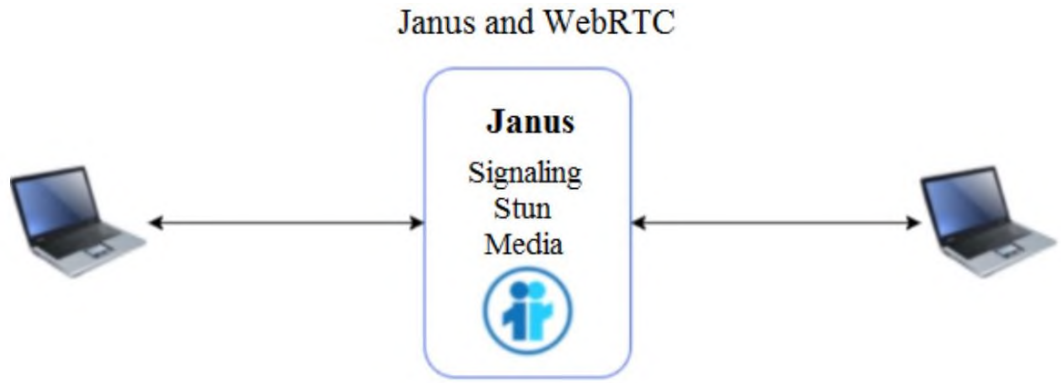

Рис.2.3. *Janus* взаємодія без *NAT*

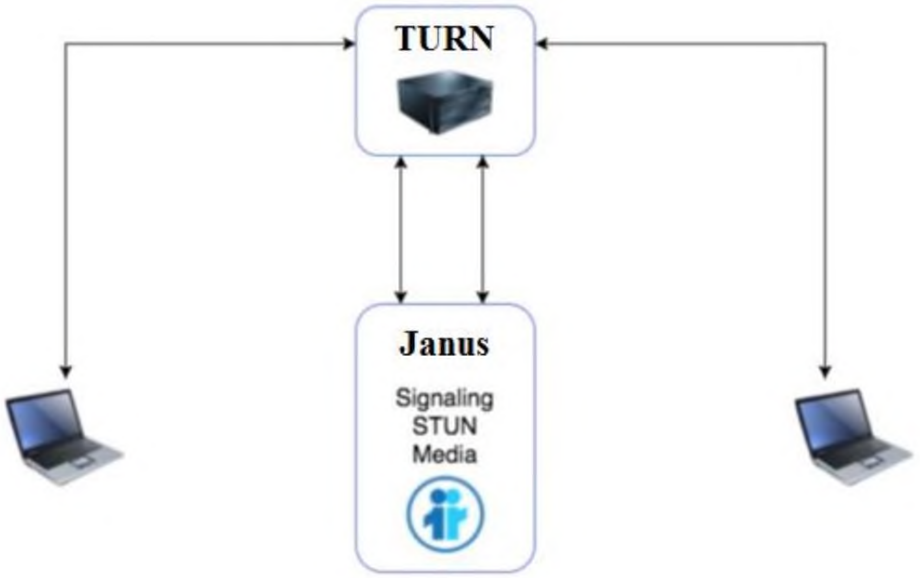

Рис.2.4. *Janus* взаємодія з *NAT*

Утиліти обходу сеансів для *NAT* (*STUN*) – це протокол, який служить інструментом для інших протоколів у роботі з обхідним процесором мережевих адрес (*NAT*). Кінцева точка може використовувати його для визначення *IP*-адреси та порту, призначеного їй *NAT*. Він також може бути використаний для перевірки зв'язку між двома кінцевими. *STUN* працює з багатьма існуючими *NAT* і не вимагає від них особливої поведінки.

STUN сам по собі не є рішенням для обходу *NAT*. Швидше, це інструмент, який слід використовувати в контексті рішення обходу *NAT*.

2.4. Опис процесу прокторингу та архітектури вебсистеми дистанційного проведення екзаменів

Система відеомоніторингу створюється для дистанційного навчання. Архітектура вебсистеми відеомоніторингу представлена на рис. 2.5.

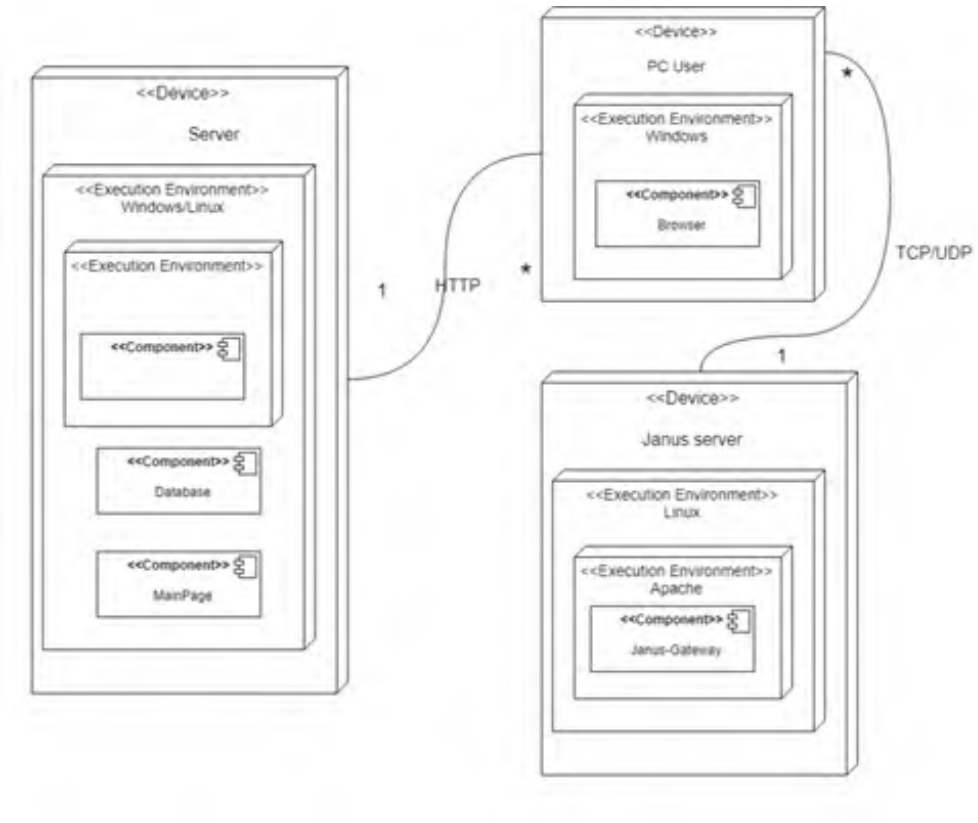

Рис. 2.5. Архітектура вебсистеми

Компонент *MainPage* – основна сторінка, з якою взаємодіє користувач. Компонент *MainPage* взаємодіє з компонентом серверу по протоколу *HTTP*. При запуску тестування *MainPage* починає передавати медіа дані компоненту *Janus-Gateway* по протоколам *TCP*/*UDP*. В цей же час компонент *MainPage* проводить ідентифікацію користувача.

Розглянемо процес проведення екзамену онлайн з системою прокторингу. При відкритті тесту студентом повинна початися ініціалізація з'єднання з *Janus* сервером. Коли студент почне тестування повинні паралельно початися передача відео на сервер і процес ідентифікації студента. При завершенні тестування передача відео повинна припинитися і відеозапис має зберегтися на сервері.

При відкритті тесту викладачем повинна початися ініціалізація з'єднання з *Janus* сервером. При натисканні на кнопку «подивитися відео» відеозапис має передатися з сервера, після чого викладач міг би його подивитися. Процес проведення тестування за допомогою вебсистеми після впровадження системи відеомоніторингу і ідентифікації користувача представлений на рис. 2.6-2.7.

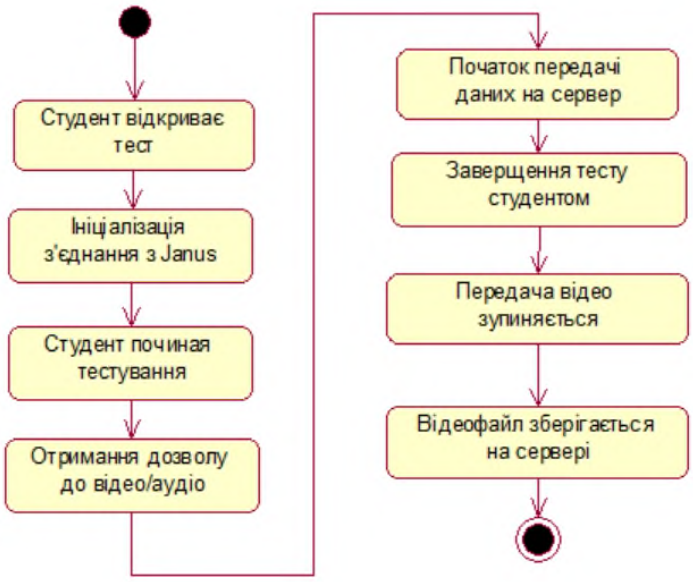

Рис. 2.6. Діаграма діяльності студента

Рис. 2.7. Діаграма діяльності проктора

Для розробки такої системи спочатку необхідно створити модуль який буде за допомогою технологій *WebRTC* записувати відео на сервер. Захват відео реалізовується за допомогою методу *getUserMedia()*. Метод *getUserMedia()* робить запит до користувача дозволити використання одного пристрою вводу відео (наприклад, камери або загального екрана) та до одного пристрою вводу звуку у якості джерела для *MediaStream*. Якщо запит на використання пристроїв з

відкритим словом, не існує сумісних пристроїв вводу або є яка-небудь інша умова, що викликає помилку, буде викликана зворотня функція *MediaStreamError з* об'єктом, який описує в чому є помилка.

Для полегшення роботи з *WebRTC* використовується сервер *Janus*. Для цього був внесений *JavaScript* код для взаємодії з *Janus*. *WebRTC* вміщує відеопотоки з вебкамер за допомогою *JavaScript*, а під час оновлення сторінок всі об'єкти ініціалізуються заново, що викликало б постійне перепідключення к серверу вебсистеми. Для цього можна прийняти рішення про розміщення сторінок в тег *<iframe>*.

Тег *<iframe>* створює фрейм, який знаходиться всередині звичайного документу, він дозволяє завантажувати заданих розмірів будь-які інші нелазжні документи; також він являє собою фрейм, вміст якого ігнорується веббраузером, не підтримуючи даний тег. Для таких браузерів можна вказати альтернативний текст, який відобразиться для користувачів. Він повинен розміщуватись між елементами *<iframe>…</iframe>*. Тепер сторінка стала статичною та нічим не заважає передачі інформації з вебкамер та передачі на сервер. Но і це може привести до проблеми.

У браузері визначена політика безпеки, яка не дозволяє головному вікну маніпулювати *DOM* – елементами *iframe*, якщо він посилається на ресурс з іншого домену. Реалізувати це можна за допомогою функцій *window.postMessage()*.

Window.postMessage() – цей метод дозволяє безпечно відправляти кросдоменні запити. Звичайно для сценріїв на різних сторінках дозволено доступ друг до друга тільки якщо сторінки, які їх виконували, передаються по одному протоколу (зазвичай це *https*), номеру порту (443 - за замовчуванням для *https*) та хосту. *Window.postMessage()* пропонує контрольований механізм щоб обійти ці обмеження способом, який безпечний при правильному використанні. При виборі методу *window.postMessage()* виклакається *MessageEvent.*

MessageEvent – має тип *message*, дані-властивості, які встановлюють значення першого аргументу в методах *window.postMessage().*

2.5. Висновки до розділу

В даному розділі було розглянуто, що таке система прокторингу, її особливості та види. Онлайн-прокторинг – це онлайн-іспит який прив'язаний до часу або ні, під час проходження якого віддалена людина – проктор спостерігає за процесом з метою виявлення шахрайства. Класифікація прокотрингу включає в себе прокторинг у реальному часу – одразу під час написання екзамену проктор спостерігає за процесом; іспити, які перевіряються пізніше на основі відеозаписів та екрану користувача та автоматизований прокторинг особливістю якого є те, що програмне забезпечення контролює процес написання екзамену.

Кожен тип системи прокторингу має як переваги, так і недоліки. Прокторинг, який проходить у реальному часі незручний тим, що користувач та проктор н має можливості для себе в будь-який час скласти/перевірити іспит. Цю проблема вирішує система прокторингу, у який проктор перевіряє вже записи екзамену, які були збережені на сервері. Але в таких системах є проблема в тому, що якщо у студента буде одразу встановлена невірно камера, або поганий кут захвату камери, то проктор не зможе в реальному часі вказати на ці недоліки, а екзамен може бути анульованим. Автоматизовані системи прокторингу частіше піддаються помилкам, тому для цього, проктор після ще перевіряє моменти, на яких система зазначила шахрайство під час іспиту. Також студенти можуть «вивчити» роботу таких систем, оскільки реальна людина-проктор є більш непередбачуваною.

Для полегшення розробки таких систем використовується технологія *WebRTC* та *Janus WebRTC Server*. *WebRTC* використовуються для передачі відео з різних браузерів з мінімальною затримкою. Дана технологія дозволяє працювати з аудіо та відеозв'язком всередині веббразуера. Алгоритм використання цієї технології такий: вебсистема робить запит на *WebRTC API,* тим самим отримує доступ до медіа.

РОЗДІЛ 3
ТЕХНОЛОГІЧНІ ЗАСОБИ ДЛЯ РОЗРОБКИ ВЕБСИСТЕМИ

3.1. Платформа розробки *ASP.NET*

ASP.NET – це платформа веброзробки, яка забезпечує модель програмування, повну інфраструктуру програмного забезпечення та різні послуги, необхідні для створення надійних вебдодатків для ПК, а також мобільних пристроїв.

ASP.NET працює поверх протоколу *HTTP* і використовує команди та політики *HTTP* для встановлення двостороннього зв'язку та співпраці між браузером і сервером.

ASP.NET є частиною платформи *Microsoft .Net.* Вебсистеми *ASP.NET* – це компільовані коди, написані з використанням розширюваних та багаторазових компонентів або об'єктів, що присутні в *.Net framework.* Ці коди можуть використовувати всю ієрархію класів у середовищі *.Net.*

Одна з ключових характеристик *ASP.NET – Code Behind Mode* – це концепція розділення дизайну та коду. Здійснивши це розділення, стає простіше підтримувати програму *ASP.NET.* Загальним типом файлу *ASP.NET* є файл *aspx*. Також ще одна характеристика *State Management – ASP.NET* має можливість контролювати управління станами. Кешування – *ASP.NET* може реалізувати концепцію кешування. Це покращує продуктивність програми. Кешуючи ті сторінки, на які користувач часто робить запит, можна зберігати у тимчасовому місці.

Коди програм *ASP.NET* можуть бути записані будь-якою з наступних мов: *C#, Visual Basic.Net, JavaScript, J #.*

ASP.NET використовується для створення інтерактивних вебдодатків, керування даними, через Інтернет. Він складається з великої кількості елементів керування, таких як текстові поля, кнопки та мітки, налаштування та обробки коду для створення *HTML*-сторінок (рис. 3.1).

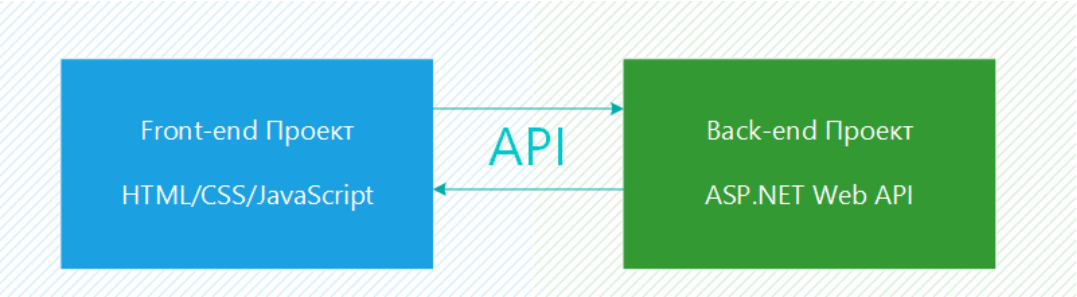

Рис. 3.1. Схема взаємодії компонентів

Модель *ASP.NET Web Forms* поширюють керовану подіями модель взаємодії у вебпрограмі. Браузер передає вебформу серверу, і він у відповідь повертає повну сторінку розмітки або сторінку *HTML*.

Усі дії користувача на стороні клієнта пересилаються на сервер для обробки даних. Сервер обробляє результати дій клієнта і викликає реакції.

HTTP протокол – це протокол без збереження станів. Фреймворк *ASP.NET* допомагає зберігати інформацію про стан програми, яка складається з стану сторінки та стану сесії.

Стан сторінки – це стан клієнта, тобто вміст різних полів для введення інформації у вебформі. Стан сеансу – це загальна інформація, отримана з різних сторінок, які користувач відвідував і на яких працював, тобто загальний стан сеансу. Наприклад, користувач додає товари у кошик. Предмети обираються на сторінці товарів, а загальна кількість обраних товарів і ціна відображаються на іншій сторінці, на сторінці кошику. Але *HTTP* не може відстежувати всю інформацію, що находить на різних сторінках. Стан сеансу *ASP.NET* та інфраструктура на стороні сервера відстежують інформацію яка зібрана глобально за сеанс.

Середа розробки *ASP.NET* передає стан сторінки на сервер і з серверу по запитам сторінок при створенні коду середовища виконання *ASP.NET*, а також включає стан компонентів на стороні сервера у прихованих полях.

Таким чином, сервер дізнається про загальний стан програми та працює способом дворівневого підключення.

Модель *ASP.NET Component* надає різні стандартні блоки сторінок *ASP.NET*. В основному це об’єктна модель, яка описує: серверні аналоги майже всіх елементів *HTML* або тегів, таких як <*form*> та <*input*> та серверні елементи управління, які допомагають у розробці складного інтерфейсу користувача. Наприклад, елемент керування календарем або елемент керування *Gridview.*

ASP.NET – це технологія, яка працює на платформі *.Net*, що містить усі функціональні можливості, пов'язані з Інтернетом. Структура *.Net* складається з об’єктно-орієнтованої ієрархії. Вебсистема *ASP.NET* складається зі сторінок. Коли користувач запитує сторінку *ASP.NET*, *IIS* делегує обробку сторінки системі виконання *ASP.NET*.

Середа виконання *ASP.NET* перетворює сторінку *.aspx в* екземпляр класу, який успадковується від сторінки базового класу платформи *.Net*. Отже, кожна сторінка *ASP.NET* є об’єктом та всі її компоненти, тобто елементи керування на стороні сервера також є об’єктами.

Компоненти *.Net Framework*:

1. *Common Language Runtime* (*CLR*). Він виконує управління пам'яттю, обробку винятків, налагодження, перевірку безпеки, виконання потоків, виконання коду, безпеку коду, перевірку та компіляцію. Код, яким безпосередньо керує *CLR*, називається керованим кодом.

2. Бібліотека класів *.Net Framework*. Вона містить величезну бібліотеку багаторазового використання. Класи, інтерфейси, структури та перелічені значення, які в сукупності називаються типами.

3. Загальномовна специфікація. Вона містить специфікації для підтримуваних мов *.Net* та реалізацію інтеграції мов.

4. Система загальних типів. Містить вказівки щодо оголошення, використання та управління типами під час виконання та міжмовної комунікації.

5. Метадані та збірки. Метадані – це двійкова інформація, що описує програму, що зберігається або у переносному виконуваному файлі (*PE*), або в пам'яті. Збірка – це логічна одиниця, що складається з маніфесту збірки, метаданих типу, коду *IL* та набору ресурсів, таких як файли зображень.

6. Форми *Windows*. Містять графічне зображення будь-якого вікна, що відображається у програмі.

7. *ASP.NET* та *ASP.NET AJAX*. *ASP.NET* – це модель веброзробки, а *AJAX* – розширення *ASP.NET* для розробки та впровадження функціональних можливостей *AJAX*. *ASP.NET AJAX* містить компоненти, які дозволяють розробнику оновлювати дані на сторінці без повного перезавантаження.

8. *ADO.NET*. Це технологія, яка використовується для роботи з даними та базами даних. Вона забезпечує доступ до джерел даних, таких як *SQL*-сервер, *OLE DB, XML* тощо. *ADO.NET* дозволяє підключатися до джерел даних для отримання, обробки та оновлення даних.

9. *Windows Workflow Foundation* (*WF*). Це допомагає створювати додатки на базі робочого циклу в *Windows*. Він містить дії, час роботи, конструктор робочих процесів та механізм правил.

10. *Windows Presentation Foundation*. Забезпечує розділення між користувальницьким інтерфейсом та бізнес-логікою.

11. *Windows CardSpace* забезпечує безпеку доступу до ресурсів та обміну особистою інформацією в Інтернеті.

12. *LINQ* надає можливості запиту даних мовам *.Net* з використанням синтаксису, подібного до традиційної мови запитів *SQL*.

3.2. Життєвий цикл програм з використанням *ASP.NET*

Життєвий цикл системи з використанням *ASP.NET* можна описати так: спочатку *ASP.NET* обробляє сторінки для отримання динамічного виведення, потім програма та її сторінки створюються і оброблюються та *ASP.NET* динамічно компілює сторінки.

Життєвий цикл програм можна розділити на дві групи: життєвий цикл програм та життєвий цикл сторінки.

Життєвий цикл програми має такі етапи:

1. Користувач робить запит на доступ до ресурсу програми, сторінки.

2. Браузер надсилає цей запит вебсерверу.

3. Комунікаційна лінія отримує перший запит і відбуваються такі події: створюється об'єкт класу *ApplicationManager*; об'єкт класу *HostingEnvironment* створюється для надання інформації щодо ресурсів; елементи верхнього рівня в програмі компілюються.

4. Створюються об'єкти відповіді. Об'єкти програми, такі як *HttpContext, HttpRequest* та *HttpResponse*, створюються та ініціалізуються.

5. Екземпляр об'єкта *HttpApplication* створюється та призначається запиту.

6. Запит обробляється класом *HttpApplication*. Цей клас викликає різні події для обробки запиту.

Якщо розглянути життєвий цикл сторінки *ASP.NET* то, коли відбувається запит на сторінку, вона завантажується в пам’ять сервера, обробляється та відправляється в браузер. Потім вивантажується з пам'яті. На кожному з цих кроків доступні методи та події, які можна замінити відповідно до потреби програми. Іншими словами, можна написати власний код, щоб замінити код за замовчуванням.

Клас *Page* створює ієрархічне дерево всіх елементів керування на сторінці. Усі компоненти на сторінці, крім директив, є частиною цього дерева управління. Фазами життєвого циклу сторінки є ініціалізація, екземпляр елементів керування на сторінці, відновлення та підтримка станів, виконання кодів обробника подій, візуалізація сторінки

Розуміння циклу сторінки допомагає писати код для того, щоб щось конкретне сталося на будь-якому етапі життєвого циклу сторінки. Це також допомагає у написанні користувацьких елементів керування та їх ініціалізування у потрібний час, заповнення їх властивостей даними про стан перегляду та запускання коду поведінки елемента керування.

Розглянемо різні етапи сторінки *ASP.NET*. Запит на сторінку – коли *ASP.NET* отримує запит на сторінку, він вирішує, чи слід аналізувати та компілювати сторінку, або це буде кешована версія сторінки; відповідна відповідь надсилається. Початок життєвого циклу сторінки – на цьому етапі встановлюються об’єкти

запиту та відповіді. Якщо запит є старим запитом або відправленням назад, для властивості *IsPostBack* сторінки встановлюється значення *true*. Ініціалізація сторінки – на цьому етапі елементам керування на сторінці присвоюється унікальний ідентифікатор шляхом встановлення властивості *UniqueID* і застосовуються теми.

Для нового запиту завантажуються дані зворотного зв'язку та властивості управління відновлюються до значень стану перегляду. Завантаження необхідні сторінки – на цьому етапі властивості елемента управління встановлюються з використанням значення стану перегляду та стану управління.

Перевірка – викликається метод керування перевіркою, і при його успішному виконанні властивість *IsValid* сторінки має значення *true*. Обробка подій зворотного зв'язку – якщо запит є зворотним зв'язком (старий запит), викликається відповідний обробник події.

Візуалізація сторінки – на цьому етапі переглядається стан сторінки та всі елементи керування зберігаються. Сторінка викликає метод *Render* для кожного елемента керування, і результат виводу відображається в класі *OutputStream* властивості *Response* сторінки.

На кожному етапі життєвого циклу сторінки з’являються деякі події, які можна закодувати. Обробник подій – це в основному функція або підпрограма, прив'язана до події, використовуючи декларативні атрибути, такі як *Onclick* або дескриптор.

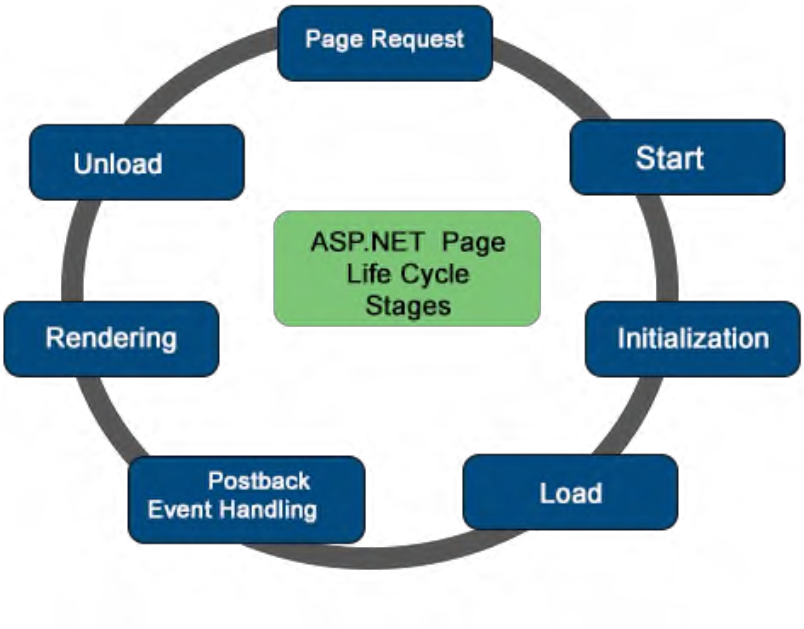

Рис. 3.2. Життєвий цикл сторінки *ASP.NET*

Події життєвого циклу сторінки (рис.3.2):

– *PreInit* – це перша подія у життєвому циклі сторінки. Вона перевіряє властивість *IsPostBack* і визначає, чи є сторінка зворотною передачею, встановлює теми та головні сторінки, створює динамічні елементи управління, отримує та встановлює значення властивостей профілю. Цю подію можна обробити, перевантаживши метод *OnPreInit* або створивши обробник *Page_PreInit.*

– *Init* – подія ініціалізує властивість елемента керування і будується дерево управління. Цю подію можна обробити, перезавантаживши метод *OnInit* або створивши обробник *Page_Init.*

– *InitComplete* – подія дозволяє відстежувати стан перегляду. Усі елементи управління включають відстеження стану перегляду.

– *LoadViewState* – подія дозволяє завантажувати інформацію про стан перегляду в елементи керування.

– *LoadPostData* – на цьому етапі обробляється вміст усіх полів введення з тегом <*form*>.

– *PreLoad* – відбувається до того, як дані завантажуються в елементи керування. Цю подію можна обробити, перевантаживши метод *OnPreLoad* або створивши обробник *Page_PreLoad*.

– *Load* – подія спочатку викликається для сторінки, а потім рекурсивно для всіх дочірніх елементів керування. Цю подію можна обробити, перевантаживши метод *OnLoad* або створивши обробник *Page_Load*.

– *LoadComplete* – процес завантаження завершено, запускаються обробники подій управління та відбувається перевірка сторінки. Цю подію можна обробити, перевантаживши метод *OnLoadComplete* або створивши обробник *Page_LoadComplete*

– *PreRender* – подія відбувається безпосередньо перед тим, як виводяться вихідні дані. Обробляючи цю подію, сторінки та елементи керування можуть виконувати будь-які оновлення перед тим, як вивести результат.

– *PreRenderComplete* – оскільки подія *PreRender* рекурсивно запускається для всіх дочірніх елементів керування, ця подія забезпечує завершення фази попереднього відтворення.

– *SaveStateComplete* – стан контролю на сторінці зберігається. Інформація про персоналізацію, контроль стану та перегляду зберігається. Розмітка *HTML* генерується. Цей етап можна обробити, замінивши метод *Render* або створивши обробник *Page_Render*.

– *UnLoad* – це остання фаза життєвого циклу сторінки. Вона викликає подію *UnLoad* для всіх елементів керування рекурсивно і, нарешті, для самої сторінки. Завершується остаточне очищення та звільняються всі ресурси та посилання, такі як підключення до бази даних. Цю подію можна обробити, змінивши метод *OnUnLoad* або створивши обробник *Page_UnLoad*.

3.3. Архітектурний шаблон *MVC*

Модель-представлення-контролер (*MVC*) – це архітектурний шаблон, який розділяє додаток на три основні логічні компоненти: модель, вигляд та контролер. Це робиться для відокремлення внутрішнього подання інформації від способів подання та прийняття інформації від користувача. Шаблон став популярним з появою вебпрограм. Сьогодні майже всі популярні мови підтримують цю архітектуру.

Модель є центральним компонентом шаблону. Це динамічна структура даних програми, незалежна від інтерфейсу користувача. Це відповідає всій логіці, пов'язаній із даними, з якою працює користувач. Він безпосередньо управляє даними, логікою та правилами програми.

Представленням може бути будь-яке вихідне подання інформації, наприклад діаграма. Можливі декілька представлень однієї і тієї ж інформації.

Третя частина або розділ – контролер, приймає введення та перетворює їх на команди для моделі чи виду. Він діє як інтерфейс між компонентами *Model* і *View* для обробки всієї бізнес-логіки та вхідних запитів, маніпулює даними за допомогою компонента *Model* і взаємодіє з видом, щоб зробити кінцевий результат.

Архітектура *.NET MVC* відповідає традиційній архітектурі *MVC*. *ASP.NET* підтримує три основні моделі розробки: вебсторінки, вебформи та *MVC* (контролер, подання, модель).

ASP.NET MVC ідеально підходить для розробки комплексних, але в той же час легких додатків. Стандарт забезпечує структуру, яка може розширюватись та підключатись, яку можна легко замінити та налаштувати. Наприклад, якщо ви не хочете використовувати вбудований механізм перегляду *Razor* або *ASPX*, тоді ви можете використовувати будь-які інші сторонні механізми перегляду або навіть налаштувати існуючі. Також використовує компонентний дизайн програми шляхом логічного розподілу її на компоненти *Model, View* та *Controller* (рис. 3.3). Це дозволяє розробникам управляти складністю масштабних проектів і працювати над окремими компонентами.

Структура *MVC* покращує тестування та перевірку програми, оскільки всі компоненти можуть бути розроблені на основі інтерфейсу та перевірені за допомогою макетних об'єктів.

ASP.NET MVC підтримує всі існуючі широкі функціональні можливості *ASP.NET*, такі як авторизація та автентифікація, основні сторінки, прив'язка даних, елементи керування для користувача, маршрутизація *ASP.NET* тощо. Дана архітектура використовує концепцію стану перегляду (яка присутня в *ASP.NET*). Це допомагає створювати легкі програми та дає повний контроль розробникам.

Таким чином, можна розглядати *MVC Framework* як основний фреймворк, побудований поверх *ASP.NET*, що забезпечує великий набір додаткових функціональних можливостей, орієнтованих на розробку та тестування на основі компонентів.

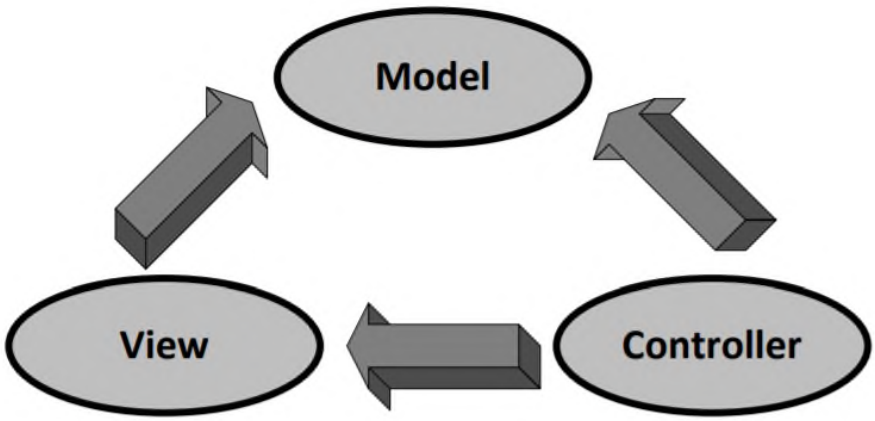

Рис. 3.3. .*NET MVC*

Контролер відображає відповідні представлення та постійно оновлює модель на основі дій користувача. Модель може не обов’язково бути тісно пов’язаною з представленням. Єдина причина, за якою .*NET* дозволяє взаємодіяти між поданням та моделлю, полягає в тому, щоб дозволити проміжні моделі, що називаються *ViewModels*, що відповідають різним представленням та одній моделі.

Структура *ASP.NET MVC* згрупувала взаємодію компонентів вебдодатків за моделями, поданнями та контролерами. Обмін даними між цими компонентами подібний до обміну в шаблоні *MVC*.

Модель відповідає за об'єкти даних та сутності реального світу. В *ASP.NET* модель має додаткові обов'язки підтримувати стан сутностей. Модель реалізує сховище дескрипторів, які використовуються поданнями для запиту стану вебпрограми. Кожен раз, коли відбувається зміна стану, модель оновлює об’єкт сховища, який потім використовується системою.

Представлення відповідає за те, щоб відображати сутності даних зручно для користувача. Воно також приймає до уваги будь-які зміни об’єктів даних та відправляє дані знову у контролер. Представлення являє собою усю вебсистему для користувача и є важливою частиною додатку. Представлення робить запит у сховищі моделі про будь-які зміни стану під час відтворення вебсторінки.

Контролер є точкою входу у вебпрограму, і він безпосередньо взаємодіє з діями користувача. Оскільки контролер має справу переважно з діями користувача, він заповнений кодом *ActionResult*. Кожен *ActionResult* може викликати іншу дію або інший контролер або викликати представлення.

Внутрішні дані в *ASP.NET ActionResults* тісно пов'язані з поданнями. Контролер також управляє оновленням моделі шляхом оновлення об'єктів даних через сховище.

Структура *.NET MVC* стала популярною, оскільки вона базується на добре відомій схемі *MVC*. Вона включає відкриті стандарти, такі як *HTML, XML* та *SOAP*. Ця нова парадигма вебсервісів та додатків спростила веброзробку.

Традиційний шаблон *MVC* не групує дії. Наприклад, розглянемо вебсайт із записом користувачів. Коли користувач редагує свій профіль, результатом буде перенаправлення користувача на його сторінку профілю. Коли користувач реєструється у вебпрограмі, результатом буде перенаправлення користувача на його сторінку профілю. Той самий випадок, коли користувач видаляє свій запис. Ці дії можна було б згрупувати. Розміщення загального коду в спільній зоні зменшило б дублювання. Однак у традиційному *MVC* контролери настільки тісно пов'язані з діями, що часткові дії можуть повторюватися в одному або декількох контролерах. Тому групування цих часткових дій у базовий контролер та виклик його на основі логіки програми було б більш доцільним.

Шаблон контролера сторінок (*page controller pattern*) *ASP.NET MVC* використовує логіку перехоплення та відправлення (рис. 3.4).

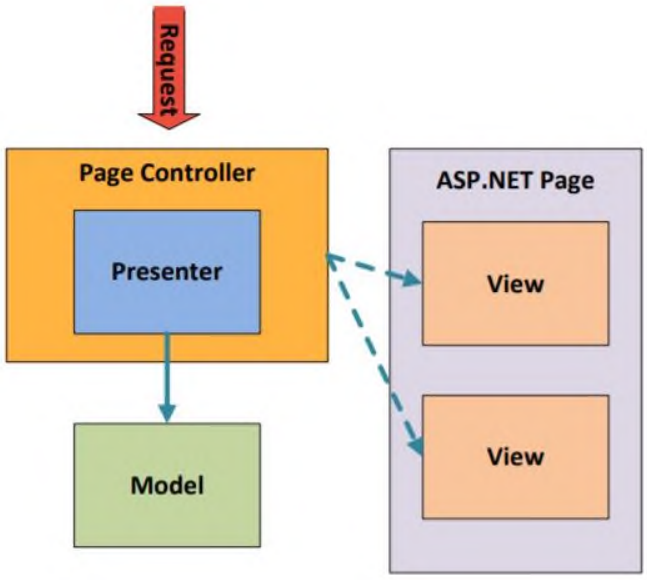

Рис. 3.4. Шаблон контролера сторінок *ASP.NET MVC*

Коли є запит на сторінку, викликається відповідна дія в моделі і відправляється відповідний вигляд. Цей механізм прихований від розробників, і розробники отримують дескриптори для обробки цих подій усередині подання. У кожному поданні розробник може заглянути в код, що стоїть позаду, і вирішити відповідну дію. Метод контролера сторінки – це один із способів відокремити логіку відправлення від логіки подання в контролері. Абстрагуючись від контролера сторінки, дублювання коду значно зменшується. Поширені способи розробки вебдодатка не полягають у тому, щоб співвідносити один до один між контролерами та кожною *URL*-адресою вебпрограми. Це призведе до дублювання коду для кожного запиту. Натомість подібні дії слід згрупувати у базовий клас.

Кроки потоку архітектури *ASP.NET MVC* (рис. 3.5): перший крок – клієнтський браузер надсилає запит до програми *MVC*. Крок 2 – *Global.ascx* отримує цей запит і виконує маршрутизацію на основі *URL*-адреси вхідного запиту, використовуючи об’єкти *RouteTable*, *RouteData*, *UrlRoutingModule* та *MvcRouteHandler*. Крок 3 – ця операція маршрутизації викликає відповідний контролер і виконує його за допомогою об'єкта *IControllerFactory* та методу *Execute*

об'єкта *MvcHandler*. Крок 4 – контролер обробляє дані за допомогою *Model* і викликає відповідний метод за допомогою об’єкта *ControllerActionInvoker*. Крок 5 – оброблена модель потім передається у подання, яке, в свою чергу, робить кінцевий результат.

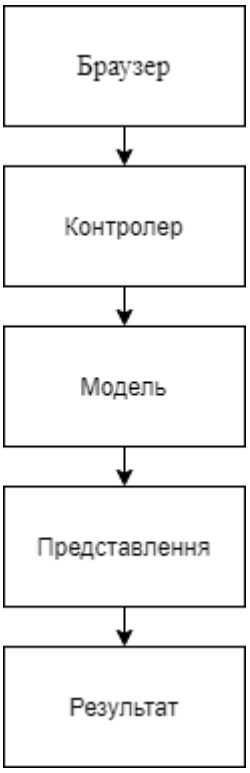

Рис. 3.5. Діаграма потоку *MVC*

3.4. Система керування базами даних

База даних – це окрема програма, в якій зберігаються набори даних. Кожна база даних має один або кілька різних *API* інтерфейсів для створення, доступу, управління, пошуку та тиражування даних, які вона містить.

Також можуть використовуватися інші види сховищ даних, такі як файли у файловій системі або великі хеш-таблиці в пам'яті, але отримання та запис даних не буде настільки швидким та простим у системах такого типу.

В даний час використовуються реляційні системи управління базами даних (СУБД) для зберігання та управління величезним обсягом даних. Це називається реляційною базою даних, оскільки всі дані зберігаються в різних таблицях, а

відносини встановлюються за допомогою первинних ключів або інших ключів, відомих як зовнішні ключі.

Реляційна система управління базами даних (*RDBMS*) – це програмне забезпечення, яке дозволяє реалізувати базу даних з таблицями, стовпцями та індексами, гарантує референтну цілісність між рядками різних таблиць, оновлює індекси автоматично, інтерпретує запит *SQL* та поєднує інформацію з різних таблиць.

MySQL – це швидка та проста у використанні СУБД, яка використовується для багатьох малих та великих підприємств. *MySQL* розробляється, продається та підтримується компанією *MySQL AB*, яка є шведською компанією.

MySQL була побудований як рішення з відкритим кодом, тому її можна налаштувати на стандартному обладнанні та масштабувати з передбачуваними витратами. Широка сумісність *MySQL* також означає, що її можна будь-коли замінити для більш адаптоване рішення (багато конкуруючих баз даних транзакцій сумісні з *MySQL* саме для цієї мети).

Розробники *MySQL* прийняли рішення надати пріоритет швидкості та продуктивності над можливостями, що робить *MySQL* більш швидкою, хоча і більш обмеженою базою даних, ніж інші провайдери в категорії транзакційних баз даних.

Для чого підходить *MySQL*? *MySQL* був в першу чергу розроблений для вебдодатків. Це особливо зручно для структурованих та добре спланованих вебдодатків. Також *MySQL* часто є вибором для невеликих компаній або стартапів через свою надійність, повсюдність та продуктивність. *MySQL* також є привабливим варіантом, оскільки легко створити копію виробничої бази даних *MySQL* для використання в якості аналітичної бази даних.

MySQL використовує стандартну форму добре відомої мови даних *SQL*. СУБД працює у багатьох операційних системах та на багатьох мовах, включаючи *PHP, PERL, C, C ++, JAVA* тощо.

MySQL дуже швидко і добре працює навіть з великими наборами даних. Вона підтримує великі бази даних, до 50 мільйонів рядків і більше в таблиці. За замовчуванням обмеження розміру файлу для таблиці становить 4Гб, але ви можете

збільшити це (якщо ваша операційна система може з цим впоратись) до теоретичного обмеження у 8 мільйонів терабайт (ТБ).

MySQL можна налаштувати. Ліцензія *GPL* з відкритим кодом дозволяє програмістам модифікувати програмне забезпечення *MySQL* відповідно до власних конкретних середовищ.

Рекомендований спосіб встановлення *MySQL* в системі *Linux* – це *RPM*. *MySQL AB* представляє такі пакети *RPM* для завантаження на своєму вебсайті: *MySQL* – сервер баз даних *MySQL* управляє базами даних і таблицями, контролює доступ користувачів та обробляє запити *SQL*; *MySQL*-клієнт – клієнтські програми *MySQL*, які дозволяють підключатися до сервера та взаємодіяти з ним; *MySQL-devel* – бібліотеки та файли заголовків, які знадобляться при компіляції інших програм, що використовують *MySQL*; *MySQL-shared* – спільні бібліотеки для клієнта *MySQL*; *MySQL-bench* – інструменти для тестування продуктивності сервера баз даних *MySQL*.

Встановлення *MySQL* у будь-якій версії *Windows* зараз набагато простіше, ніж було раніше, оскільки *MySQL* тепер поставляється в комплекті з інсталятором. Необхідно просто завантажити пакет інсталятора, розпакувати його, де завгодно і запустити файл *setup.exe*.

Розглянемо список важливих команд *MySQL*, які часто використовуються під час для роботи з базою даних *MySQL*:

– *USE Databasename* –використовується для вибору бази даних у робочому середовищі *MySQL*;

– *SHOW DATABASES* – перераховує бази даних, доступні СУБД *MySQL*;

– *SHOW TABLES* – показує таблиці в базі даних після того, як базу даних було вибрано командою *use*;

– *SHOW COLUMNS FROM tablename* – показує атрибути, типи атрибутів, ключову інформацію, чи нульове поле, за замовчуванням та іншу інформацію для таблиці;

– *SHOW INDEX FROM tablename* – представляє деталі всіх індексів таблиці, включаючи *Primary Key*.

Архітектура *MySQL* складається з декількох рівнів. Прикладний рівень *MySQL* – це те, як різні клієнти підключаються до *MySQL* та відправляють запити. Такі програми, як *MySQL Workbench* і *Looker*, підключаються до прикладного рівня *MySQL*, щоб надсилати запити та контролювати доступ користувачів до бази даних. Після того, як із прикладного рівня видається запит на читання або запис даних із базового сховища, обробник запитів перетворює запит у план запиту, який база даних може виконати.

MySQL є *ACID* сумісною, тобто набір запитів може бути інкапсульований у транзакцію, і всі вони успішні або всі не будуть вдалими. Менеджер транзакцій управляє цим, видаючи команду *COMMIT* для виконання кожної транзакції. Якщо транзакція невдала, менеджер транзакцій виконає команду *ROLLBACK*, щоб скасувати будь-які зміни в базі даних.

MySQL відрізняється високою стійкістю, а менеджер відновлення відповідає за повернення бази даних до останнього стабільного стану в разі аварії. Він реєструє кожну операцію, яка була виконана в базі даних (з моменту її створення), і в разі аварії виконує кожну команду в журналі, таким чином ефективно повертаючи базу даних до останнього стабільного стану.

Менеджер зберігання відповідає за розподіл ресурсів пам'яті, необхідних для вилучення даних з фізичного диска та доставки результатів клієнту.

3.5. Висновки до розділу

В даному розділі було розглянуто платформу для розробки вебсистем *ASP.NET*. Вебсистема *ASP.NET* – це компільовані коди, написані з викоистанням розширювальних та багаторазових компонентів або об’єктів які присутні в *.Net framework. ASP.NET* – це платформа веброзрбки, яка забезпечує модель програмування, повну інфраструктуру програмного забезпечення та різні послуги, які необхідні для створення вебдодатків.

Життєвий цикл програми на платформі *ASP.NET* складається з таких етапів: користувач робить запит; браузер надсилає відповідь на даний запит і відбуваються такі події як створення об’єкту класу *AplicationManager*; створення об’єктів відповіді такі як *HttpContext, HttpRequest* та *HttpResponse*; екземпляр обєекту *HttpAplication* створюється та призначається запиту; запит обробляється даним класом.

Архітектурний шаблон *MVC* ще одна з технологій, яка використана під час розробки вебсистеми. Цей архітектурний шаблон розділяє систему на три частини: представлення, модель та контролер. Представлення відповідає за інтерфейс. Модель – ц прошарок, який забезпечує взаємозв’язок з базою даних та допомагає керувати даними. Контролер – це частина яка відповідає за логіку програми. Цей шаблон є дуже розповсюдженим, оскільки з ним зручно розділити програму на необхідні частини, що забезпечить більш структурований код вебсистеми.

Будь-яка вебсистема не може обійтись без бази даних. У розробці моєї системи було обрано *MySQL* – це швидка та проста у використанні СУБД, яка використовується для багатьох великих та малих підприємств. Дана СУБД використовує стандартну форму дуже відомої мови даних *SQL*. Дана система управління базами даних відрізняється високою стійкістю і може працювати навіть з великими об’ємами даних.

РОЗДІЛ 4
РОЗРОБКА ВЕБСИСТЕМИ ОРГАНІЗАЦІЇ РОЗКЛАДУ ТА ПРОКТОРИНГУ ЕКЗАМЕНІВ

4.1. Середовище розробки, фреймоворки, мови програмування

Після порівняння доступних фреймворків та мов програмування, які можна використовувати для розробки вебсистем, *Microsoft Visual Studio* була обрана як середовище розробки, яка підтримуватиме *ASP.NET Framework* з мовою програмування *C#*.

C# - це об'єктно-орієнтована мова програмування, створена *Microsoft*, яка працює на *.NET Framework*. Один з варіантів її використання це розробка вебсистем на стороні сервера. *ASP.NET* – це фреймворк, розроблений *Microsoft*, який використовується на стороні сервера. З його допомогою можна створювати динамічні вебсторінки. *ASP.NET* використовує *C#* як мову програмування.

Підхід до проектування контролер-представлення-модель (*MVC*), це одна з функцій, пропонованих у *ASP.NET Framework*, може бути використана для розділення моделювання, представлення та логіки програми на три різні рівні. Це допоможе керувати структурою кодування для мого проекту з чітким розділенням даних програми, логіки та правил.

Оскільки *ASP.NET* доповнюється розширеним набором інструментів, що складається з розширених елементів інтерфейсу користувача, це суттєво зменшить кількість часу кодування, яке необхідне для створення багатофункціональної вебсистеми.

Використовуючи *ASP.NET* і *C#* для веб розробки можна легко інтегрувати *HTML5, CSS* та *JavaScript* код. Мова розмітки *HyperText* (*HTML*), каскадні таблиці стилів (*CSS*) та *JavaScript* – це мови, що використовуються у веброзробці. *HTML*

забезпечує базову структуру вебсайтів, яка вдосконалюється та модифікується іншими технологіями, такими як *CSS* та *JavaScript*. *CSS* використовується для управління презентацією, форматуванням і макетом. А *JavaScript* використовується для контролю поведінки різних елементів.

Мова розмітки *HyperText* (*HTML*) може бути розбита на *HyperText*, що надає доступ до інших текстів за допомогою посилань, і розмітка, яка окреслює основну структуру та вигляд необробленого тексту. Це означає, що *HTML* описує та визначає зміст та базову структуру вебсайту. Це робиться за допомогою спеціальних тегів або кодів, які вказують браузеру, що робити. *HTML* – це основа вебсистеми.

Каскадна таблиця стилів – це «аксесуари» вебсайту. Вона відповідає за окреслення кольорів, шрифту та позиціонування вмісту на сторінці. Це додає певного стилю та структури змісту. Для того, щоб скористатися можливостями *CSS*, його потрібно зв'язати з *HTML*. *CSS* повідомляє браузеру, як відображати наявний *HTML*.

Для легшої розробки також було використано *Bootstrap*. Це потужний набір інструментів – колекція інструментів *HTML, CSS* та *JavaScript* для створення вебсторінок та вебдодатків. Вебдизайнери та веброзробники люблять *Bootstrap,* оскільки він гнучкий і простий у роботі. Основні його переваги в тому, що він адаптивний до дизайну, підтримує широку сумісність з браузерами, дуже простий у використанні та швидкий у навчанні. Він пропонує широку розширюваність за допомогою *JavaScript*, оснащений вбудованою підтримкою плагінів *jQuery* та *API JavaScript*. *Bootstrap* можна використовувати з будь-якою *IDE* або редактором, а також будь-якою технологією та мовою на стороні сервера, від *ASP.NET* до *PHP* до *Ruby on Rails*.

MySQL Workbench була обрана як захищена база даних для управління даними студентів, університетів та іспитів, оскільки вона часто оновлюється

функціями для покращеннями безпеки з великим акцентом на швидкості та надійності. Ще однією причиною вибору *MySQL Workbench* є те, що вона забезпечує інструментами моделювання даних та комплексні інструменти для адміністрування користувачів, резервного копіювання тощо.

4.2. Вимоги до вебсистеми

У будь-якому програмному проекті збір вимог розглядається як перша фаза процесу розробки програмного забезпечення. Перелік вимог (функціональних, нефункціональних, технічних тощо) зазвичай формується від різних зацікавлених сторін, що беруть участь у проекті.

Так само для мого вебпроекту було зібрано різні типи вимог, з урахуванням користувача системи та самої системи, для моделювання структури веб-системи.

Функціональні вимоги – це, як правило, заявки на високому рівні про функції, які повинна надавати система, як система повинна реагувати на конкретні вхідні данні та як система повинна поводитися в конкретному та/або заданому стані. Основні функціональні вимоги вебсистеми, розробленої для управління іспитами, які були визначені, представлені нижче (табл. 4.1).

Таблиця 4.1

Функціональні вимоги до системи

Вимога	Користувач
Користувач повинен мати змогу отримати реєстраційні дані разом із реєстраційними даними веб-системи після успішного завершення реєстрації	Адміністратор
Користувач повинен мати можливість змінити та скинути свій пароль	Студент Адміністратор

Закінчення таблиці 4.1

Користувач повинен мати можливість переглядати всіх студентів, які зареєстровані	Адміністратор
Користувач повинен мати можливість фільтрувати студентів за їх курсом	Адміністратор
Користувач повинен мати можливість видалити обліковий запис студента	Адміністратор Студент
Користувач повинен мати можливість додавати графіки іспитів до вебсистеми	Адміністратор
Користувач повинен мати можливість переглядати графіки майбутніх іспитів у вебсистемі	Адміністратор
Користувач повинен мати можливість зв’язатися з університетом, надіславши запит	Адміністратор Студент
Користувач повинен мати можливість перейти до місця іспиту	Студент
Користувач повинен мати можливість розрахувати загальну оцінку з дисципліни	Студент
Користувач може скласти онлайн-іспит	Студент
Користувач може переглянути відео файли написання екзамену	Адміністратор
Кожен користувач, що використовує вебсистему, повинен бути ідентифікований своїм 9-значним номером користувача із сумішшю цифр та літер	Адміністратор Студент
Кожному користувачеві надається роль, щоб однозначно визначити, студент він чи адміністратор	Адміністратор Студент
Користувач повинен бути повідомлений про оновлення облікового запису та скидання пароля електронною поштою	Студент

Нефункціональні вимоги, як правило, фокусуються на тому, наскільки ефективно працює система, включаючи час відгуку, обсяги даних та міркування щодо безпеки. Кажуть, що нефункціональні вимоги є більш критичними, ніж функціональні вимоги. Нефункціональний – це також тип вимоги, який визначає умову/критерії, за якими можна судити про роботу системи, а не про поведінку системи.

Основні нефункціональні вимоги вебсистеми, розробленої для управління іспитами студентів:

1. Дані користувача, такі як особисті дані студентів, позначки їх модулів та деталі, а також адміністраторські дані повинні зберігатися в захищеній базі даних, застосовуючи відповідні методи шифрування проти *SQL Injection.*

2. Дані як були введені користувачем у всіх вебформах повинні бути належним чином перевірені за допомогою відповідних регулярних виразів.

3. Вхідні дані користувача у форму для входу повинні бути перевірені шляхом перевірки введених даних щодо запису, що зберігається в базі даних.

4. Одна і та ж адреса електронної пошти не може використовуватися в кількох дисциплінах студентів та реєстрація вебсистеми.

5. Пароль користувача слід правильно хешувати, використовуючи відповідний алгоритм хешування перед тим, як зберігати його в базі даних.

6. Функція веб-системи, така як додавання розкладу студента та іспиту, повинна бути доступна лише адміністратору.

7. Користувач (студент) повинен увійти в систему, щоб побачити розклади майбутніх іспитів.

8. Користувач (студент) повинен увійти в систему, щоб використовувати функцію розрахунку оцінок.

9. Студенту надсилається електронне повідомлення як підтвердження реєстрації, а також у випадку оновлення облікового запису та пароля.

10. Користувач (студент) повинен бути точно переведений до місця іспиту за допомогою геолокації, що надана *Google API.*

11. Користувач повинен бути перенаправлений на домашню сторінку вебсистеми не менш ніж за 2 секунд після успішного входу в систему.

У будь-якій вебсистемі зручність використання системи є ключовим фактором, який повинна забезпечувати система. Вимоги щодо зручності використання – це задокументовані очікування та специфікації, розроблені для забезпечення простоти використання продукту, послуги, процесу або середовища.

Основні вимоги щодо зручності використання вебсистеми:

1. Користувач повинен автоматично перейти на домашню сторінку веб-системи зі сторінки входу після успішного входу в систему.

2. Користувач (студент) повинен мати можливість переглядати навігаційні меню на навігаційній панелі.

3. Користувач (адміністратор) повинен мати можливість переглядати навігаційні меню на навігаційній панелі, такі як «Мій обліковий запис», «Додати студента», «Додати іспит», «Показати іспити», «Переглянути студентів» та «Вийти» після того, як він / вона успішно авторизована.

4. Вебсистема повинна бути адаптованою до різних роздільних здатностей екрану.

5. Вебсистема повинна бути сумісною з платформою, щоб вона працювала та відображала вебвміст однаково у всіх веббраузерах.

4.3. Створення інтерфейсу та основні етапи розробки системи

Діаграми уніфікованих мов моделювання (*UML*) використовувались для моделювання структур, поведінки та взаємодії програми.

«Картинка коштує тисячі слів» - ця ідіома абсолютно відповідає опису *UML*. Об'єктно-орієнтовані концепції були введені набагато раніше, ніж *UML*. На той

момент не існувало стандартних методологій для організації та консолідації об'єктно-орієнтованого розвитку. Саме тоді *UML* з’явився в картині.

Існує низка цілей для розробки *UML*, але найголовніше – це визначити якусь мову моделювання загального призначення, якою можуть користуватися всі моделісти, а також її потрібно спростити для розуміння та використання.

Діаграми *UML* створені не тільки для розробників, але й для ділових користувачів, простих людей та всіх, хто зацікавлений у розумінні системи. Система може бути програмною або непрограмною системою. Таким чином, повинно бути зрозуміло, що *UML* не є методом розробки, а супроводжує процеси.

Мету *UML* можна визначити як простий механізм моделювання для створення всіх можливих практичних систем у сучасних складних умовах. Концептуальна модель – це перший крок перед складанням діаграми *UML*. Це допомагає зрозуміти сутності в реальному світі та те, як вони взаємодіють між собою. Оскільки *UML* описує системи реального часу, дуже важливо скласти концептуальну модель, а потім поступово діяти.

Наведена нижче діаграма (рис. 4.1) – це схема класу, яка моделює всю вебсистему для ефективного управління іспитами студентів. Класи пов'язані між собою за допомогою різних видів спеціальних відносин.

Діаграма класів - це статична діаграма. Він представляє статичний вигляд програми. Діаграма класів використовується не тільки для візуалізації, опису та документування різних аспектів системи, а й для побудови виконуваного коду програмного додатку.

Діаграма класу описує атрибути та операції класу, а також обмеження, накладені на систему. Діаграми класів широко використовуються при моделюванні об'єктно-орієнтованих систем, оскільки це єдині діаграми *UML*, які можна безпосередньо зіставити з об'єктно-орієнтованими мовами.

Діаграма класів показує набір класів, інтерфейсів, асоціацій та обмежень. Мета діаграми класів – моделювати статичний вигляд програми.

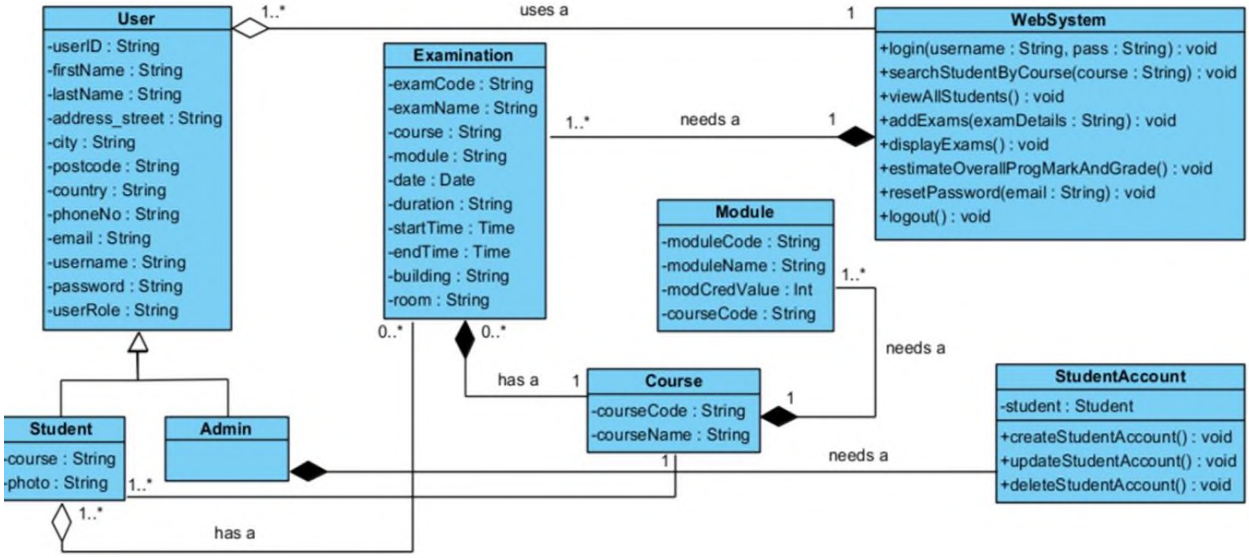

Рис. 4.1. Діаграма класів системи

Перший крок розробки був зосереджений на впровадженні вебформи, що дозволяє адміністратору вводити особисті дані студента, щоб зареєструвати студента. Крім того, була також реалізована вебформа, що дозволяє користувачам входити до вебсистеми. Елементи та функції інтерфейсу *ASP.NET,* такі як текстові поля, кнопки, мітки, випадаючі списоки тощо, використовувались для налаштування вебформ разом із властивостями *Bootstrap* та *CSS*.

Visual Studio .NET полегшує перевірку даних форми, оскільки різноманітні засоби контролю перевірки можуть бути додані та використані у вебформі, що запобіжить надсилання даних неправильного типу. Під час впровадження вебформ для реєстрації та для дозволу користувачам входити в систему до вебформ додавались різні типи валідаторів для перевірки введених користувачем даних.

Наприклад, валідатор порівняння було використано для перевірки пароля шляхом порівняння пароля, введеного користувачем у два різні текстові поля, тобто пароль та підтвердження пароля. Подібним чином, валідатор використовувався для перевірки вхідних даних, щоб переконатися, що вхідні дані мають допустимий тип даних. Нарешті, для перевірки вхідних даних у формі було

використано обов’язковий валідатор поля, перевіривши, чи містить поле форми якесь значення.

```
protected void loginBtn_Click(object sender, EventArgs e)
{
    bool test;  //a boolean variable to store either true or a false.

    String username = usernameTxt.Text;
    String password = passwordTxt.Text;

    //hashing the password and storing it into a new local variable.
    String passwordEncrypt = FormsAuthentication.HashPasswordForStoringInConfigFile(password, "SHA1");

    if (charactersTxt.Text.Equals(charsToValidateLbl.Text))
    {
        //calling a authenticateLogin function of class Login and then providing the inputs
        //entered by a user to the parameters.
        test = userLogin.authenticateUsersViaDatabase(username, passwordEncrypt);

        if (test && userLogin.getUserRole(username).Equals("student"))
        {
            Session["username"] = username;     //creating a session variable for username.
            string userID = userLogin.getStudentID(username);
            Session["userID"] = userID;      //creating a session variable for student ID.
            //System.Web.Security.FormsAuthentication.RedirectFromLoginPage(usernameTxt.Text, false);
            Response.Redirect("Default.aspx");
        }
        else if(test && userLogin.getUserRole(username).Equals("admin"))
        {
```

Рис. 4.2. Створення функціоналу для входу в систему

Іншим важливим завданням – було реалізація поштового сервера для надсилання електронних листів студентам із підтвердженням їх реєстрації, а також облікових даних для входу, створених для студентів адміністратором.

Після того, як продукт з мінімальними функціями зроблено, які орієнтовані на вебфункції реєстрації та функції входу, пріоритет був наданий реалізації однієї з найважливіших функцій вебсистеми – того, щоб адміністратор міг додавати та оновлювати графіки іспитів в Інтернет-системі, завдяки якій студенти зможуть переглядати розклади майбутніх іспитів своїх дисциплін, включаючи деталі, такі як час, тривалість, місце проведення та інше.

Для цього створена вебформа, що дозволяє адміністратору вводити деталі екзамену, такі як: модуль, дата, тривалість, час та місце проведення, а також додані різні типи валідаторів для перевірки введених даних. Також був створений зручний інтерфейс для відображення розкладу іспитів з використанням *Bootstrap* та *CSS*.

Для цілей цієї вебсистеми важливо було врахувати, що не всі функції вебсистеми можуть бути доступними для всіх типів користувачів. Тобто деякі функції можуть бути використані як і адміністратор, так і студент, тоді як до деяких

доступ може отримати лише адміністратор, а до деяких – лише студент. Тому було важливо, щоб функції вебсистеми були обмежені певними типами користувачів. Це було досягнуто шляхом створення змінної сеансу (рис. 4.3) для імені користувача, як тільки він входить до системи.

Відомо, що змінна сеансу використовуються для зберігання інформації користувача, до якої потрібно отримати доступ на декількох сторінках вебпрограми. Проведено перевірку на кожній вебсторінці під час її завантаження, щоб з'ясувати, чи авторизувався до системи якийсь користувач та визначити тип користувача. Користувач буде перенаправлений на сторінку входу, якщо він спробує отримати доступ до певної вебсторінки без входу в систему.

```
Ссылок: 0
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["adminUsername"] == null && Session["username"] == null)
    {
        Response.Redirect("login.aspx");
    }
    else if (Session["username"] != null)
    {
        Response.Redirect("Default.aspx");
    }
}
```

Рис. 4.3. Використання змінної сеансу

Впровадження оцінювача оцінок студентів було одним з завдань на останньому етапі, яке включало створення вебформи, що дозволяє студентам вводити свої оцінки, а також включало розробку математичної формули для розрахунку загальної очікуваної оцінки. Також функція адміністратора для перегляду студентів, зареєстрованих у вебсистемі, фільтруючи їх за курсами, також була реалізована з додатковими привілеями для перегляду, оновлення та видалення облікових записів студентів.

4.4. База даних *MySQL* та захист даних

Для збереження та захисту інформації у вебсистемі було використано базу даних *MySQL*. Вирішити питання про створення відповідних таблиць бази даних, що відповідають типам даних, що зберігаються в базі даних, було складним завданням. Такі таблиці, як "Студенти", "Курси", "Модулі" та "Іспити" були створено та пов'язано між собою за необхідності за допомогою первинного та зовнішнього ключів (рис. 4.4).

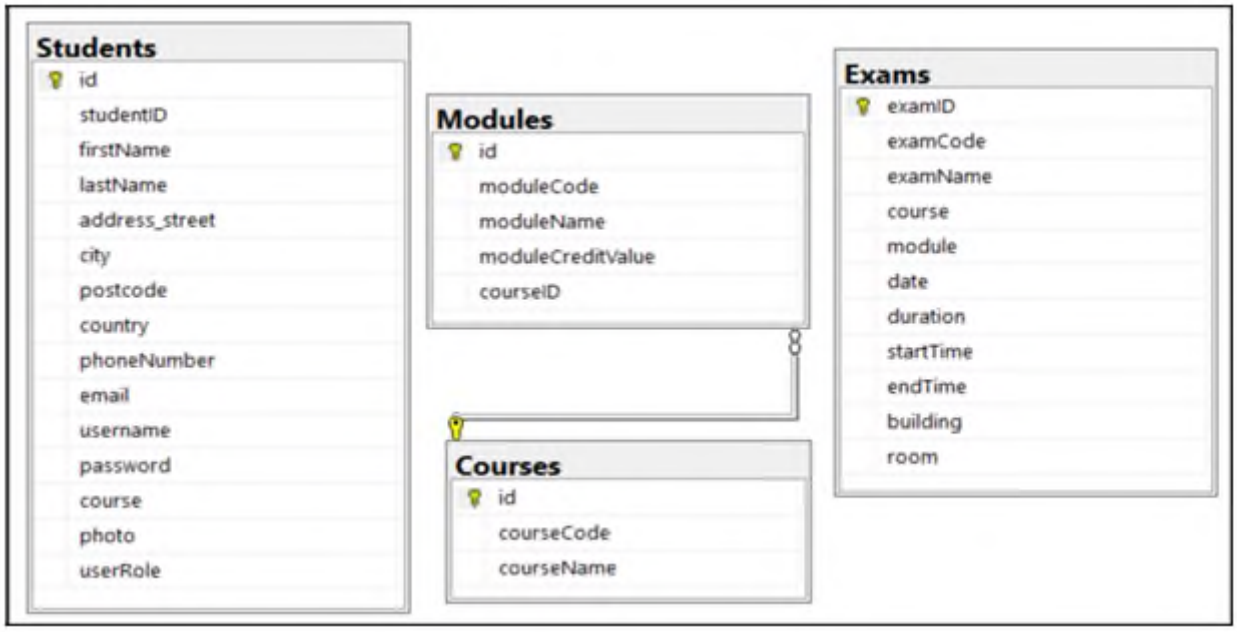

Рис. 4.4. Представлення таблиць в базі даних та їх зв’язки

Зважаючи на те, що система, яка розроблюється, заснована на веброзробці, вкрай важливо, щоб всі ключові аспекти та вимоги системи було продумано та розглянуто з урахуванням безпеки. Зловмисники можуть зробити все можливе, щоб використати програму, щоб порушити безпеку програми. Тому безпека розглядалася майже у всіх аспектах вебсистеми.

Валідатор використовувався у всіх вебформах для перевірки тексту, введеного користувачем у текстові поля, з діапазоном регулярних виразів для різних типів вхідних даних: адреси, номера телефону, електронної пошти, дати тощо. Валідатор використовується для забезпечення захисту програми від

некоректних вхідних даних та перевірки всіх вхідних даних, можна уникнути найпоширеніших атак, таких як *SQL Injection* та *Butter Overflow.*

Елементи контролю перевірки *ASP.NET* перевіряють введені користувачем дані, щоб переконатися, що марні, неаутентифіковані або суперечливі дані не зберігаються. *ASP.NET* надає такі засоби перевірки: *RequiredFieldValidator, RangeValidator, CompareValidator, RegularExpressionValidator, CustomValidator, ValidationSummary*. Класи контролю перевірки успадковуються від класу *BaseValidator*. Принцип роботи валідотора у вебсистемі зображено на рис. 4.5.

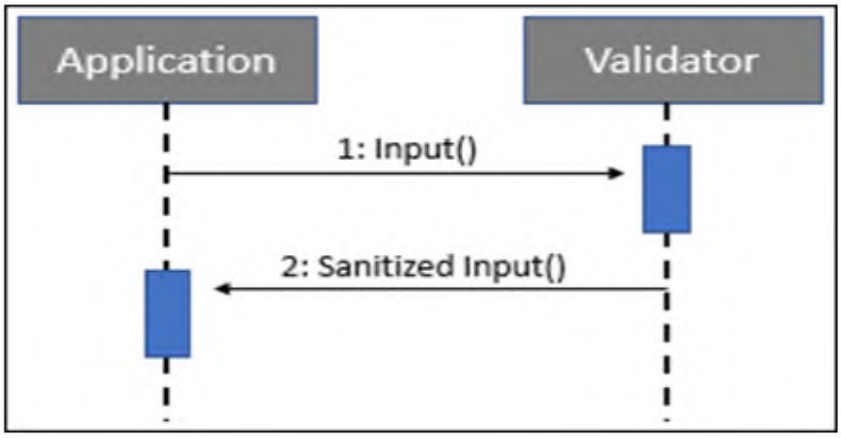

Рис. 4.5. Робота валідатора

Зазвичай конфіденційні дані користувача, включаючи його особисту інформацію та облікові дані для входу, зберігаються у вигляді звичайного тексту у базі даних. У випадку, коли зловмисник потрапляє в базу даних, увесь обсяг даних, що зберігається компрометуються, що може бути використано для різних цілей.

Пароль розглядається як найбільш конфіденційна інформація з усіх, що зберігаються в базі даних. Тому було важливо, щоб відповідний алгоритм хешування використовувався для шифрування пароля, шляхом перетворення його в хешований пароль, а потім, нарешті, зберігання хешованого пароля в базі даних (рис. 4.6).

Рис. 4.6. Процес шифрування даних

Хешування може бути корисним для безпечної автентифікація. Конфіденційні дані, що зберігаються в базах даних, можна хешувати. Завдяки детермінованій властивості хешування дані, такі як паролі або особиста інформація, можуть бути зіставлені, не демонструючи ніде фактичних даних.

Наприклад, при введенні пароля в базу даних для входу система зберігає у своїй базі лише хеш пароля. Після того, як захочете увійти та ввести свій пароль, це спочатку хешується. Це, на основі детермінованого принципу, створить той самий хеш. Потім вони збігаються, і отримуєься доступ.

Хешування також використовується для перевірка транзакції. Транзакції можна хешувати, щоб скоротити посилання та полегшити процес. Будь-яка зміна транзакції (навіть найменша) також змінить хеш. Це захищає оригінальну транзакцію і фіксує всі зміни, внесені в блокчейн. Потім транзакції називаються їх хешем, і саме таким чином вони також зберігаються на блокчейні.

Індексація – коли блокчейн розгорнуто для використання в системі баз даних, елементи бази даних можуть хешуватися і зберігатися на блоках. Тоді легше знайти короткі хеш-номери та отримати необхідні дані, ніж шукати вихідні значення. Знову ж таки, цей підхід використовує основні принципи хешування.

4.5. Робота користувача з додатком

Для запуску вебсистеми користувачем необіхдно відкрити посилання. А на етапі розробки – це *localhost (*рис. 4.7).

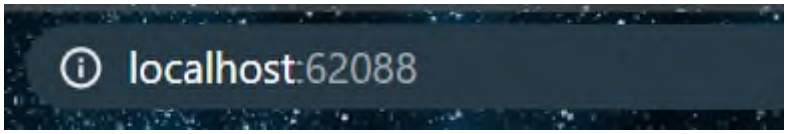

Рис.4.7. Відкриття вебсистеми у браузері

Після того як вебсистема завантажиться, відкриється головна сторінка. На ній присутнє меню системи, яке відображене за допомогою вкладок «Вхід до системи» та «Контакти» (рис. 4.8).

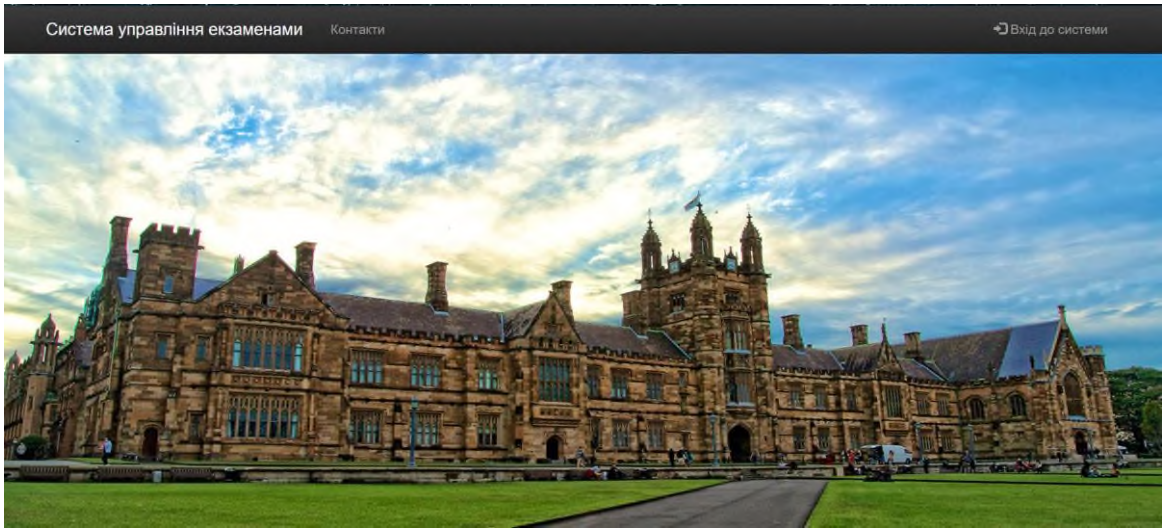

Рис.4.8. Головна сторінка

На сторінці контакти зберігається інформація про університет та контактні дані. За допомогою входу до системи можна авторизуватися як адмін або систем. Форма входу до системи зображена на рис. 4.9.

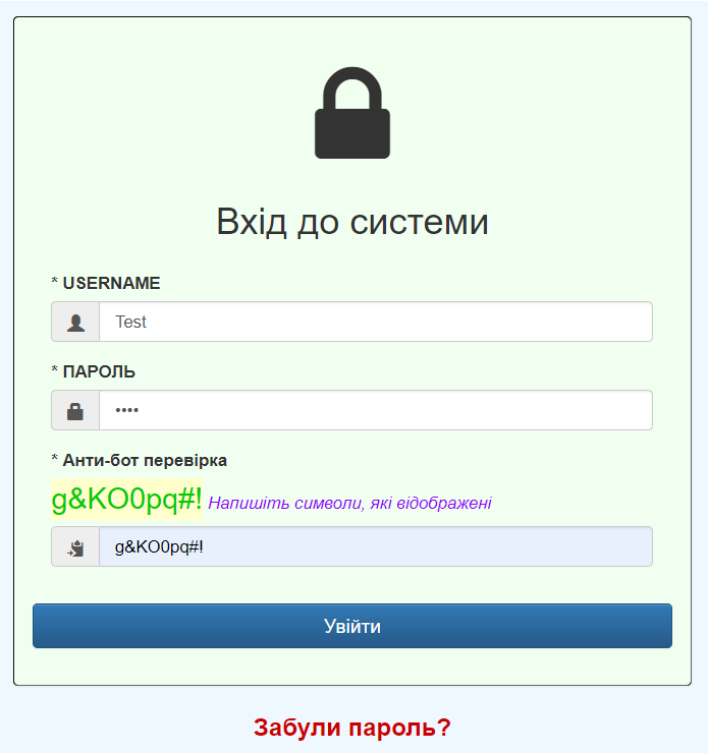

Рис. 4.9. Вхід до системи

Якщо авторизуватися до системи як адміністратор, то буде наданий доступ до таких функцій як: додати або показати екзамен, перегляд студентів та можливість додати нового студента до системи (рис. 4.10).

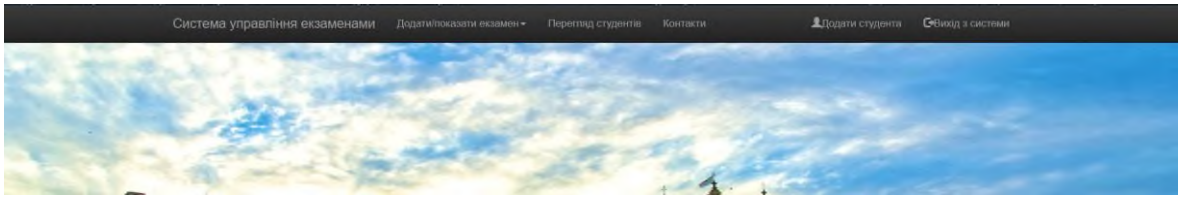

Рис. 4.10. Панель керування для адміністратора

Адміністратор може заповнити форму, щоб додати нового студента до системи, після чого передати ці данні йому. Для її заповнення необхідно внести наступні дані: ім’я та прізвище студента, його адресу, номер телефону та електронну пошту, придумати імя користувача для входу в систему та додати пароль, після чого користувач може редагувати свої дані (рис. 4.11).

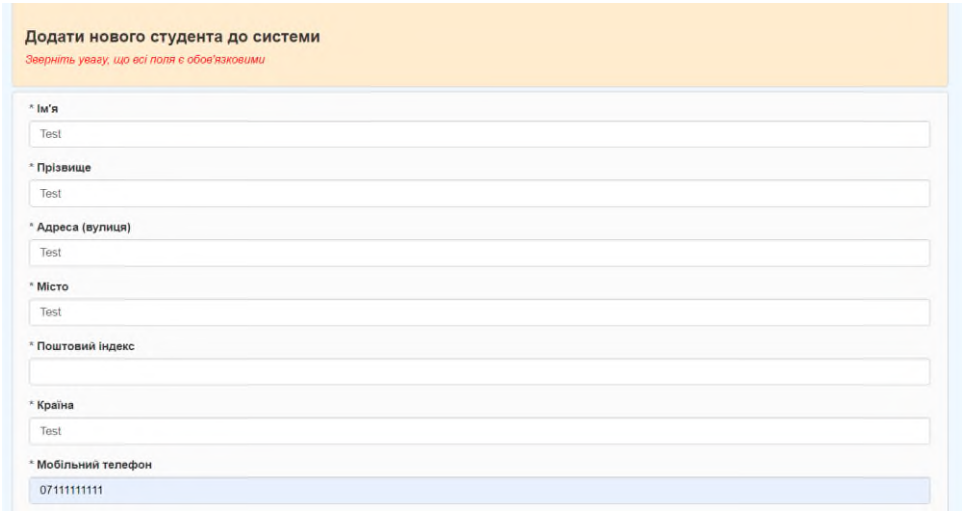

Рис. 4.11. Форма створення нового користувача (студента)

Аналогічно адміністратор може додати дисципліну, вказавши назву екзамену та його код, обрати курс та модуль, дату та тривалість екзамену, вказати чи онлайн він буде проведений або в аудиторії, якщо друге, то також слід вказати номер аудиторії. Коли адміністратор заповнює ці дані, для студента формується графік проведення екзаменів.

Розглянемо систему зі сторони студента. Якщо авторизуватися до системи з правами системи у верхньому меню (рис. 4.12) можна отримати доступ до таких функцій: переглянути всі екзамени, розрахувати загальну оцінку, доступ до власного профілю.

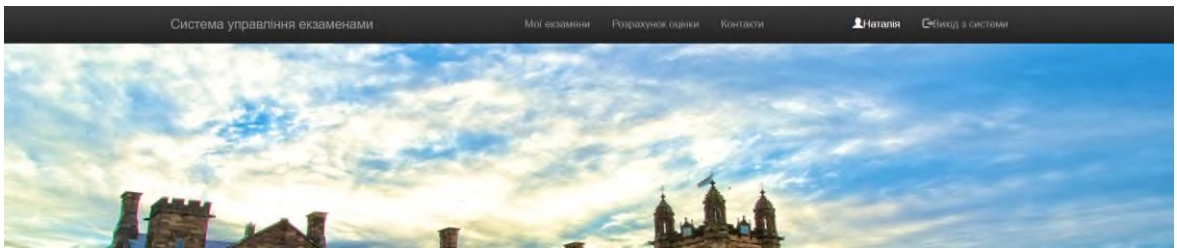

Рис. 4.12. Панель керування для студента

Натиснувши на власне ім’я поруч з іконкою користувача, відкриється віно з інформацію про студента (рис. 4.13). У користувача-студента є можливість змінювати інформацію про себе, лише поля ім’я, прізвище та ім’я для авторизації у систему недоступні для зміни.

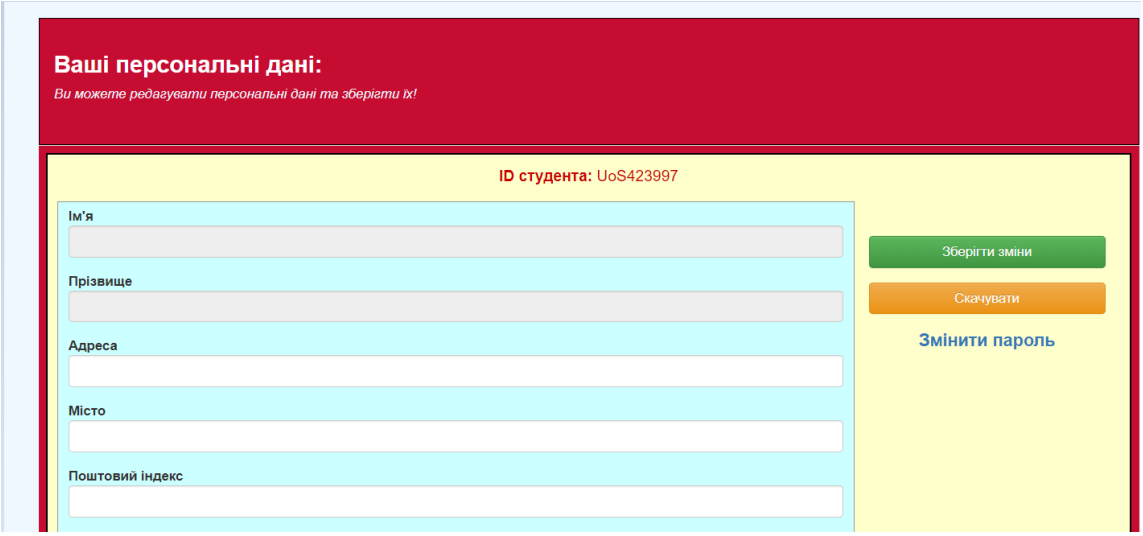

Рис. 4.13. Екран редагування особистої інформації

Якщо відкрити вкладку з екзаменами, то буде відображено найближчі екзамени (рис. 4.14) відфільтровані за датою. На цій формі користувач може побачити необхідно інформацію про екзамен: код екзамену, тема модулю та назва дисципліни з якої буде проведено екзамен.

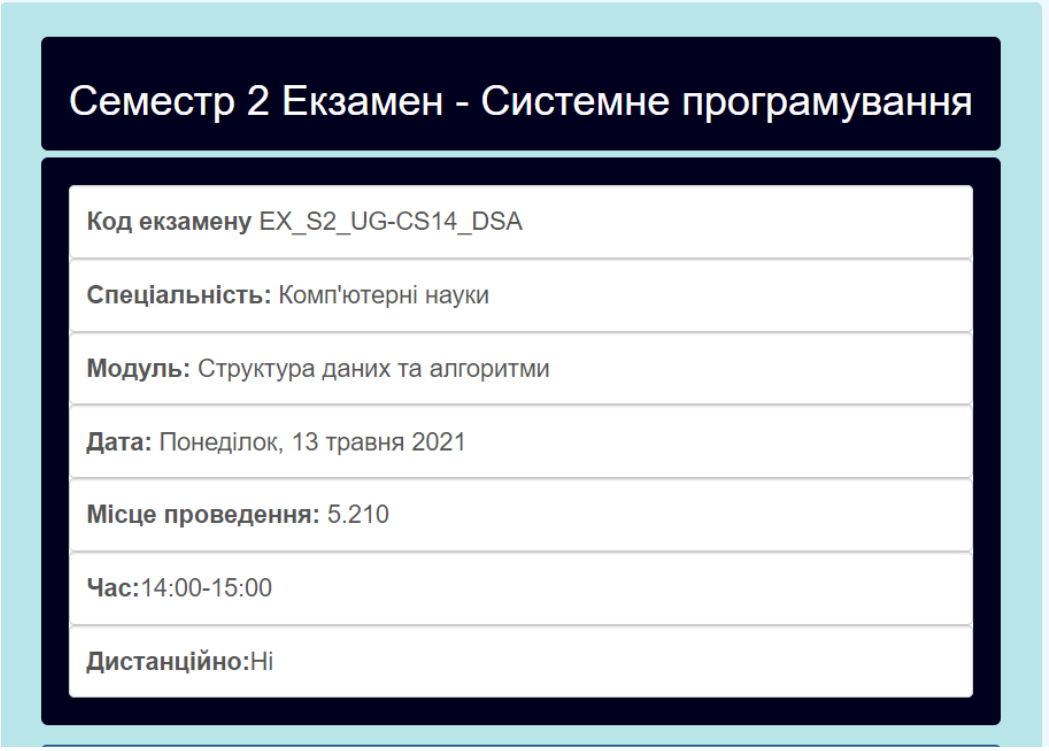

Рис. 4.14. Форма відображення інформації про екзамен

4.6. Тестування

Вебтестування – це перевірка вебсистеми на наявність потенційних помилок до того, як ця система стане доступною для широкої публіки. Вебтестування перевіряє функціональність, зручність використання, безпеку, сумісність, ефективність вебпрограми.

На цьому етапі перевіряються такі питання, як безпека вебдодатків, функціонування сайту тощо.

Інтеграційне тестування – це тестування класу або компонента з його зовнішніми залежностями. Він перевіряє інтеграцію кодів додатків із конкретними залежностями, такими як файли, бази даних, вебслужби тощо. Кажуть, що інтеграційні тести виконуються довше, оскільки вони часто передбачають читання та запис у базу даних.

Перша функція – це фукція перевірки логіна та пароля користувача, що зберігаються в базі даних. Простий текстовий пароль для імені користувача зазначено – «*test*». Цей тест повинен пройти (рис. 4.15), оскільки введені ім’я користувача та пароль правильні і повинні відповідати запису, що зберігається в базі даних

Рис. 4.15. Результати проведення тесту

Аналогічним способом було проведено ще декілька тестів. Друга функція (*Check_GetUserRole_ByUserName*) яка реалізована – тестує повернення ролі користувача, коли вказано ім’я користувача (рис. 4.16). Ім'я користувача надано

"*teststudent*", який повинен повернути роль користувача – студент. Однак для вказаного імені користувача очікується роль користувача – адміністратор. Тест стверджує, що очікуваний і фактичний результат не однакові.

Рис. 4.16. Тест *Check_GetUserRole_ByUserName*

Функція перевірки електронної пошти користувача (*check_userEmail*), що зберігається в базі даних, тестується, щоб перевірити, чи існує вона. Надана електронна адреса вказана неправильно, тому це не є дійсною адресою електронної пошти та не існує в базі даних. Тест стверджує, що результат хибний, оскільки наданий електронний лист не існує у базі даних.

Функція повернення студентського ідентифікатора (*Check_StudentId*), коли вказано ім’я користувача студента. Ім'я користувача надано «*teststudent*», який має студентський ідентифікатор - «*UoS930121*». Згідно з твердженням тесту, очікуваний та фактичний результат однакові, тому цей тест повинен пройти.

Також тестується функція повернення ролі користувача, коли вказано ім’я користувача. На відміну від інших тестів, цей тест є прикладом відмови. Оскільки ім'я користувача надано «*Teststudent*», який повинен повернути роль користувача – студент. Однак для вказаного імені користувача очікується роль користувача – адміністратор (рис. 4.17).

Рис. 4.17. Перевірка коректного повернення ролі користувача

Також було використано модульне тестування. Це тестування окремого програмного компонента або модуля, яке зазвичай виконують програмісти, а не тестери, оскільки що потрібні детальні знання про внутрішню розробку програми та код. Індивідуальні методи та функції класів перевіряються без будь-яких зовнішніх залежностей, таких як файли, бази даних, вебслужби тощо.

Наприклад один із тестів використаний для розрахунку оцінок з певної дисципліни (рис. 4.18) – випробовується функція повернення суми найкращих 5 балів із 6, досягнутих за другий рік. Функція додає всі позначки в масиві і віднімає із суми найнижчу позначку в масиві. У тесті сума всіх оцінок у масиві повинна бути 21, а сума найкращих 5 балів повинна бути 20. Очікуваний результат - 20, а фактичний – також 20.

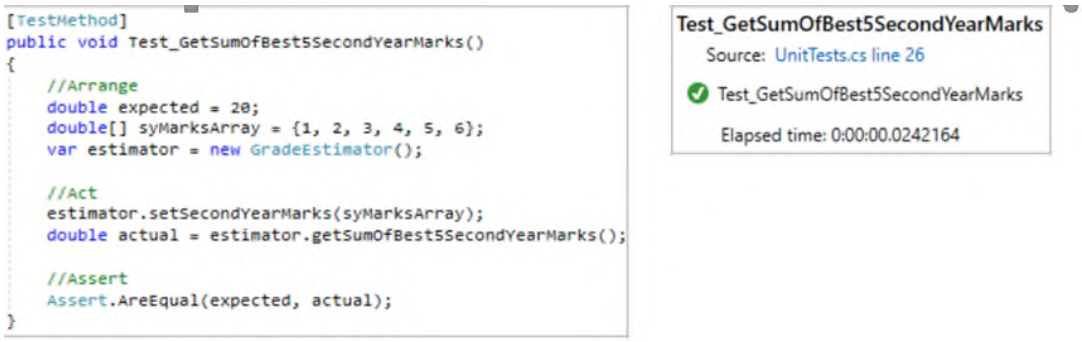

Рис. 4.18. Модульне тестування

4.7. Висновки до розділу

У даному розділі було описано процес розробки вебсистеми для управління розкладом та прокторингу екзаменів. Для створення вебсистеми було обрано *Microsoft Visual Studio* як середовище розробки, яке підтримуватиме *ASP.NET Framework* з мовою програмування *C#*. Також було обрано шаблон *MVC*, який виділяє модель (взаємодія з данними), контролер (опис логіки в системі) та представлення (відображення даних для користувача). Для візуальної частини обрано *HTML, CSS*.

Одним з важливих етапів створення вебсистеми – це виділення вимог до системи. Вони описуються основні функції та можливості системи, які необхідно розробити. Було виділено функціональні вимоги до моєї вебсистеми та описано у таблиці, а конкретно розглянуто можливості розробленої системи та який тип користувачів має доступ до кожного з функціоналу. Також розглянуто нефункціональні вимоги до системи – вони визначають умову, критерії за якими можна судити про роботу системи.

Також в даному розділі був описаний процес та основні етапи створення вебсистеми для управління розкладом та прокторингу екзаменів, проведено тестування розробленої системи та описано роботу користувача з вебсистемою.

ВИСНОВКИ

В рамках дипломної роботи було проведено дослідження про розвиток інтернет-технологій для створення вебсистем та розроблено вебсистему організації розкладу та прокторингу екзаменів в навчальному закладі.

Під час виконання дипломної роботи було:

− сформулювано вимоги до створюваної вебсистеми організації розкладу та прокторингу екзаменів;

− організовано збереження даних, таких як особисті дані студентів, позначки їх модулів, деталі курсу та дані адміністратора, ефективно та безпечно, створюючи захищену базу даних та застосовуючи відповідні методи шифрування;

− описано функціональні характеристики вебсистеми за допомогою таблиць та діаграм *UML*;

− розроблено прототип та каркас для вебінтерфейсу для забезпечення включення та досягнення функцій, визначених на етапі вимог користувача, а також для визначення логічної навігації вебсистеми;

− визначено та обрано технології, платформу та середовище розробки вебсистеми шляхом дослідження найбільш часто використовуваних технологій та середовища розробки для створення вебсистем;

− розроблено вебсистему із використанням макету з урахуванням бажаних вимог користувача шляхом ретельного тестування вебпрограми в кінці кожного кроку, а також на завершальній стадії розробки.

Проаналізувавши їх для створення власної системи було обрано платформу *ASP.NET*, яка реалізує шаблон *MVC*. В даному шаблоні виділено три основні частини: модель – представляє дані і реагує на команди контролера, змінюючи свій стан; контролер – відповідає за логіку вебсистеми; представлення – відповідає за відображення даних для користувача. Основна ціль цього шаблону це відділити бізнес-логіку від її представлення. Задачі, які виконує даний шаблон: до однієї моделі можна приєднати декілька представлень; не змінюючи реалізацію

представлень можна змінити реакцію за дії користувача, для цього слід використати інший контролер; є можливість розділення задач для вебпрограмістів, тобто той, хто розробляє бізнес-логіку, можуть взагалі не мати уявлення, яке представлення буде використане.

ASP.NET, дозволяє створювати вебсистеми за допомогою мови програмування *C#*, а саме серверну частину. *C#* - це мова кодування, яка використовується всередині фреймворка *.NET*. *C#* - це об'єктно-орієнтована мова програмування, тобто вона може збільшити продуктивність у процесі розробки. *C#* має основні плюси - безпека типів, спрощене об’явлення типів, підтримка версій та масштабованість а також інші функції, які роблять розробку рішень швидшою та простішою.

Для візуальної частини було обрано *HTML, CSS*. *HTML*, мова розмітки *HyperText*, надає структуру та значення вмісту вебсторінки, визначаючи цей вміст як, наприклад, заголовки, абзаци або зображення. *CSS*, або каскадні таблиці стилів – це мова презентацій, створена для стилізації зовнішнього вигляду вмісту, використовуючи, наприклад, шрифти або кольори.

Одні з важливих аспектів веброзробки це є кросплатформенність та використання технології *AJAX*. Кросплатформенність – це зручність як і для веброзробника, так і для користувача системою. Оскільки розроблена система є вебсистемою, це забезпечує можливість користуватися нею незалежно від операційної системи та пристрою, який буде користувач використовувати. Щодо технології *AJAX* вона допомагає звертатися до серверу без перезавантаження сторінки.

Щодо напрямку створення вебсистеми було розглянуто сферу освіти. У навчальних закладах у контексті розкладу, управління ним та прокторингу іспитів студентів, дуже зручно університету мати вебсистему для публікації екзаменів та їх деталей для того, щоб студент міг переглянути свої розклади у зручній формі.

На даний момент було виділено такі незручності як відсутність системи, у якій можна отримати структуровану інформацію про розклад, до якої матимуть дозвіл як студенти, так і адміністратор, але з різними функціональними

можливостями. Також враховуючи актуальність теми дистанційного навчання, було зроблено висновок, що зручно було б реалізувати функцію онлайн проведення екзаменів.

Якісне дослідження було проведено в цій галузі, що передбачало огляд на якому рівні знаходиться управління розкладом іспитів та їх проведення в різних навчальних закладах та виявлено, що дуже часто інформація приходить на електронну пошту або знаходиться на інформаційній дошці в університеті, що не зовсім зручно для студентів.

З впровадженням різних новітніх технологій у ІТ-галузі завжди залишається місце для розробки програмного забезпечення та інтеграції нових функцій та функціональних можливостей. Тому однією з функцій, яку можна буде додати до цієї вебсистеми в майбутньому може бути додати, це буде геолокація для направлення студентів до місць проведення іспитів.

Веброзробники повинні бути обізнані з етичними проблемами та законами, що стосуються підприємств та приватних осіб, що працюють в Інтернеті. Для належного функціонування вебсистеми конфіденційна інформація студентів зберігається в локальній базі даних, включаючи облікові дані для входу. Пароль, що зберігається у базі даних, знаходиться у зашифрованому вигляді, який зашифровано за допомогою відповідного алгоритму хешування, але інші деталі зберігаються у звичайному тексті.

Зазначимо, щодо юридичних та етичних питань: інформація про студентів, що зберігається у базі даних, не буде використана з іншою метою, крім тієї, для якої вона була зібрана спочатку. Оцінка та оцінки модуля студентів жодним чином не зберігатимуться в системі, оскільки це не є основною метою проекту. Студенти повинні вводити позначки своїх модулів кожного разу, коли вони хочуть використовувати функцію оцінки оцінок проекту.

У вебсистемі реалізована функція написання онлайн-іспитів. В наш час це є дуже актуальним. У разі проведення онлайн екзамену, у користувача буде запрошено дозвіл до відео камери та захвату екрану, після скаладання іспиту ці файли будуть збережені на сервері. Проктор отримає доступ до них, та зможе

проаналізувати процес написання екзамену, чи не біло виявлено шахрайства. Даний підхід є зручним, оскільки студенти можуть самі обрати час для здавання онлайн-іспиту, оскільки перевірка на чесність написання їх роботи, буде виконана пізніше. Для реалізації такої функції було розглянуто технологію *WebRTC*. Ця технологія дозволяє отримувати медіадані через браузер. Для отримання доступу до них необхідно використати метод *getUserMedia()* та передати три параметри: обєкт зі списком того, що необхідно (аудіо, відео), *success callback, error callback.*

Під час розробки вебсистеми спочатку визначився ряд функціональних та нефункціональних вимог до системи. Це допомагає структуровано описати функціонал системи. Окрім цього для зручності розробки було створено *UML*-діаграми. Фінальною точкою веброзробки було написання тестів, які перевіряють коректність роботи вебсистеми.

Основні результати створення дипломної роботи:

1. Виконано опис основних етапів створення вебсистем, досліджено як можна у вебсистемі реалізувати функцію для онлайн-проведення екзаменів.

2. Проаналізовано організацію навчального процесу в закладах освіти: інформування студентів про екзамени та написання іспитів онлайн.

3. Розроблено вебсистему для організації розкладу екзаменів.

4. Розробленоі функціонал для проведення онлайн-екзаменів, який забезпечує можливість здавати іспити під час дистанційного навчання.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.
2. ДСТУ ГОСТ 3008-95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.»
3. Роббинс Д. *HTML5, CSS3* и *JavaScript*. Исчерпывающее руководство. / Д. Роббинс. – М.: Эксмо, 2014. – 211 с.
4. Джамса Крис. Эффективный самоучитель по креативному *Web*-дизайну. *HTML, XHTML, CSS,JavaScript, PHP, ASP, ActiveX*. Текст, графика, звук и анимация. Пер с англ./Крис Джамса, Конрад Кинг, Энди Андерсон - М.: ООО "ДиаСофтЮП", 2005.- 672 с.
5. Дженнифер Н. Роббинс. *HTML5, CSS3* и *JavaScript*. Исчерпывающее руководство. – М.: Эксмо, 2014. – 528 с.
6. Стивен Шафер. *HTML*, *XHTML* и *CSS*. Библия пользователя, 5-е издание. – М.: «Диалектика», 2010. – 656 с.
7. *WebRTC*. [Електронний ресурс] / *WebRTC*. *–URL: https://webrtc.org* Дата звернення: 06.11.2020.
8. *How WebRTC Is Revolutionizing Telephony* [Електронний ресурс] / Trilogy-*LTE. –URL: http://blogs.trilogy-lte.com/post/77427158750/how-webrtc-is-revolutionizing-telephony* Дата звернення: 06.11.2020.
9. Тег *<iframe>* [Електронний ресурс] *htmlbook.ru. URL*: *http://htmlbook.ru/html/iframe* Дата звернення: 07.11.2020.
10. *Chan, S., & Zhang, Y. (1997). EMS: An examination scheduling and management system. Expert Systems With Applications, 12(3), 311-321. doi: 10.1016/s0957-4174(96)00102-9.*
11. *Dimopoulou, M., & Miliotis, P. (2001). Implementation of a university course and examination timetabling system. European Journal Of Operational Research, 130(1), 202-213. doi: 10.1016/s0377-2217(00)00052-7.*

12. *Janus* [Електронний ресурс] *Meetecho*. *URL*: *https://janus.conf.meetecho.com/* Дата звернення: 06.11.2020.

13. *face-api.js — JavaScript API for Face Recognition in the Browser with tensorflow.js* [Електронний ресурс] *itnext. URL: https://itnext.io/face-api-js-javascript-api-for-face-recognition-in-the-browser-with-tensorflow-js-bcc2a6c4cf07* Дата звернення*: 10.11.2020.*

14. *Navigator.getUserMedia()* [Електронний ресурс] *MDNWebDocs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/getUserMedia* Дата звернення: 16.11.2020.

15. *What is usability testing?* [Електронний ресурс] *URL: https://www.experienceux.co.uk/faqs/what-is-usability-testing/* Дата звернення: 16.11.2020.

16. *Features*. [Електронний ресурс] *Retrieved from Visual Paradigm: https://www.visual-paradigm.com/features* Дата звернення: 17.11.2020.

17. *Spacey, J. (2017). 11 Examples of Usability Requirements.* [Електронний ресурс]. *https://simplicable.com/new/usability-requirements* Дата звернення: 17.11.2020.

18. *Maurya, A. (2017). What is a Minimum Viable Product (MVP).* [Електронний ресурс] *https://blog.leanstack.com/minimum-viable-product-mvp-7e280b0b9418* Дата звернення: 17.11.2020.

19. *Makabee, H. (2014). The Minimum Viable Product and Incremental Software Development.* [Електронний ресурс] *https://effectivesoftwaredesign.com/2014/11/02/the-minimum-viable-product-and-incremental-software-development/*

20. *What is Software Testing? - Definition from Techopedia.* [Електронний ресурс] *https://www.techopedia.com/definition/17681/software-testing*

21. *Lashkari, A., Parhizkar, B., & Tayyud, J. (2010). A New Exam Management System Based onSemi-Automated Answer Checking System. (IJCSIS) International Journal Of Computer Science And Information Security, 8(1).*

22. *Academic Writing Success | Academic Writing Coach Reviews. (2018).* [Електронний ресурс] *https://academiccoachingandwriting.org/dissertation-doctor/dissertation-doctor-blog/iv-the-structure-of-your-literature-review*

23. *Stack Exchange. (2018). When to use PHP or ASP.NET?*. [Електронний ресурс] *https://softwareengineering.stackexchange.com/questions/65414/when-to-use-php-or-asp-net*

24. *McLeod, S. (2017). Qualitative vs Quantitative Research | Simply Psychology.* [Електронний ресурс] *https://www.simplypsychology.org/qualitative-quantitative.html*

25. *Pal, S. (2016). Software Engineering | Spiral Model - GeeksforGeeks.* [Електронний ресурс] *https://www.geeksforgeeks.org/software-engineering-spiral-model/*

26. *Ghahrai, A. (2017). Incremental Model - Advantages and Disadvantages.* [Електронний ресурс] *https://www.testingexcellence.com/incremental-model/*

27. *Solutions, M. (2017). A Comparison between MySQL vs. MS SQL Server* [Електронний ресурс] *https://medium.com/@mindfiresolutions.usa/a-comparison-between-mysql-vs-ms-sql-server-58b537e474be*

28. *Frederik, J. (2014). PHP vs ASP.NET? What you should really be comparing instead...*

29. [Електронний ресурс] *https://www.linkedin.com/pulse/2014-1114182637-12880086-php-vs-asp-net-what-you-should-really-be-comparing-instead/*

30. *DeFranzo, S. (2011). Difference between qualitative and quantitative research.* [Електронний ресурс] *https://www.snapsurveys.com/blog/qualitative-vs-quantitative-research/*

Додаток А

Лістинг файлу *login.aspx.cs*

```
using System;
using System.Web.Security;

public partial class login : System.Web.UI.Page
{
   //creating a new instance of class Login
   Login userLogin = new Login();

   protected void Page_Load(object sender, EventArgs e)
   {
      if(Session["adminUsername"] != null || Session["username"] != null)
      {
         Response.Redirect("Default.aspx");
      }
   }


   protected void loginBtn_Click(object sender, EventArgs e)
   {
      bool test;  //a boolean variable to store either true or a false.

      String username = usernameTxt.Text;
      String password = passwordTxt.Text;

      //hashing the password and storing it into a new local variable.
      String passwordEncrypt =
FormsAuthentication.HashPasswordForStoringInConfigFile(password, "SHA1");

      if (charactersTxt.Text.Equals(charsToValidateLbl.Text))
      {
         //calling a authenticateLogin function of class Login and then providing
the inputs
         //entered by a user to the parameters.
         test = userLogin.authenticateUsersViaDatabase(username,
passwordEncrypt);

         if (test && userLogin.getUserRole(username).Equals("student"))
         {
```

```
                Session["username"] = username;     //creating a session variable for
username.
                string userID = userLogin.getStudentID(username);
                Session["userID"] = userID;      //creating a session variable for
student ID.

//System.Web.Security.FormsAuthentication.RedirectFromLoginPage(usernameTxt.Text
, false);
                Response.Redirect("Default.aspx");
            }
            else if(test && userLogin.getUserRole(username).Equals("admin"))
            {
                Session["adminUsername"] = username;     //creating a session
variable for username.
                string adminUserID = userLogin.getStudentID(username);
                Session["adminUserID"] = adminUserID;      //creating a session
variable for admin user ID.

//System.Web.Security.FormsAuthentication.RedirectFromLoginPage(usernameTxt.Text
, false);
                Response.Redirect("Default.aspx");
            }
            else
            {
                incorrectLoginLbl.Text = "***ERROR: Incorrect Username or
Password !";
            }
        }
        else
        {
            //***ERROR: Characters do not match!;
        }
    }
}
```

Лістинг файлу *addExams.aspx.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
```

```
using System.Web.UI.WebControls;

public partial class addExams : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (Session["adminUsername"] == null && Session["username"] == null)
        {
            Response.Redirect("login.aspx");
        }
        else if (Session["username"] != null)
        {
            Response.Redirect("Default.aspx");
        }
    }

    protected void submitBtn_Click(object sender, EventArgs e)
    {
        if (IsValid)
        {
            //if the page validation is successful, then do the following...

            //get the values from textboxes entered by a user and store them in the local
variables.
            string examCode = examCodeTxt.Text;
            string examName = examNameTxt.Text;
            string course = chooseCourseDropDownList.Text;
            string module = moduleDropDownList.Text;

            //formatting the date..
            DateTime date = Convert.ToDateTime(dateTxt.Text.Trim());
            String dateString = date.DayOfWeek.ToString() + ", " +
date.ToLongDateString();

            //formating the exam start time
            DateTime startTime = Convert.ToDateTime(startTimeTxt.Text.Trim());
            string startTimeString = string.Format("{0:hh:mm tt}", startTime);

            //formatting the exam end time
            DateTime endTime = Convert.ToDateTime(endTimeTxt.Text.Trim());
            string endTimeString = string.Format("{0:hh:mm tt}", endTime);

            string duration = durationTxt.Text;
            string building = buildingTxt.Text;
```

```
        string room = roomTxt.Text;

        //statusLbl.Text = examCode + ",\t" + examName + ",\t" + course + ",\t" +
module + ",\t" + dateString + ",\t\t\t" + startTimeString + ",\t\t\t" + endTimeString +
",\t" + duration + ",\t" + building + ",\t" + room;

        //inserting an examination into a database
        string mySql;
        mySql = "insert into Exams
(examCode,examName,course,module,date,duration,startTime,endTime,building,room)
" +
            "values ('" + examCode + "','" + examName + "','" + course + "','" + module
+ "','" + dateString + "','" + duration + "','" + startTimeString + "','"
            + endTimeString + "','" + building + "','" + room + "')";
        examDataSource.SelectCommand = mySql;
        examDataList.DataBind();

        statusLbl.Text = "Examination successfully added to a database!";
        statusLbl.ForeColor = System.Drawing.Color.Green;
        Response.AddHeader("REFRESH", "5;URL=addExams.aspx");
      }
   }
       }
```

Лістинг файлу *myExams.aspx.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class myExams : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      if (Session["adminUsername"] == null && Session["username"] == null)
      {
         Response.Redirect("login.aspx");
      }
      else if (Session["adminUsername"] != null)
```

```
        {
            Response.Redirect("Default.aspx");
        }
    }

    protected void myExamsListView_SelectedIndexChanged(object sender, EventArgs e)
    {

    }
}
```