Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Пояснювальна записка

до магістерської кваліфікаційної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: Розробка методів шифрування даних для WEB-системи дистанційного навчання

Виконав: студент II курсу групи 1ПІ-18м
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Погодич. Р. В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Майданюк. В.П.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. КН Крилик Л.В.
(прізвище та ініціали)

ВНТУ – 2019

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Освітньо-кваліфікаційний рівень – магістр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
"____"_____2019 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Погодичу Роману Васильовичу

1. Тема роботи: Розробка методів шифрування даних для WEB-системи дистанційного навчання
керівник роботи: к.т.н., доцент кафедри ПЗ Майданюк В. П. затверджені наказом вищого навчального закладу від "___"_____20__року №__

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи : мова програмування – JavaScript; фреймворк – Sails.js; тип навчання – дистанційне; тип обробки даних – текстовий, аудіо, відео; Тип додатку – WEB-система; максимальний час обробки expairedAt – одна доба.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ; обґрунтування вибору методу розробки та постановка задачі дослідження; розробка системи захисту web-системи для проведення дистанційного навчання; розробка програмних модулів системи дистанційного навчання на основі web-конференцій; тестування роботи системи дистанційного навчання на основі web-конференцій; висновки; додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): тема роботи, мета і задачі роботи; шифрування даних в системі; переваги розроблювальної системи; структура системи; ER діаграма системи; сторінки системи; основні результати роботи.

6. Консультанти розділів магістерської кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Майданюк В. П., к.т.н., доцент кафедри ПЗ		
5	Бальзан М.В., к.е.н., доцент кафедри ЕПіВМ		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Обґрунтування вибору методу розробки та постановка задачі дослідження		Вик.
2	Розробка системи захисту web-системи для проведення дистанційного навчання		Вик.
3	Розробка програмних засобів системи дистанційного навчання на основі web-конференцій		Вик.
4	Розробка інтерфейсу користувача системи		Вик.
5	Розробка модулів системи		Вик.
6	Тестування роботи системи		Вик.
7	Тестування роботи системи дистанційного навчання на основі web-конференцій		Вик.

Студент _____ **Погодич Р.В.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Майданюк В. П.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У магістерській кваліфікаційній роботі проведено детальний аналіз методів шифрування даних у WEB-системах. Сформульовано мету досліджень - підвищення рівня захисту даних від несанкціонованого доступу у WEB-системах дистанційного навчання за рахунок застосування криптографічних методів та автоматизації процесу навчання.

Запропоновано метод реалізації WEB-додатків з використанням архітектурного шаблону MVC(Model-View-Contrller), відмінністю якого є розбиття системи на 3 сутності – моделі даних, інтерфейс користувача та модулі керування, що надає можливість повторного використання окремих компонентів програми. Розроблено структуру WEB-системи для дистанційного навчання на основі шаблону MVC, яка забезпечує гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення системи. Подальшого розвитку отримав метод захисту даних в системах дистанційного навчання, особливістю якого є шифрування бази даних з використанням асиметричного алгоритму, що дозволило підвищити надійність захисту даних.

Розробка виконана з використанням платформи Node.js та високопродуктивного фреймворку Sails.js із застосуванням сучасних практик та підходів до структуризації коду, що дозволило значно підвищити швидкість розробки системи та її захищеність від несанкціонованого доступу.

Отримані в магістерській кваліфікаційній роботі наукові та практичні результати можна використати для організації систем для проведення дистанційного навчання, які можуть бути використані у навчальних закладах.

ANNOTATION

A detailed analysis of data encryption methods in WEB systems is carried out. The purpose of the research was formulated to increase the level of protection of data against unauthorized access in WEB-systems of distance learning through the use of cryptographic methods and automation of the learning process.

A method for implementing WEB applications using the Model-View-Contrller (MVC) architectural template is proposed, the difference being the system's breakdown into three entities - data models, user interface, and control modules, which makes it possible to reuse individual components of the program. We have developed a web-based web-based e-learning system based on the MVC template that provides flexible software design that should facilitate further changes or extensions of the system. The method of data protection in distance learning systems, whose feature is the encryption of the database with the use of asymmetric algorithm, has further developed, which allowed to increase the security of data protection.

The development was performed using the Node.js platform and the high-performance Sails.js framework using modern code structuring practices and approaches, which significantly increased the speed of system development and its protection against unauthorized access.

The scientific and practical results obtained can be used to organize systems for conducting distance learning that can be used in educational institutions.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Нововведення, або інновації, характерні для будь-якої професійної діяльності людини, а тому стають предметом вивчення, аналізу та впровадження. Інновації в системі освіти самі по собі не виникають, вони є результатом наукових пошуків, передового педагогічного досвіду окремих викладачів і цілих колективів.

Одним з видів інновацій в організації професійної освіти є введення дистанційного навчання. На відміну від заочного навчання дистанційне навчання дає можливість вчитися, перебуваючи на будь-якій відстані від навчального закладу. І якщо при заочному навчанні студенту доводиться неодноразово приїжджати в навчальний заклад, то дистанційне навчання дозволяє практично повністю цього уникнути. Ідея дистанційного навчання полягає в тому, що взаємодія викладача й студента відбувається у віртуальному просторі: обоє вони перебувають за своїми комп'ютерами й спілкуються за допомогою Інтернету.

Дистанційне навчання - сукупність технологій, що забезпечують доставку студентам основного обсягу навчального матеріалу, інтерактивна взаємодія студентів і викладачів у процесі навчання, надання студентам можливості самостійної роботи з навчальними матеріалами, а також у процесі навчання.

Широке впровадження інформаційних технологій в систему освіти робить закономірною та актуальною проблему захисту інформації. Дослідження показують, що лише половина фахівців з інформаційної безпеки вважають свою програму чи WEB-систему такою, що готова протистояти сучасним інформаційним загрозам, зокрема і таким, що можуть призвести до неконтрольованого поширення інформації за межі інформаційних систем, у яких вона обробляється.

Актуальність теми полягає в тому, що однією з важливих галузей досліджень в системах, мережах і пристроях IT є дослідження та розробка нових методів захисту інформації та забезпечення інформаційної безпеки систем, мереж

і пристроїв. Захист інформації значною мірою базується на використанні криптографічних методів, пов'язаних з шифруванням даних.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення рівня захисту даних від несанкціонованого доступу у WEB-системах дистанційного навчання за рахунок застосування криптографічних методів та автоматизації процесу навчання.

Відповідно до поставленої мети в роботі вирішуються такі завдання:

- проведення аналізу існуючих методів захисту даних в системах дистанційного навчання для визначення їх продуктивності та надійності;
- розробка структури та алгоритмів програмного продукту;
- розробка інтерфейсу системи для дистанційного навчання;
- розробка програмних модулів системи проведення дистанційного навчання.

Об’єкт дослідження – процес захисту даних в системі дистанційного навчання.

Предмет дослідження – методи та засоби шифрування даних в WEB-систем для дистанційного навчання.

Методи дослідження. У процесі досліджень використовувались: прикладна теорія інформації, методи теорії алгоритмів для розробки алгоритмів і програмних модулів, методи теорії баз даних для розробки структури бази даних; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод реалізації веб-додатків з використанням архітектурного шаблону MVC(Model-View-Contrller), відмінністю якого є розбиття системи на 3 сутності – моделі даних,

інтерфейс користувача та модулі керування, що надає можливість повторного використання окремих компонентів програми.

2. Подальшого розвитку отримав метод захисту даних в системах дистанційного навчання, особливістю якого є шифрування бази даних з використанням асиметричного алгоритму, що дозволило підвищити надійність захисту даних.

Практична цінність отриманих результатів. Практична цінність полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби захищеної WEB-системи дистанційного навчання.

Впровадження. Впровадження результатів досліджень підтверджуються відповідним актом та використовуються у Вінницькому національному медичному університеті ім. М.І. Пирогова.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто.

Структура та обсяг роботи. Магістерська кваліфікаційна роботи складається зі вступу, чотирьох розділів, висновків, списку літератури, що містить 9 найменувань, 4 додатків. Робота містить 12 ілюстрацій, 5 таблиць. У першому розділі виконано обґрунтування вибору методу розробки та здійснено поставку задачі дослідження. У другому розділі було розроблено систему захисту WEB-системи для проведення дистанційного навчання та розглянуто підходи до захисту систем в Інтернеті. У третьому розділі було здійснено розробку програмних засобів WEB-системи для проведення дистанційного навчання. У четвертому розділі було проведено тестування роботи системи. У п’ятому розділі проведено економічне обґрунтування доцільності та економічної вигоди даного проекту. У додатках міститься технічне завдання на роботу, акт впровадження, лістинг коду та ілюстративний матеріал до захисту роботи.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Методи захисту інформації

Одним з найпопулярніших методів захисту WEB-систем є криптографічний захист інформації, що передбачає використання математичних методів для перетворення інформації за допомогою шифрування. Криптографічний захист переважно здійснюється в процесі передавання інформації каналами зв’язку або під час її опрацювання на робочих станціях і серверах[1].

До передавання інформації каналами зв’язку ставлять такі вимоги:

- забезпечення конфіденційності інформації – гарантія того, що конкретна інформація доступна тільки тому кругу користувачів, яким вона призначена;
- забезпечення цілісності інформації – гарантія того, що інформація існує в її початковому вигляді, тобто при її передача не було здійснено її модифікацію;
- автентичність сторін інформаційного обміну – гарантія того, що джерелом інформації являється той, хто є її автором.

Порушення хоча б однієї з трьох складових призводить до порушення інформаційної безпеки WEB- системи в цілому. Конфіденційність інформації можна досягнути за допомогою симетричного або асиметричного підходу шифрування. Цілісність інформації та автентичність сторін досягається використання хеш-функцій та технології цифрового підпису. У процесі клієнт-серверної взаємодії захист інформації, забезпечується за допомогою протоколів SSL, TLS, SSH.

Проблема захисту інформації через її перетворення, робить неможливим її прочитання сторонніми особами.

Методи криптографічного захисту передбачають як програмне так і

апаратне використання. При апаратної реалізації всі процедури шифрування і дешифрування виконуються спеціальними електронними схемами. Найбільшого поширення набули модулі, що реалізують комбіновані методи. Більшість зарубіжних серійних засобів шифрування засноване на американському стандарті DES. Основною перевагою програмних методів реалізації захисту є їх гнучкість, тобто можливість швидкої зміни алгоритмів шифрування.

1.2 Шифрування даних в WEB-системах

Шифрування – це кодування даних з метою захисту від несанкціонованого доступу. Процес кодування називається шифруванням, а процес декодування – розшифруванням. Саме кодоване повідомлення називається шифрованим, а застосований метод називається шифром. Основна вимога до шифру полягає в тому, щоби розшифрування (і, можливо, шифрування) були можливі тільки при наявності санкцій, тобто деякої додаткової інформації (або пристрою), яка називається ключем шифру. Процес декодування шифровки без ключа називається дешифруванням. Галузь знань про шифри, методи їх побудови та розкриття називається криптографією. Властивість шифру протистояти розкриттю називається криптостійкістю або надійністю і звичайно визначається складністю алгоритму дешифровки. У практичній криптографії криптостійкість шифру оцінюється з економічних міркувань. Якщо розкриття шифру коштує (в грошовому еквіваленті, включаючи необхідні комп’ютерні ресурси, спеціальні пристрої тощо) більше, за саму зашифровану інформацію, то шифр вважається достатньо надійним. Наука яка займається шифруванням називається криптографією.

Криптогра́фія (від грецького κρυπτός — прихований і gráphein — писати) — наука про математичні методи забезпечення конфіденційності, цілісності і автентичності інформації. Тривалий час під криптографією розумілось лише шифрування — процес перетворення звичайної інформації (відкритого тексту)

в незрозуміле «сміття» (тобто, шифротекст). Дешифрування — зворотний процес відтворення інформації із шифротексту. Шифром називається пара алгоритмів шифрування/розшифрування. Дія шифру керується як алгоритмами, та, в кожному випадку, ключем. Ключ — секретний параметр (в ідеалі, відомий лише двом сторонам) для окремого контексту під час передачі повідомлення. Ключі мають велику важливість, оскільки без змінних ключів алгоритми шифрування легко зламуються і непридатні для використання в більшості випадків. Історично склалось так, що шифри часто використовуються для шифрування та дешифрування, без виконання додаткових процедур, таких як аутентифікація або перевірка цілісності[2].

До нашого часу криптографія займалася виключно забезпеченням конфіденційності повідомлень (тобто шифруванням) — перетворенням повідомлень із зрозумілої форми в незрозумілу і зворотне відновлення на стороні одержувача, роблячи його неможливим для прочитання для того, хто перехопив або підслухав без секретного знання (а саме ключа, необхідного для дешифрування повідомлення). В останні десятиліття сфера застосування криптографії розширилася і включає не лише таємну передачу повідомлень, але і методи перевірки цілісності повідомлень, ідентифікування відправника/одержувача (аутентифікація), цифрові підписи, інтерактивні підтвердження, та технології безпечного спілкування тощо.

В криптографичній термінології вихідне повідомлення називають відкритим текстом (plaintext або cleartext). Зміна вихідного тексту так, щоб скрити від інших його склад, називають шифруванням (encryption). Зашифроване повідомлення називають шифротекстом (ciphertext). Процес, при якому із шифротексту видаляється відкритий текст називають дешифруванням (decryption). Переажно в процесі шифрування і дешифрування використовується деякий ключ (key) і алгоритм забезпечує, що дешифрування можна зробити лише знаючи ключ.

Актуальність теми полягає в тому, що однією з важливих галузей досліджень в системах, мережах і пристроях IT є дослідження та розробка

нових методів захисту інформації та забезпечення інформаційної безпеки систем, мереж і пристроїв. Захист інформації значною мірою базується на використанні криптографічних методів, пов'язаних з шифруванням даних в базі даних.

1.3 Симетричний підхід до шифрування даних

Симетричний підхід до шифрування даних передбачає використання одного і того ж ключа для шифрування даних і розшифрування їх. Симетричний підхід шифрування реалізує велика кількість алгоритмів. Симетричні алгоритми шифрування поділяються на потокові та блочні алгоритми шифрування. Потокові алгоритми шифрування – це такі, послідовно обробляють текст повідомлення. Блочні алгоритми працюють з блоками фіксованого розміру. Як правило, довжина блоку дорівнює 64 бітам, але, в алгоритмі AES використовуються блоки довжиною 128 біт.

До найвідоміших блочних шифрів належать DES, AES, ГОСТ 28147-89, Camellia, Twofish, Blowfish, IDEA, RC4 та ін.

В основному, симетричні алгоритми шифрування не вимагають великої кількості обчислень, мають досить високу швидкість та довжина ключа не довга. В більшості випадків, це означає, що якісні асиметричні алгоритми в сотні або в тисячі разів повільніші за якісні симетричні алгоритми. Недоліком симетричних алгоритмів є необхідність мати секретний ключ у всіх клієнтів, хто хоче розшифрувати інформацію. Так як ключі є предметом можливого перехоплення, їх необхідно часто змінювати та передавати по безпечних каналах передачі інформації під час розповсюдження та якщо система розростається в розмірах, потрібно зберігати і обробляти декілька пар ключів.

Сьогодні найнадійнішим симетричним алгоритмом вважається стандарт шифрування AES (Advanced Encryption Standard). Шифр AES передбачає певне число повторних циклів зміни коду, внаслідок чого вихідний звичайний текст перетворюється на шифр. Кожен цикл складається з декількох етапів обробки,

включаючи той, який залежить від ключа шифрування. Використовуючи той же ключ шифрування, застосовується набір циклів у зворотній послідовності, щоб перевести шифр назад в простий текст. Схема методу симетричного шифрування зображена на рисунку 1.1.

Рисунок 1.1 – Схема симетричного шифрування

1.4 Асиметричний підхід до шифрування даних

Проблемою симетричного шифрування є необхідність передачі ключа, для розшифрування інформації, таким чином ключ може бути перехоплений кимось іншим.

Будь хто, знаючи секретний ключ, може розшифрувати інформацію. Тоді як в асиметричному шифруванні є два пов'язаних ключа — пара ключів. Відкритий ключ (англ. public key) — публічний, до нього повинні мати доступ всі ті, хто матиме потребу зашифрувати інформацію. Тоді як закритий ключ — приватний ключ (англ. private key), повинен бути доступним лише тому хто має право розшифрувати інформацію, за своїм розміром він значно більший від секретного ключа симетричного шифрування.

Метод асиметричного шифрування (або метод відкритого ключа) передбачає використання в парі два різних ключі - відкритий і секретний. Відкритий ключ (public key) вільно розповсюджується в мережі, в той час як секретний ключ (private key) завжди тримається в секреті. В асиметричному шифруванні ключі працюють в парі - якщо дані шифруються відкритим ключем, то розшифрувати їх можна тільки відповідним секретним ключем і навпаки -

якщо дані шифруються секретним ключем, то розшифрувати їх можна тільки відповідним відкритим ключем. Використовувати відкритий ключ з однієї пари і секретний з іншого - неможливо. Кожна пара асиметричних ключів пов'язана математичними залежностями.

Саме ці алгоритми покладено в основу задач автентифікації користувачів, контролю цілісності інформації, унеможливлення відмови від авторства чи факту одержання даних тощо. Важливого практичного значення асиметричні криптоалгоритми набули у застосуванні систем електронно-цифрового підпису (ЕЦП).

Асиметричне шифрування дозволяє вирішити завдання з поширенням ключів (на відміну від симетричного шифрування). Ви можете відправляти свій відкритий ключ всім, з ким хочете взаємодіяти, і не використовувати унікальні пари ключів для кожного випадку. Асиметричний шифр може забезпечити аутентифікацію і гарантувати неможливість заперечення авторства, залежно від використання алгоритму. Асиметричні алгоритми приблизно в тисячу разів повільніше, ніж симетричні, тому що вони використовують складніші математичні розрахунки, які вимагають більше обчислювальних ресурсів. Схема методу асиметричного шифрування зображена на рисунку 1.2.

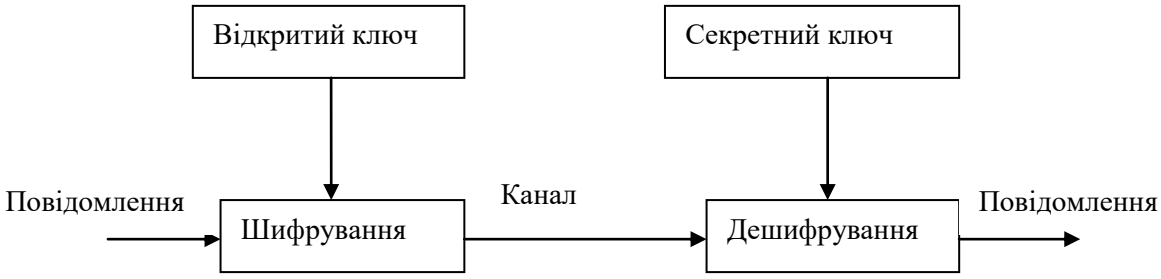


Рисунок 1.2 - Схема асиметричного шифрування

1.5 Хешування даних

Хешування (англ. Hashing) - перетворення вхідного масиву даних довільної довжини в вихідний бітовий рядок фіксованої довжини. Такі

перетворення також називаються хеш-функціями або функціями згортки, а їх результати називають хешем, хеш-кодом або дайджестом повідомлення (англ. message digest). Хешування застосовується для порівняння даних: якщо у двох масивів хеш-коди різні, масиви гарантовано розрізняються; якщо однакові - масиви, швидше за все, однакові. У загальному випадку однозначної відповідності між вихідними даними і хеш-кодом немає в силу того, що кількість значень хеш-функцій менше, ніж варіантів вхідного масиву; існує безліч масивів, що дають однакові хеш-коди - так звані колізії. Імовірність виникнення колізій відіграє важливу роль в оцінці якості хеш-функцій. Існує безліч алгоритмів хешування з різними характеристиками (розрядність, обчислювальна складність, крипостійкість і т. п.). Вибір тієї чи іншої хеш-функції визначається специфікою розв'язуваної задачі. Найпростішими прикладами хеш-функцій можуть служити контрольна сума або CRC.

Контрольна сума — розраховане значення, на основі набору даних з використанням певного алгоритму, що використовується для перевірки цілісності даних при їх передачі або збереженні. Також контрольні суми можуть використовуватись для швидкого порівняння двох наборів даних на нееквівалентність: з великою вірогідністю різні набори даних матимуть відмінні контрольні суми. Це може бути застосовано, наприклад, для детектування комп'ютерних вірусів. Перевірочна інформація при систематичному кодуванні дописується, найчастіше, на кінець повідомлення — після корисних даних. З приймального боку абонент знає алгоритм обчислення контрольної суми: відповідно, програма має можливість перевірити коректність прийнятих даних.

1.6 Аналіз методів розв’язання поставленої задачі

В наш час існує декілька методів створення системи для проведення дистанційного навчання на основі WEB-конференцій.

Першим методом є підключення вебінар платформи до власного сайту, і

ми отримаємо готову систему проведення дистанційного навчання яка в більшості випадків буде платною і функціонал її буде обмежений.

Для створення системи проведення дистанційного навчання, було прийнято рішення створити власну систему з використанням сучасних технологій та мов програмування. Дана система розробляється з урахуванням недоліків аналогічних систем є абсолютно безкоштовною і містить весь необхідний функціонал для планування і організації WEB-конференці й в вищих навчальних закладах.

Для реалізації методів захисту даних від несанкціонованого доступу, було прийнято рішення використовувати програмний підхід до шифрування даних, а саме використання алгоритмів шифрування та хешування на рівні сервера та на рівні бази даних.

1.7 Постановка задачі розробки

Для системи проведення дистанційного навчання на основі WEB-конференцій, було поставлено наступні завдання, які необхідно виконати для забезпечення надійного захисту даних в системі та розробки продукту:

- вихідний текст із зашифрованого тексту можна відтворити лише за допомогою ключа шифрування;
- послідовне перебирання ключів дешифрування з метою отримання вихідного тексту потребує значного часу обчислень або великих затрат на реалізацію цих обчислень;
- інформація про алгоритм шифрування не повинна впливати на стійкість до зламування системи шифрування;
- обрати інструментальні засоби для розробки;
- розробити сучасний інтерфейс платформи;
- розробити структуру бази даних платформи;
- розробити розподілену систему доступу до платформи;
- розробити можливість планування і автоматичного створення

конференцій;

- розробити методи захисту системи від несанкціонованого доступу до даних;
- розробити службу підтримки;
- провести тестування системи.

1.8 Висновки

Проведено аналіз проблеми захисту інформації в WEB-системах та проаналізовано можливі шляхи вирішення цієї проблеми. Розглянуто криптографічні методи захисту для збереження цілісності, конфіденційності та автентичності інформації в WEB-системах. Розглянуто криптографічні способи реалізації алгоритмів шифрування, та було прийнято рішення програмної реалізації криптографічного захисту, оскільки вона гнучкіша в реалізації. Було поставлено задачі розробки та проаналізовано їх рішення. Прийнято рішення здійснювати захист системи на рівні сервера та на рівні бази даних.

2 РОЗРОБКА СИСТЕМИ ЗАХИСТУ WEB-СИСТЕМИ ДЛЯ ПРОВЕДЕННЯ ДИСТАНЦІЙНОГО НАВЧАННЯ

2.1 Встановлення захищеного з’єднання між клієнтом і сервером

SSL (англ. Secure Sockets Layer — рівень захищених сокетів) — криптографічний протокол, який забезпечує встановлення безпечного з'єднання між клієнтом і сервером. Протокол забезпечує конфіденційність обміну даними між клієнтом і сервером, що використовують TCP/IP, причому для шифрування використовується асиметричний алгоритм з відкритим ключем. При шифруванні з відкритим ключем використовується два ключі, причому будь-який з них може використовуватися для шифрування повідомлення. Значне використання протоколу SSL призвело до формування протоколу HTTPS (Hypertext Transfer Protocol Secure), що підтримує шифрування. Дані, які передаються по протоколу HTTPS, «упаковуються» в криптографічний протокол SSL, тим самим забезпечуючи захист цих даних. Такий спосіб захисту широко використовується у світі Web для додатків, в яких важлива безпека з'єднання, наприклад у платіжних системах. HTTPS підтримується всіма браузерами. Для організації безпечної взаємодії використовується мережевий протокол передачі гіпертексту HTTPS. HTTPS – це майже той же HTTP тільки дані будуть проходити шифрування через прошарок безпеки - SSL. Тобто, відправляючи форму входу, логін та пароль шифруються за допомогою алгоритму, що використовує синхронний ключ і тільки після цього відправляються на сервер. Сервер розшифровує дані і далі працює з ними згідно своєї логіки. І навіть якщо, злодій перехопить дані, він отримає набір, непотрібних йому, біт інформації, котрі треба буде ще розшифрувати, а розшифрувати зможе тільки той, хто встановив з’єднання з сервером.

Взаємодія клієнта і сервера відбувається наступним чином:

– Клієнт встановлює з’єднання з сервером і запитує захищене з’єднання;

- При встановленні захищеного з’єднання клієнт відправляє серверу перелік алгоритмів шифрування (шифрів), котрі він знає. Сервер звіряє цей перелік із своїми шифрами та обирає найбільш надійний і повідомляє клієнта котрий алгоритм буде використовуватися;
- Сервер відправляє клієнту свій цифровий сертифікат, підписаний відповідним центром сертифікації, та відкритий ключ сервера;
- Клієнт перевіряє сертифікат. Він навіть може зробити запит до відповідного центру сертифікації, але зазвичай в ОС вже є набір встановлених кореневих сертифікатів надійних центрів сертифікації з якими браузер звіряє підпис отриманого від сервера сертифіката. Також звіряється, чи не закінчився термін дії сертифіката, чи виданий він для потрібного домена та інші речі.
- Генерується сеансовий ключ для захищеного з’єднання: клієнт генерує випадковий рядок, шифрує цей рядок відкритим ключем і відправляє на сервер, сервер розшифровує цей рядок за допомогою приватного ключа.

Тобто, як видно з опису вище, браузер і сервер використовують асиметричне шифрування лише один раз, для того, щоб створити ключ сесії. Для кожного клієнта, браузер робить цей ключ унікальним. Шифрування цим одним ключем називають симетричним. Генерація SSL сертифікату відбувалася на сайті https://letsencrypt.org/.

Проект Let's Encrypt створений для того, щоб більша частина інтернет-сайтів змогла перейти до шифрованих з'єднань (HTTPS). На відміну від комерційних центрів сертифікації, в даному проекті не вимагається оплата, переконфігурація веб-серверів, використання електронної пошти, обробка прострочених сертифікатів, що робить процес встановлення та налаштування SSL-шифрування значно більш простим. Наприклад, на типовому веб-сервер на базі Linux потрібно виконати дві команди, які налаштують HTTPS-шифрування, отримають і встановлять сертифікат приблизно за 20-30 секунд.

2.2 Модифікація методу шифрування бази даних веб-сервером та сервером БД

Процес шифрування здійснюється веб-сервером системи, який створює або отримує дані від клієнтів, тобто шифрування відбувається перед записом в базу даних. Цей підхід є досить гнучким, оскільки веб-серверу відомі ролі або права доступу користувачів, а також інформація про те, які дані є конфіденційними.

Переваги:

Одним з головних переваг шифрування веб-сервером, є те, що немає необхідності використовувати додаткове рішення для захисту даних при передачі по каналах зв'язку, так як вони відправляються вже зашифрованими. Ще одна перевага такого методу — це те, що крадіжка конфіденційної інформації стає складніше, так як зловмисник повинен мати доступ до додатка для того, щоб розшифрувати дані, що зберігаються в БД. Дане шифрування також гнучке через те, що дозволяє нам використовувати будь-який алгоритм шифрування.

Недоліки:

Для реалізації шифрування на рівні веб-сервера потрібно внесення змін не тільки в сам додаток, але і в базу даних. Також можуть виникнути проблеми з продуктивністю БД, у якій, наприклад, пропадає можливість індексування і пошуку. Ще одним недоліком є управління ключами в системі з таким шифруванням. Так як кілька програм можуть використовувати БД, то ключі зберігаються у багатьох місцях, тому неправильне управління ними може призвести до крадіжки інформації або унеможливлення її використання. До того ж, якщо виникає необхідність зміни ключа, то для початку потрібно розшифрувати всі дані зі старим ключем, і потім знову зашифрувати, використовуючи новий ключ.

Шифрування бази даних — використання алгоритми шифрування для перетворення інформації, що зберігається в базі даних (БД), в шифротекст, що

робить її прочитання неможливим для осіб, що не володіють ключами шифрування.

Прозоре шифрування бази даних (англ. Transparent Database Encryption, TDE) — технологія, яка застосовується, у продуктах Microsoft і Oracle для шифрування і дешифрування вводу-виводу файлів БД. Дані шифруються перед записом на диск і дешифруються під час читання в пам'ять, що вирішує проблему захисту «неактивних» даних, але не забезпечує збереження інформації при передачі по каналах зв'язку або під час використання. Перевагою TDE є те, що шифрування і дешифрування виконуються прозоро для додатків, тобто їх модифікація не потрібна. Застосування TDE не збільшує розмір зашифрованої БД і має незначний вплив на її продуктивність.

Шифрування на рівні стовпців бази даних (англ. Column-Level Encryption) на відміну від TDE, що шифрує БД повністю. Цей метод дозволяє шифрувати окремі стовпці з різними ключами, що забезпечує додаткову гнучкість при захисті даних. Ключі можуть бути призначені користувачам і захищені паролем для запобігання автоматичної розшифровки, однак це ускладнює адміністрування БД. При використанні шифрування на рівні стовпців необхідно внесення змін до клієнтських додатків. Крім цього зменшується продуктивність БД.

Шифрування на рівні поля (field-level encryption, FLE) – механізм, що не допускає зберігання криптографічних ключів і проведення будь-яких операцій щодо шифрування/дешифрування на сервері. Замість цього всі операції проводяться в використовуваної додатками бібліотеці клієнта MongoDB (драйвер).

Для сервера відправлений клієнтом шифротекст (зашифровані дані) є всього лише черговим типом даних для зберігання. Тобто, навіть якщо сервер буде скомпрометований або база даних раптом виявиться доступною через Інтернет, зловмисники не зможуть отримати зашифровану інформацію.

Що стосується клієнта, то механізм FLE повністю прозорий для додатка, оскільки за нього відповідає драйвер MongoDB, а не код самого додатка.

Іншими словами, у багатьох випадках для використання нової функції в код додатків не потрібно вносити ніяких змін, розробникам досить лише оновити драйвер.

Для системи проведення WEB-конференцій, було прийнято рішення виконувати шифрування даних користувачів на стороні сервера.

2.3 Авторизація і аудентифікація користувачів

Аутентифікація і авторизація необхідні для обмеження доступу користувачів, до модулів і функцій, які для них не призначені. Аутентифікація - перевірка, чи є хтось тим, за кого себе видає. Зазвичай вона має на увазі введення логіна і пароля, але також можуть бути використані й інші засоби, такі як використання смарт-карти, відбитків пальців та ін. Авторизація - перевірка, чи може аутентифікований користувач виконувати певні дії (їх часто позначають як ресурси). Найчастіше це визначається перевіркою, чи призначена користувачеві певна роль, яка має доступ до ресурсів.

Згодом, постає проблема збереження паролів зареєстрованих користувачів. Паролі зареєстрованих користувачів, в базі даних будемо зберігати у зашифрованому вигляді – це захистить від того, якщо зловмисники отримають доступ до бази даних, вони не зможуть розшифрувати паролі користувачів і здійснити вхід з-під їх імені в систему.

Для шифрування паролів користувачів, використаємо такий підхід, як хешування. Хешування — перетворення вхідного масиву даних довільної довжини у вихідний бітовий рядок фіксованої довжини. Такі перетворення також називаються функцією хешування, або хеш-функцією. Для того, щоб зловмисники, які зможуть отримати доступ БД, не дізналися всю інформацію про користувачів системи і не змогли використати їх облікові записи, запропоновано рішення хешування паролів в БД. Хешування паролів відбувається на рівні сервера, клієнт присилає свій пароль, сервер його хешує і зберігає в зашифрованому вигляді в базу даних. При наступному запиті, коли

користувач буде робити авторизацію, він присилає свій пароль, пароль шифрується і порівнюється з тим хешом, що збережений в базі даних. Якщо хеші співпадають, користувач успішно здійснює вхід в систему.

Для реалізації була використана бібліотека – bcrypt. Bcrypt — адаптивна криптографічна функція формування ключа, що використовується для безпечного зберігання паролів. Для захисту від атак за допомогою райдужних таблиць bcrypt використовує сіль (salt). Крім того, функція є адаптивною, час її роботи легко налаштовується і її можна сповільнити, щоб ускладнити атаки перебором.

2.4 Засоби захисту бази даних

У дволанковій архітектурі «клієнт - сервер» прикладна частина виконується клієнтом (на робочій станції, вузлі мережі і т. п.), а сервер здійснює доступ до БД. У випадку складності і ємності прикладної обробки у цю архітектуру додається сервер застосувань, що бере на себе основну логіку опрацювання інформації і доступу до БД. При цьому БД може бути централізованою (один сервер) або розподіленою (декілька серверів). У розподіленій БД основною умовою збереження даних є автономність і відсутність прямих і транзитивних зв'язків між таблицями, що розташовані у віддалених розділах БД. Оскільки ця умова обмежує цілісність даних, то до складу серверу включаються збережені процедури, за допомогою яких встановлюються посилання на інші розділи БД.

Як правило, захист БД здійснює спеціальний сервер. До його функцій входить забезпечення парольного захисту, шифрування, установлення прав доступу до об'єктів БД, захист полів і записів таблиць тощо.

Паролі встановлюються кінцевими користувачами або адміністраторами БД, зберігаються у захищеному вигляді у спеціальних файлах і використовуються сервером під час доступу користувачів до БД.

Шифрування здійснюється за допомогою ключів фіксованої і нефіксованої довжини для засекречування збереженої в БД інформації або при передачі інформації мережею від відправника до одержувача й навпаки.

Встановлення прав доступу до БД полягає у реєстрації користувачів для захисту від несанкціонованого доступу. До прав доступу відносяться: читання, модифікація, додавання, вилучення, зміна структур таблиць і т. ін. За допомогою дозволених для кожного користувача прав здійснюється контроль їх доступу до об'єктів БД і приймаються заходи для захисту окремих рядків, стовпчиків, полів або БД у цілому.

До основних заходів щодо проведення захисту даних БД відносяться:

- режимні, що включають парольну, криптографічну перевірки користувачів тощо;
- технологічні, що містять резервне копіювання даних, правильне їхнє збереження, експлуатацію й ін.;
- системні, з процедурами автоматизованої перевірки повноважень і істинності користувачів, які запитують доступ до даних, їх привілеїв, аудиту подій, що відбуваються, тощо.

Режимні і технологічні заходи підтримуються методичними матеріалами і стандартами, що регламентують дії групи осіб, відповідальних за безпеку БД. Системні – утворюють сервіс безпеки, що включає сервер БД із функціями захисту.

Особливістю розподілених БД (РБД) є розміщення окремих розділів даних на різних вузлах мережі, інформація про розташування яких відображається в глобальному словнику даних і використовується при доступі до РБД різних користувачів.

2.5 Модифікація методу реалізації веб-додатків з використанням архітектурного шаблону MVC

Для розробки модулів системи було використано високорівневий фреймворк Sails.js, який реалізує архітектурний шаблон MVC. У рамках архітектурного шаблону модель–вигляд–контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами. Модель (Model) відповідає за зберігання даних і забезпечення інтерфейсу до них. Вигляд (View) відповідальний за представлення цих даних користувачеві. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача (зміна положення курсора миші, натискання кнопки, ввід даних в текстове поле) і передає дані у модель.

- *Модель* є центральним компонентом шаблону MVC і відображає поведінку додатку, незалежну від інтерфейсу користувача. Модель стосується прямого керування даними, логікою та правилами додатку.
- *Вигляд* може являти собою будь-яке представлення інформації, одержуване на виході, наприклад графік чи діаграму. Одночасно можуть співіснувати кілька виглядів (представлень) однієї і тієї ж інформації, наприклад гістограма для керівництва компанії й таблиці для бухгалтерії.
- *Контролер* одержує вхідні дані й перетворює їх на команди для моделі чи вигляду.

Модель інкапсулює ядро даних і основний функціонал їхньої обробки і не залежить від процесу вводу чи виводу даних.

Вигляд може мати декілька взаємопов'язаних областей, наприклад різні таблиці і поля форм, в яких відображаються дані.

У функції контролера входить відстеження визначених подій, що виникають в результаті дій користувача. Контролер дозволяє структурувати код

шляхом групування пов'язаних дій в окремий клас. Наприклад у типовому MVC-проекті може бути користувацький контролер, що містить групу методів, пов'язаних з управлінням обліковим записом користувача, таких як реєстрація, авторизація, редагування профілю та зміна пароля.

Зареєстровані події транслюються в різні запити, що спрямовуються компонентам моделі або об'єктам, відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, якщо користувач через контролер внесе зміни до моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися.

2.6 Моделювання потоків даних в системі

Суть моделювання потоків даних при проектуванні складних інформаційно-керуючих систем – це створення так званих діаграм потоків даних. Діаграми потоків даних відображають шляхи проходження та перетворення інформації в системі.

Діаграма потоків даних (англ. Data Flow Diagram) — модель проектування, графічне представлення «потоків» даних в інформаційній системі. Діаграма потоків даних також може використовуватись для візуалізації процесів обробки даних. Діаграма потоків даних відображає джерела та споживачів інформації, вид та напрямок передачі інформації, елементи накопичення та процеси перетворення, при цьому використовуються різні засоби відображення елементів (нотації). Діаграма потоків даних зображена на рисунку 2.1.

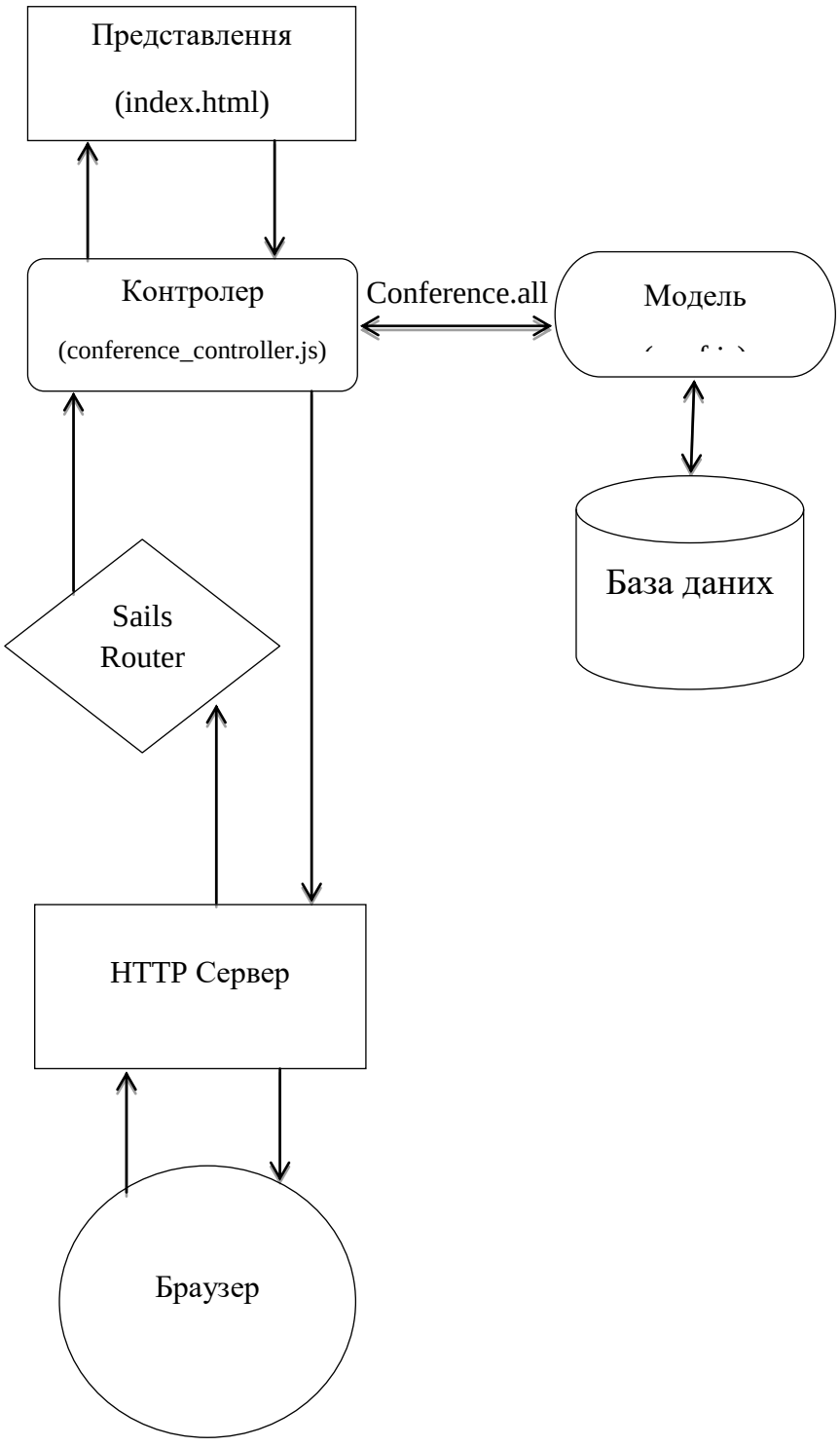


Рисунок 2.1 – Діаграма потоків даних(DFD діаграма)

На рисунку 2.1 показано типовий запит браузера перехід на indexсторінку всіх конференцій на /conferences - в термінах MVC, для відображення списку всіх створених конференцій.

1. Браузер відсилає запит наURL /conferences.

2. Sails направляє /conferences до getAll дії в контролері Conference.
3. Index дія робить запит, щоб модель Conference отримала всі конференції(Conference.all).
4. МодельConference отримує всі конференції з бази даних.
5. Модель Conference повертає всі конференції в контролер.
6. Контролер отримує список всіх конференцій і записує їх в змінну і передає її в представлення .
7. Представлення візуалізує html сторінку.

2.7 Інформаційні об’єкти системи

Розробку специфікацій програмного забезпечення починають з аналізу вимог до функціональності, вказаних в технічному завданні. В процесі аналізу виявляють зовнішніх користувачів програмного забезпечення, що розробляється, і перелік окремих аспектів його поведінки в процесі взаємодії з конкретними користувачами. Аспекти поведінки програмного забезпечення називають «варіантами використання» або «прецедентами» (use cases).

Кожен варіант використання пов'язаний з деякою метою, що має самостійне значення. Діаграми варіантів використання дозволяють наочно представити очікувану поведінку системи. Основними поняттями діаграм варіантів використання є: дійова особа, варіант використання, зв'язок.

Дійовими особами системи є адміністратор системи, викладач та студент. Варіанти використання виявляємо, аналізуючи технічне завдання, і зображуємо на діаграмі, пов'язуючи з відповідними діючими особами

Діаграма варіантів використання – UML – візуальне представлення графу використання представлена на рисунку 2.2:

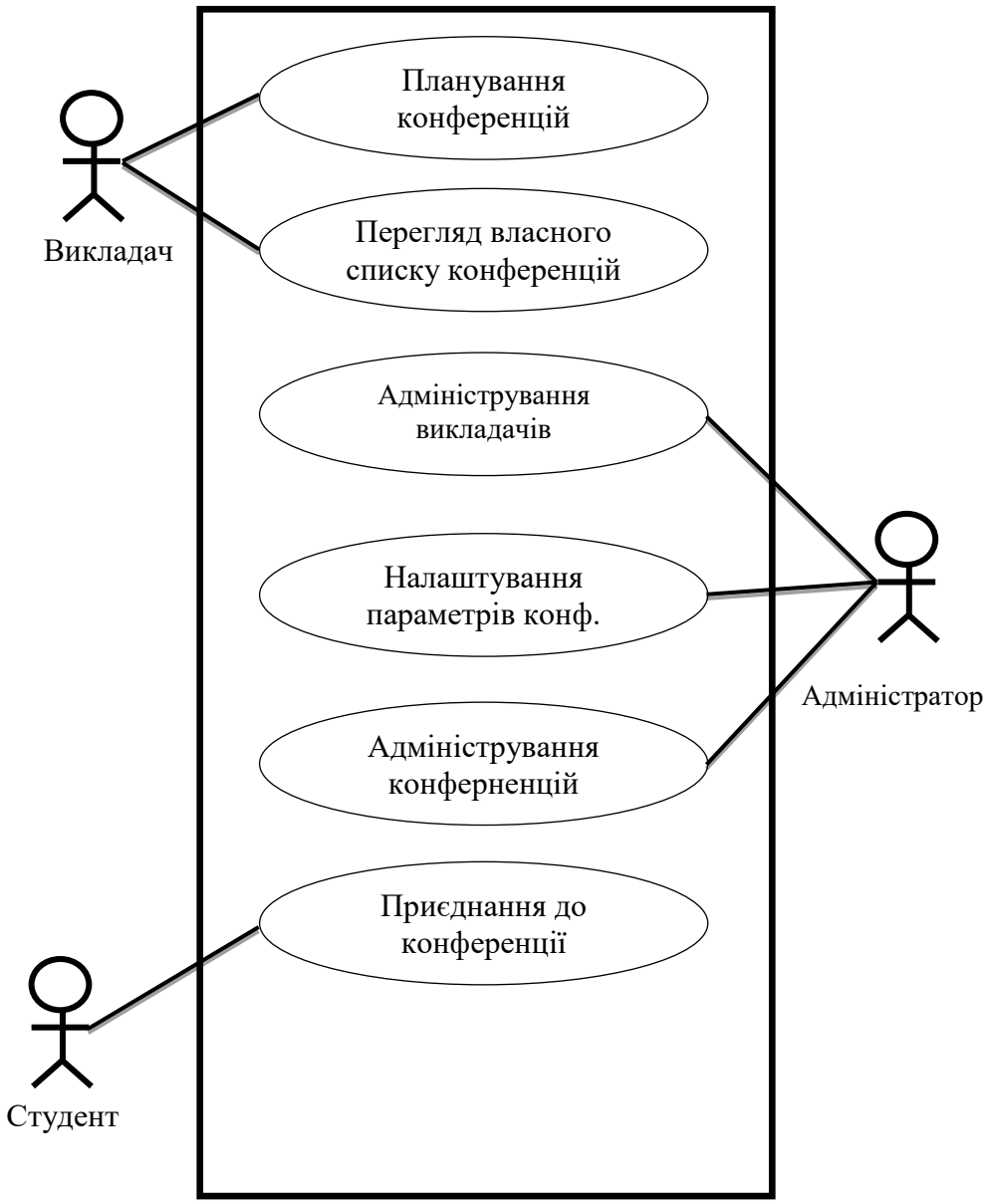


Рисунок 2.2 – Діаграма використання системи

2.8 Розробка структури бази даних для системи дистанційного навчання для проведення веб-конференцій

База даних (англ. database) – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; ця сукупність підтримує щонайменше одну з областей застосування

У реальному проектуванні структури бази даних застосовується метод – так зване, семантичне моделювання. Семантичне моделювання є моделювання

структури даних, спираючись на зміст цих даних. Як інструмент семантичного моделювання використовуються різні варіанти діаграм сутність-зв'язок (ER – Entity-Relationship).

Сутність – це клас однотипних об'єктів, інформація про яких повинна бути врахована в моделі.

Примірник сутності – це конкретний представник даної сутності.

Атрибут сутності – це іменована характеристика, що є деякою властивістю сутності.

Ключ сутності – це ненадлишковий набір атрибутів, значення яких у сукупності є унікальними для кожного екземпляра сутності.

Зв'язок – це деяка асоціація між двома сутностями. Одна сутність може бути пов'язана з іншою сутністю або сама з собою.

Існує декілька типів зв’язку:

- один-до-одного означає, що один примірник першої сутності (лівої) пов'язаний з одним примірником другий сутності (правої). Зв'язок один-до-одного найчастіше свідчить про те, що насправді ми маємо всього одну сутність, неправильно розділену на дві;
- один-до-багатьох означає, що один примірник першої сутності (лівої) пов'язаний з декількома екземплярами другий сутності (правої). Це найбільш часто використовуваний тип зв'язку. Ліва сутність (з боку "один") називається батьківського, Права (з боку "багато") – дочірньої;
- багато-до-багатьох означає, що кожен екземпляр першої сутності може бути пов'язаний з декількома екземплярами другий сутності, і кожен примірник другої суті може бути пов'язаний з декількома примірниками першої сутності.

Розглянемо структуру бази даних для заданої предметної області (рис. 2.3).

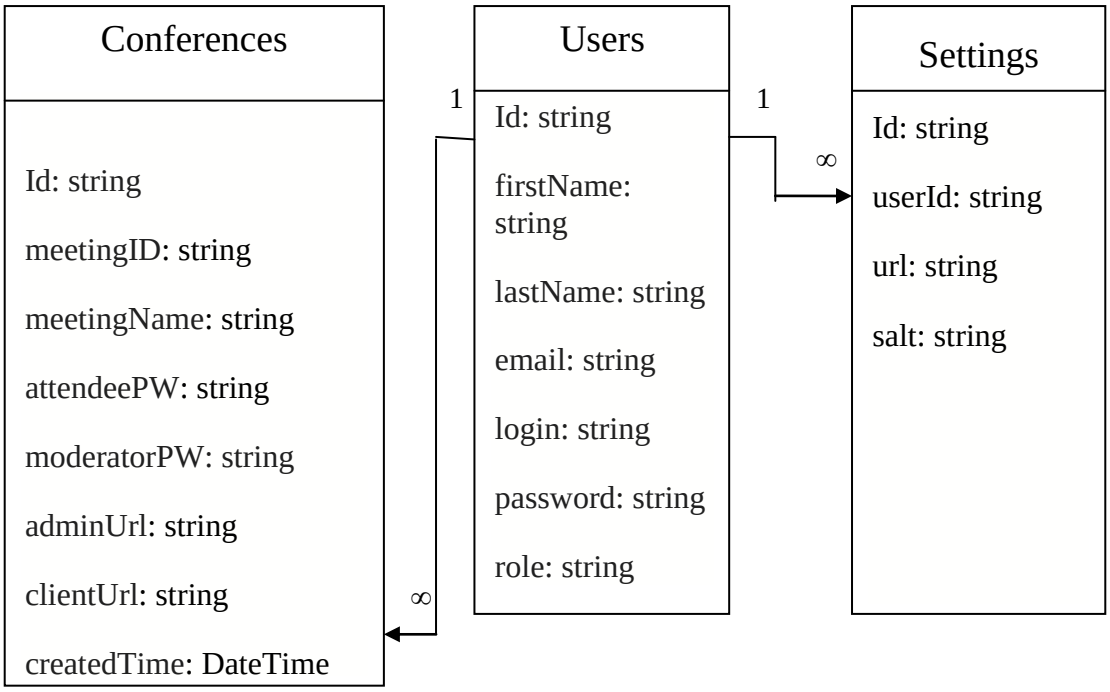


Рисунок 2.3 – Структура бази даних

Conferences – таблиця, що містить всі необхідні дані про конференції які створені системою.

Settings – таблиця, в якій зберігаються налаштування користувачів системи.

Users- таблиця, що зберігає інформацію про всіх користувачів системи.

2.9 Висновки

Визначено методи встановлення безпечного сплкування між клієнтом і сервером, засоби захисту бази даних системи і шифрування даних. Описано інформаційні об’єкти системи та наведено варіанти використання системи.

Аналіз стандартних архітектурних шаблонів показав, що для створення систем для дистанційного навчання на основі Web-конференцій найбільш прийнятним є архітектурний шаблон MVC(Model-View-Controller), оскільки він забезпечує розбиття логіки програми на 3 основні сутності та взаємодію між цими сутностями.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ ДИСТАНЦІЙНОГО НАВЧАННЯ НА ОСНОВІ WEB-КОНФЕРЕНЦІЙ

3.1 Обґрунтування вибору технологій

Для розробки модулів програмного продукту було обрано мову програмування JavaScript а саме серверну реалізацію мови - Nodejs, фреймворк Sails.js, середовище розробки JetBrains WebStorm Storm.

JavaScript (JS) — динамічна, прототипно-орієнтована мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується як частина браузера, що надає можливість коду на стороні клієнта (такому, що виконується на пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки[3].

Еволюція JavaScript з кожним роком дає можливість для веб-розробників створювати велику кількість нових технологій та інноваційних програм. Один з найбільш цікавих і популярних інструментів для створення легко масштабованих мережних додатків є Node.js - це серверна реалізація мови програмування JavaScript, заснована на движку V8.

Node.js створений у 2009 році Райаном Далем. Він створив програмну платформу, засновану на JavaScript V8. Незвично те, що платформа має вбудовані бібліотеки для обробки запитів і відповідей, не потрібно використовувати сторонній веб-сервер і будь-які інші залежності. Node.js набирає обертів і використовується такими компаніями, як Microsoft, Yahoo, LinkedIn і PayPal.

Перш за все, Node.js відрізняється від класичного JavaScript тим, що виконуваний код виконується на стороні сервера (backend), а не на стороні браузера. Для інтерпретації коду Node.js використовує движок V8, який в даний час застосовується в Google Chrome.

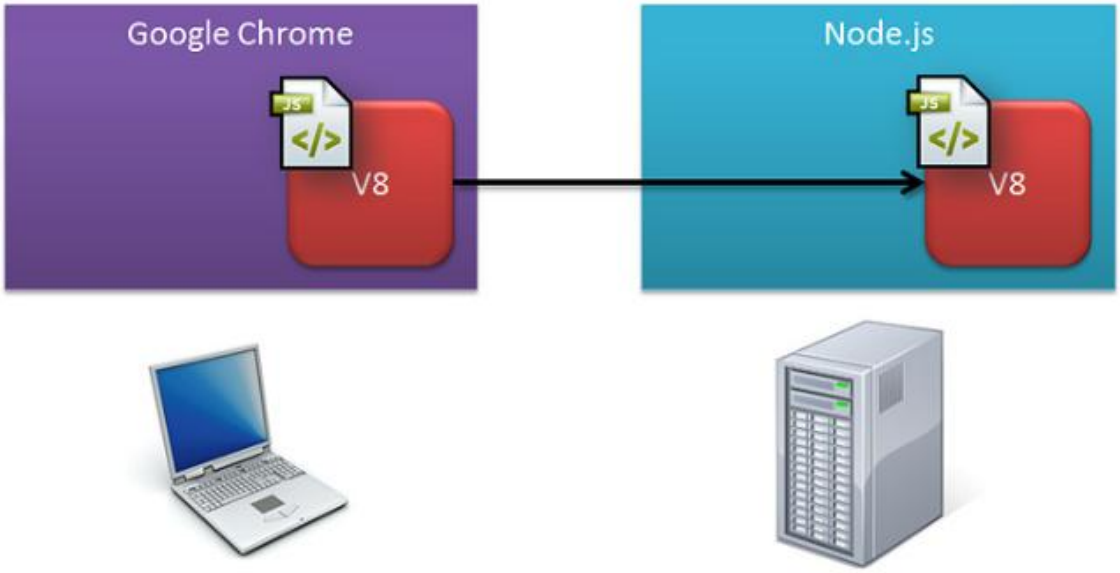


Рисунок 3.1 – Інтерпретація коду JavaScript на сервері

Крім того, всі механізми обробки запитів та інших операцій введення / виведення (I / O) побудовані на подіях. Це означає, що в Node.js немає ніякого способу, щоб заблокувати працюючий в даний момент потік. Кожна операція в Node.js виконується асинхронно. Це є величезною перевагою, особливо якщо ваш код повинен бути побудований на операціях введення-виведення: читання дисків, підключення до бази даних, веб-сервіси і т.д.

В якості серверного фреймворку на бекенді використовувався SailsJS. Sails.js - це в першу чергу MVC-фреймворк. При наявності досвіду роботи з будь-яким іншим популярним фреймворком (RoR, Yii, CodeIgniter, ASP .NET MVC) відразу після ознайомлення зі структурою проекту Sails.js відчуваєте комфорт, адже в очі кинуться знайомі по патерну MVC директорії: models, views, controllers і інші характерні для подібних продуктів речі. Основної перевагою SailsJS перед іншими MVC фреймворками NodeJS є автоматична генерація RESTful API, повна інтеграція з socket.io та своя власна ORM для виконання запитів до бази даних WaterLine. Зразу з коробки Sails пропонує адаптери до найбільш популярних баз даних таких як: MySQL, PostgreSQL,

Redis, MongoDB а також надає можливість використовувати звичайний файл для збереження даних.

В якості фронтенд фреймворку у дипломній роботі використовувся AngularJS. AngularJS - структурований JavaScript-фреймворк з відкритим вихідним кодом для динамічних web-додатків. Він ризначений для розробки односторінкових додатків, дозволяє використовувати HTML в якості мови шаблонів, а так же розширювати HTML-синтаксис, щоб код вашої програми виглядав коротко і лаконічно. Його мета - розширення браузерних додатків на основі MVC шаблону, а також спрощення тестування і розробки.

Основні переваги використання Angular:

– декларативний стиль коду;
– використання спеціальних директив;
– модульність;
– простота тестування;
– завчасно готові рішення, які спрощують розробку.

3.2 Обґрунтування вибору системи керування базами даних

Система керування базами даних (СКБД) або Система управління базами даних (СУБД) — комплекс програмного забезпечення, що надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних. Для розробки дипломного проекту було використано документно-орієнтовну систему керування базами даних MongoDB, для реалізації запитів в базу даних було використано об’єктно реляційне представлення Waterline[4].

MongoDB це крос-платформна, потужна, гнучка, документно-орієнтована база даних, що легко масштабується. Вона також має вбудовану підтримку MapReduce-style Aggregation та Geospatial індексів[5].

MongoDB складається з баз даних, які зберігають в собі колекції. Кожна колекція складається з документів, які в свою чергу складаються з полів. Поля є

парами ключ-значення. Колекції також можуть бути індексовані, що покращує швидкість вибірки та сортування.

Основні можливості MongoDB:

- документо-орієнтоване сховище (проста та потужна JSON-подібна схема даних);
- досить гнучка мова для формування запитів;
- динамічні запити;
- повна підтримка індексів;
- швидкі оновлення «на місці»;
- ефективне зберігання бінарних даних великих обсягів, наприклад, фото та відео;
- журналювання операцій, що модифікують дані в БД;
- підтримка відмовостійкості і масштабованості: асинхронна реплікація, набір реплік і шардінг;
- може працювати відповідно до парадигми MapReduce.

ORM або Object-relational mapping (Об'єктно-реляційне відображення) - це технологія програмування, яка дозволяє перетворювати несумісні типи моделей в ООП, зокрема, між базою даних і об'єктами програмування. ORM використовується для спрощення процесу збереження об'єктів в базу даних і їх вилучення, при цьому ORM сама піклується про перетворення даних між двома несумісними станами. Більшість ORM-інструментів значною мірою покладаються на метадані бази даних і об'єктів, так що об'єктам нічого не потрібно знати про структуру бази даних, а базі даних - нічого про те, як дані організовані в додатку. ORM забезпечує повне розділення завдань в добре спроектованих додатках, при якому і база даних, і додаток можуть працювати з даними кожен у своїй вихідній формі[6].

Ключовою особливістю ORM є відображення, яке використовується для прив'язки об'єкта до його даних в БД. ORM як би створює «віртуальну» схему бази даних в пам'яті і дозволяє маніпулювати даними вже на рівні об'єктів. Відображення показує як об'єкт і його властивості пов'язані з однією або

декількома таблицями і їх полями в базі даних. ORM використовує інформацію цього відображення для управління процесом перетворення даних між базою і формами об'єктів, а також для створення SQL-запитів для вставки, оновлення та видалення даних у відповідь на зміни, які додаток вносить в ці об'єкти. У SailsJS використовується вбудована ORM WaterLine, тим самим звільняючи нас від написання SQL запитів.

WaterLine надає набагато більше можливостей для роботи з окремими записами, ніж просто дії CRUD (англ. Create, Read, Update, Delete - «створення, читання, оновлення, видалення»). Він надає поєднання JS-подібного інтерфейсу і можливості робити майже все, що можна зробити за допомогою "чистого" SQL[7].

Запити WaterLine повертають відкладені відношення (Relations). Relations дають вам цю гнучкість і дозволяють більш ефективно використовувати дорогоцінний час вашої бази даних.

Для того щоб змінити схему бази даних в Sails використовуються міграції. Міграції - це зручний спосіб змінювати схему вашої бази даних весь час досить простим чином. Вони використовують Node DSL. Тому вам не потрібно писати SQL вручну, дозволяючи вашій схемі бути незалежною від бази даних. Кожну міграцію можна розглядати як нову "версію" бази даних. Схема спочатку нічого не містить, а кожна міграція змінює її, додаючи або видаляючи таблиці, стовпці або записи. WaterLine знає, як оновлювати вашу схему з часом, переносячи її з певної точки в минулому в останню версію.

Приклади роботи з базою даних на основі WaterLine:

1. Створення таблиці в БД:

Виконаємо команду для генерації моделі

```
$ bin/sails generate model Categories title:string
```

Sailsстворить файл яка буде мати вигляд:

```
module.exports = {
  attributes: {
    title:{
```

```
      type: 'string
    }
  }
};
```

Для запуску міграції використовується команда *rake db:migrate.*

2. Видалення таблиці

Виконаємо команду створення міграції на видалення

$ bin/sails generate migration DropCitiesTable

Запустим міграцію на видалення таблиці Categoriesкомандою rake db:migrate.

Інтерфейс запитів на основі Waterline:

Запит на пошук об’єкта в БД:

```
phone = Phones.find({id:10})
```

Цей самий запит на SQL:

```
SELECT * FROM phones WHERE (phones.id = 10) LIMIT 1
```

Запит на отримання всіх замовлень відсортованих по даті за спаданням:

```
Orders.find({}).sort('createdAt DESC').then(orders =>
    res.send(200, orders);
```

Цей самий запит мовою SQL:

```
SELECT orders.* FROM orders
OrderBy createdAt DESC
```

Як ми бачимо інтерфейс запитів WaterLine набагато зручніший перед «чистим» SQL.

3.3 Розробка модулів системи

При створенні проекту на SailsJS, командою sails new «appName», sails генерує стандартну структуру файлів і папок, що являється однією з його переваг, ця структура є загальною для всіх проектів, в ній можна легко зорієнтуватися дивлячись на чужий код. Структура Sails проекту представлена у Таблиці 3.1.

Таблиця 3.1 – Структура файлів проекту

Файл/директорія	Призначення
api	Серверна частина бізнес логіки
api/controllers	Контролери додатку, використовуються для взаємодії з моделлю і обслуговування клієнтських запитів
api/models	Моделі, які описують структури для зберігання даних додатку
api/polices	Так звані посередники, які виконуються перед конкретною дією контролера
api/responses	Логіка відповідей на запити
api/services	Сервіси, які виступають як сінгелтони, для вирішення однієї конкретної задачі
config/	Конфігураційні файли проекту
views/	Відображення додатку, за замовчуванням в якості шаблонізатору використовується EJS(embedded JavaScript)
tasks/	За замовчуванням Sails використовує Grunt для управління залежностями. Тут визначені задачі і налаштування Grunt.
assets/	Зберігаються всі клієнські бібліотеки
assets/js/	JSфайли які виконуються на клієнті

Оскільки Sails.js реалізує архітектурий шаблон MVC, для створення повноцінного модуля нам потрібно створити модель в системі, описати контролер цієї моделі та створити сторінку відображення. Розглянемо приклад створення модуля на основі модуля користувачів системи.

Для генерації моделі в Sails виконаємо команду sails generate model Users, команда створить в директорії models файл Users.js. Далі опишемо нашу модель, вона буде мати наступний вигляд:

```
module.exports = {
  schema: true,

  attributes: {
    firstName: {
      type: 'string',
      required: true
    },
    lastName: {
      type: 'string',
      required: true
    },
```

```
    email: {
      type: 'email',
      required: true
    },
    login: {
      type: 'string',
      required: true
    },
    password: {
      type: 'string',
      required: 'true'
    },
    role: {
      type: 'string',
      enum: ['user', 'admin'],
      defaultsTo: 'user'
    }
  }
};
```

Флаг schema: true означає те, що тільки ці поля можуть бути записані в базу даних. Для генерації контролера моделі виконаємо команду sails generate controller Users. Ця команда створить файл UsersController.js в паці controllers. Контролери призначені для взаємодії з моделю та передачі даних представленню. Приклад події контролера для відображення всіх користувачів системи, ця функція доступна тільки адміністратору системи:

```
getAll(req, res) {
    const {user} = req;
    let query;
    if (user.role === 'admin') {
      query = Users.find({});
    } else {
      console.log('FORBITTEN');
    }
    query.sort('createdTime DESC');
    query.exec((error, users) => {
```

```
        if (error) {
            res.badRequest();
        }
        res.send(200, users);
    });
}
```

Res.send(200, users) - відправляє на клієнт змінну users яка містить в собі всіх користувачів системи в форматі JSON, першим параметром функція приймає http статус код, наприклад 200 – ОК, 403 – заборонено, 404 – не знайдено, 500- внутрішня помилка серверу. Далі ці дані відправляються в браузер користувача і обробляються відповідним чином для їх представлення користувачу який зробив запит на отримання цих даних.

3.4 Висновки

Аналіз мов програмування показав, що найбільш прийнятною для розробки є застосування мови програмування JavaScript, а саме її серверної реалізації Node.js, оскільки вона забезпечує досить велику швидкість обробки даних. Обґрунтовано вибір системи керування базами даних а саме обрано MongoDB, оскільки ця база даних є документно-орієнтованою та не має жорсткої структури. На наглядному прикладі продемонстровано процес створення нового модуля авторизації користувачів системи за допомогою фреймворку Sails.js.

4 ТЕСТУВАННЯ РОБОТИ СИСТЕМИ ДИСТАНЦІЙНОГО НАВЧАННЯ НА ОСНОВІ WEB-КОНФЕРЕНЦІЙ

4.1 Інструкція користувача

Розглянемо процес планування WEB-конференції, адміністрування користувачів системи та приєднання студентів до WEB-конференції. Авторизований викладач відкриває головну сторінку та бачить перед собою список всіх своїх конференцій (рис. 4.1).

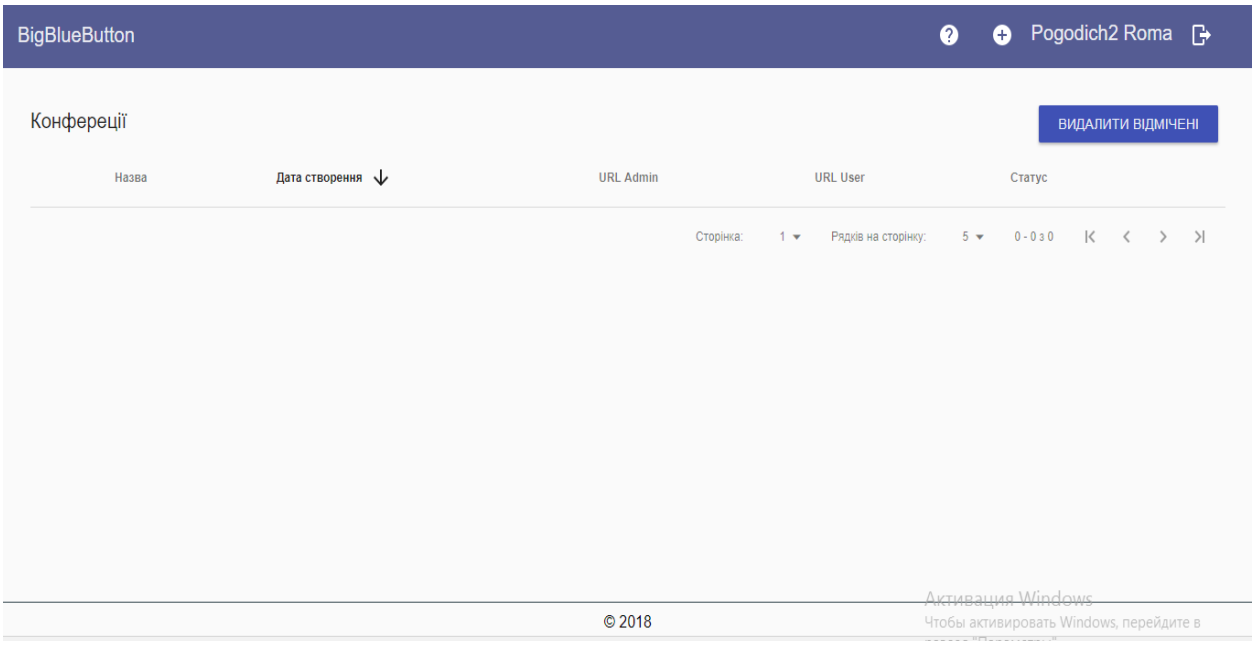


Рисунок 4.1 – Головна сторінка

Для планування WEB-конференції необхідно натиснути на кнопку «Створити» в шапці сайту і на екрані з’явиться діалогове вікно для заповнення всіх необхідних даних про конференцію (рис 4.2). Слід зазначити те, що при створенні конференції користувач заповнює тільки 3 поля, це ім’я адміністратора яке буде показуватися в WEB-конференції, час на який буде запланована ця конференція та назву самої WEB-конференції. Найчастіше назвою WEB-конференції виступає її тема для того щоб викладач при великій кількості конференцій міг легко знайти ту яка йому потрібна.

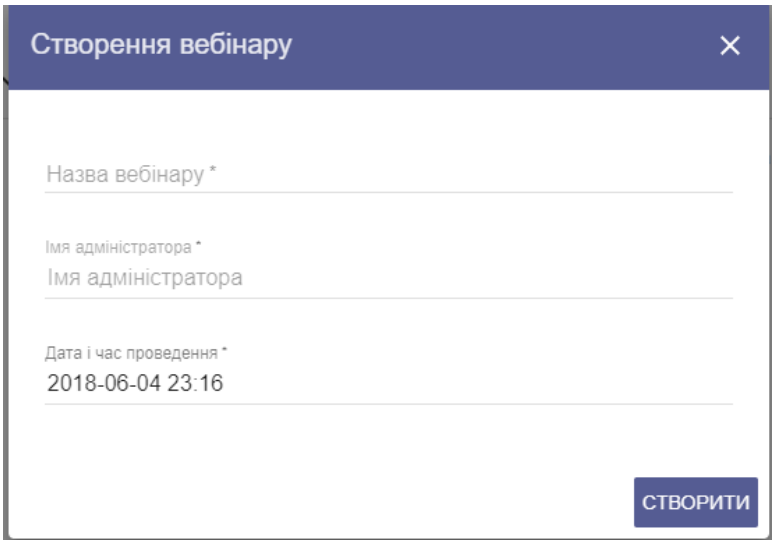


Рисунок 4.2 – Діалогове вікно створення нової конференції

Після створення конференції вона з'явиться на головній сторінці викладача, система автоматично показує скільки часу залишилося до початку конференції. Спочатку конференція перебуває в статусі очікування створення(рис. 4.3), згодом вона переходить в статус створена і повертає викладачу посилання адміністратора та звичайного слухача конференції.

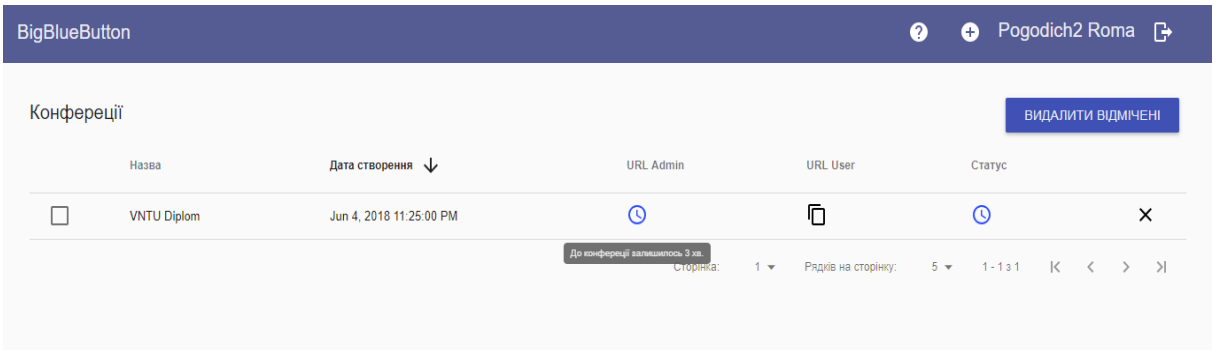


Рисунок 4.3 – Конференція перебуває в статусі очікування створення

Приклад посилання адміністратора системи: https://conference-distant.vnmu.edu.ua/bigbluebutton/api/join?meetingID=XY9MpxyT&fullName=Roma_Pogodich&password=xvLvei0i&checksum=1f27133a67ee47feaf9590ce12572ac234b6f551.

Приклад посилання користувача системи: https://webinar.vnmu.edu.ua/8zv6duouND0qd6P52pwDGPrM22fsuFrJ

Користувач одразу після переходу по своєму посиланню бачить сторінку приєднання до конференції(рис. 4.4).

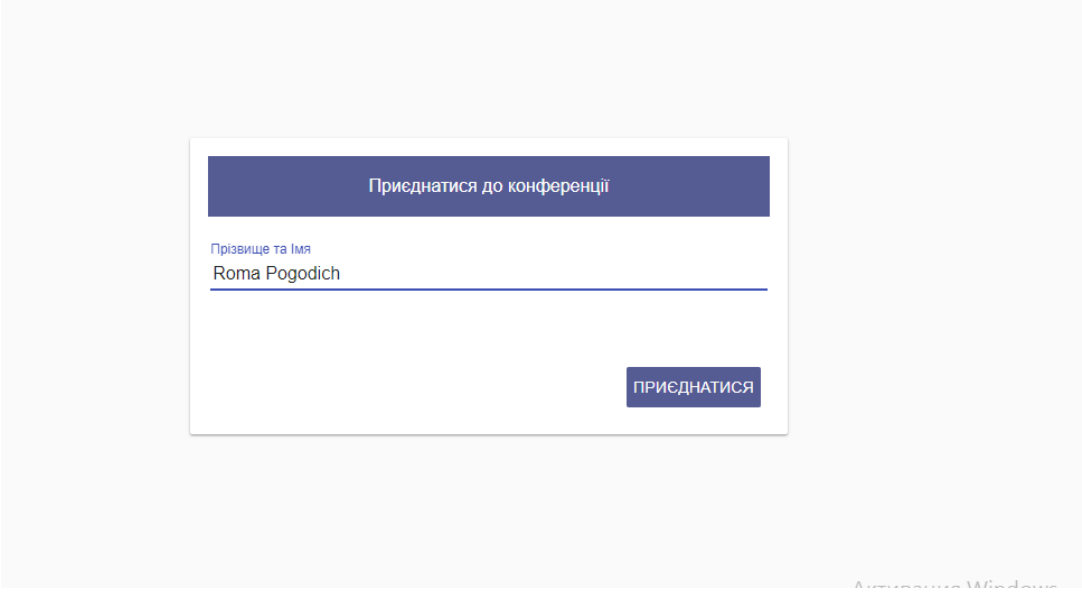


Рисунок 4.4 – Сторінка приєднання до конференції

Адміністратор системи бачить список всіх WEB-конференцій всіх користувачів системи, йому доступні всі посилання системи також він може видаляти конференції які створили викладачі системи(рис. 4.5). При великій кількості конференцій їх можна відсортувати по даті створення.

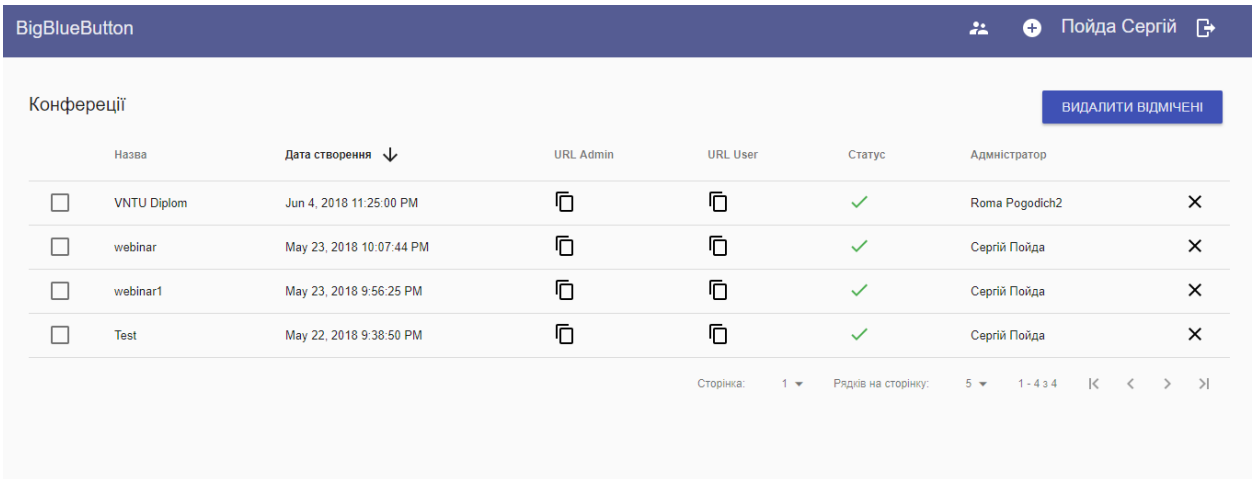


Рисунок 4.5 – Головна сторінка адміністратора системи

4.2 Методика тестування системи

Тестування програмного забезпечення - це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог, які описані в стандарті ISO 9126. Тестування - це процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою знайдення помилок[8].

В процесі розробки програмного проекту всі помилки відслідковувались на етапі розробки за допомогою IDE WebStorm.

SailsJS має інтуїтивно зрозумілий підхід до тестування, який приділяє особливу увагу поведінці додатка, а не деталям його реалізації, що є варіантом розробки через тестування (TDD) відомим як розробка через поведінку (BDD). Основними інструментами для цього є інтеграційні тести. Інтеграційні тести, відомі в контексті RSpec як request specs, дозволять нам симулювати дії користувача взаємодіє з нашим додатком через веб-браузер. Спільно з чудовим синтаксисом надаються Capybara, інтеграційні тести забезпечують потужну методику тестування функціоналу нашого застосування без необхідності вручну перевіряти кожну сторінку в браузері[9].

Основоположний принцип TDD це написання тестів до написання коду програми. Написання провального тесту і подальша реалізація коду програми для його проходження збільшує нашу впевненість у тому, що даний тест дійсно покриває функціональність для якої він був написаний. Крім того, цикл розробки провал-реалізація-проходження викликає потоковий стан, що призводить до високої продуктивності. Нарешті, тести відіграють роль клієнта для коду програми, що зазвичай призводить до більш елегантному дизайну веб-додатку.

4.3 Розгортання та впровадження системи

Для розгортання системи було вибрано хмарну платформу Heroku. Heroku — хмарна PaaS-платформа, що підтримує ряд мов програмування. Heroku, одна з перших хмарних платформ, з'явилась в червні 2007 року і спочатку підтримувала тільки мову програмування Ruby, але на даний момент список підтримуваних мов також включає в себе Java, Node.js, Scala, Clojure, Python і PHP. На серверах Heroku використовуються операційні системи Debian або Ubuntu (яка також заснована на Debian).

У якості технології передачі файлів обрано завантаження проекту в систему контролю версій GitHub, що дозволяє розгорнути проект через Heroku просто за посиланням на репозиторій GitHub.

Для того щоб розвернути систему на Herokuнеобхідно:

1. Необхідно встановити ПЗ для виконання команд Herokuв терміналі. Процес встановлення заключається в виконанні команди:
   ```
   user@host:~myapp$ wget -qO- https://toolbelt.heroku.com/install-ubuntu.sh
   ```
2. Необхідно ввійти в систему Herokuз командного рядка:
   ```
   user@host:~/sails/myapp$ heroku login
   Enter your Heroku credentials.
   Email: user@example.com
   Password:
   Could not find an existing public key.
   Would you like to generate one? [Yn]
   Generating new SSH public key.
   Uploading ssh public key /home/user/.ssh/id_rsa.pub
   ```
3. Ініціалізувати git-репозиторій:
   ```
   user@host:~/sails/myapp$ git init && git commit -am "init"
   ```
4. Створення додатку на heroku:
   ```
   user@host:~/sails/myapp$ heroku create myapp
   Creating myapp... done, stack is cedar
   ```

http://myapp.herokuapp.com/ | git@heroku.com:myapp.git

Git remote heroku added

Тепер система доступна за адресою myapp.herokuapp.com, але там ще поки нічого немає.

5. Розміщення додатку:

   user@host:~/sails/myapp$ git push heroku master

6. Виконуємо всі міграції бази даних:

   user@host:~/sails/myapp$ heroku run rake db:migrate

7. Можемо переглянути наш додаток виконавши команду:

   user@host:~/sails/myapp$ heroku open

Вимоги до складу і параметрів технічних засобів. У таблиці 4.1 наведено мінімальну та рекомендовану конфігурацію сервера для коректної роботи системи.

Таблиця 4.1 Мінімальні та рекомендовані вимоги до сервера

Найменування вимоги	Мінімальні вимоги	Рекомендовані вимоги
Процесор	1.0 Ghz	1.5 Ghz
Вільний простір на HDD	500 Mb	1 Gb
Оперативна пам’ять	512 Mb	1 Gb

4.4 Системне програмне забезпечення

Системне програмне забезпечення — відоміше як операційна система, це будь-яке програмне забезпечення, що забезпечує інфраструктуру, на якій можуть працювати прикладні програми, тобто воно керує і контролює комп'ютерним обладнанням, для можливості виконання прикладних програм.

Система для проведення дистанційного навчання написана з використанням високорівневого фреймворку Sails.

WEB-додаток — додаток, в якому клієнтом виступає браузер, а сервером — веб-сервер. Браузер є різновидом так званих тонких клієнтів. Браузер здатний відображати веб-сторінки і, як правило, входить до складу операційної

системи, а функції його оновлення і супроводу лежать на постачальнику операційної системи. Вся логіка додатку зосереджується на сервері, а браузер лише відображає інформацію, завантажену по мережі з серверу. Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи, і веб-додатки, таким чином, є міжплатформеними сервісами.

Істотною перевагою побудови веб додатків для підтримки стандартних функцій браузера полягає в тому, що функції повинні виконуватися, незалежно від операційної системи даного клієнта. Замість того, щоб писати різні версії для MS Windows, Mac OS X, GNU/Linux і інших операційних систем, додаток створюється один раз і розвертається на будь-якій платформі.

Для того щоб розвернути Sails додаток необхідно мати *nix подібну платформу та веб-сервер. Веб-сервер (англ. Web Server) — це сервер, що приймає HTTP-запити від клієнтів, зазвичай веб-браузерів, видає їм HTTP-відповіді, зазвичай разом з HTML-сторінкою, зображенням, файлом, медіа-потоком або іншими даними.

Вибір серверної операційної системи

Призначення серверної операційної системи – це управління додатками, що обслуговують всіх користувачів корпоративної мережі, а нерідко і зовнішніх користувачів. До таких додатків ставляться сучасні системи управління базами даних, засоби управління мережами та аналізу подій в мережі, служби каталогів, засоби обміну повідомленнями і групової роботи, Web-сервери, поштові сервери, корпоративні брандмауери, сервери додатків найрізноманітнішого призначення, серверні частини бізнес-додатків. Вимоги до продуктивності і надійності вказаних операційних систем дуже високі; нерідко сюди входять і підтримка кластерів (набору ряду однотипних комп'ютерів, які виконують одну й ту ж задачу й поділяють між собою навантаження), і можливості дублювання і резервування, і переконфігурації програмного і апаратного забезпечення без перезавантаження операційної системи.

Вибір серверної операційної системи і апаратної платформи для неї в першу чергу визначається тим, які додатки під її управлінням повинні виконуватися (як мінімум, вибрані додатки повинні існувати у версії для даної платформи) і які вимоги пред'являються до її продуктивності, надійності та доступності.

Проведемо порівняння двох популярних серверних операційних систем Windows Server і Ubuntu.

Багато комерційних організацій віддають перевагу саме операційним системам сімейства Windows. І не випадково. Корпорація Microsoft давно і успішно працює на ринку програмного забезпечення. Додатки, пропоновані нею для вирішення конкретних завдань, широкодоступним і ретельно протестовані. Погодьтеся, це серйозний плюс, коли ми вибираємо ОС для сервера. Розглянемо Microsoft server на прикладі конкретної операційної системи Windows Server 2008 R2.

Загалом, це універсальна система з величезним запасом продуктивності (забезпечує підтримку 256 логічних процесорів). Володіє власним інструментом резервного копіювання, що істотно підвищує надійність. Можна сказати, що саме вона є найкращим рішенням, якщо ми вибираємо ОС для сервера файлів або терміналів.

Основним недоліком можна назвати вимогливість до апаратної частини. Windows Server 2008 R2 вже не підтримує 32-х розрядну архітектуру, та й в цілому вимагає для роботи істотно більше ресурсів, ніж аналоги. Крім того, продукт вимагає придбання ліцензії. Слід враховувати також, що саме на ОС сімейства Windows орієнтована більшість існуючих шкідливих програм.

Таким чином, якщо у вашому розпорядженні сучасна апаратна частина, і ви можете дозволити собі покупку ліцензії та регулярне оновлення антивірусного ПЗ, то Windows Server - відмінний вибір.

На основі Debian розробляється безліч (більше сотні) самостійних дистрибутивів, і типовим представником є Ubuntu. Слід відразу сказати, що ця ОС не позиціонується як серверна. А починаючи з версії 12.04 навіть різниця

між серверним і десктопним ядром офіційно скасована, хоча пакети десктоп / сервер і мають деякі відмінності. Однак безліч людей у всьому світі віддають перевагу саме цьому представнику Linux дистрибутивів для вирішення серверних задач. Основна причина - простота установки і обслуговування, і просто величезна кількість фанатів, прийнято вважати, що «з установкою Ubuntu впорається навіть домогосподарка».

Якщо ми вибираємо операційну систему для сервера, на який не буде припадати занадто великих навантажень, якщо наш бюджет обмежений, і нам не потрібно забезпечувати Аптайм на рівні максимально близькому до 100%, а вільного часу не вистачає на читання безлічі документацій, то Ubuntu - цілком прийнятне рішення. Оскільки в мережі є маса готових how-to, і співтовариство цієї операційної системи вкрай чуйне і лояльне до новачків. Крім того дана ОС (як десктопна) широко використовується у всьому світі, регулярно оновлюється і має в своєму арсеналі безліч корисного і свіжого ПО.

Отже явну перевагу має Ubuntu.

4.5 Розробка засобів захисту системи від зовнішніх атак

Небезпека більшості DDoS-атак – в їх абсолютній прозорості і "нормальності". Адже якщо помилка в ПЗ завжди може бути виправлена, то повне зжирання ресурсів - явище майже буденне. З ними стикаються багато адміністраторів, коли ресурсів машини (ширина каналу) стає недостатньо, або веб-сайт піддається слешдот-ефекту. І якщо різати трафік і ресурси для всіх підряд, то врятуєшся від DDoS, но втратиш велику половину клієнтів.

Атака на відмову в обслуговуванні, розподілена атака на відмову в обслуговуванні (англ. DoS attack, DDoS attack, (Distributed) Denial-of-service attack) — напад на комп'ютерну систему з наміром зробити комп'ютерні ресурси недоступними користувачам, для яких комп'ютерна система була призначена.

Одним із найпоширеніших методів нападу є насичення атакованого комп'ютера або мережевого устаткування великою кількістю зовнішніх запитів (часто безглуздих або неправильно сформульованих) таким чином атаковане устаткування не може відповісти користувачам, або відповідає настільки повільно, що стає фактично недоступним.

Наслідки DDoS-атак і їх ефективність можна істотно понизити за рахунок правильного налаштування маршрутизатора, брандмауера і постійного аналізу аномалій в мережевому трафіку.

Є витонченіший спосіб захисту. Він заснований на організації розподіленої обчислювальної мережі, що включає безліч дублюючих серверів, які підключені до різних магістральних каналів. Коли обчислювальні потужності або пропускна спроможність каналу закінчуються, все нові клієнти перенаправляються на інший сервер (або ж поступово "розмазуються" по серверах за принципом round-robin).

Інше більш-менш ефективне рішення полягає в покупці дорогих хардварних систем Cisco Traffic Anomaly Detector і Cisco Guard. Працюючи у в'язці, вони можуть подавити атаку, що починається, але, як і більшість інших рішень, заснованих на вченні і аналізі станів, дають збої.

Отже, існує два типи DoS/DDoS-атак, і найбільш поширена з них заснована на ідеї флуда, тобто завалення жертви величезною кількістю пакетів. Флуд буває різним: ICMP-флуд, SYN-флуд, UDP-флуд і HTTP-флуд. Сучасні DoS-боти можуть використовувати всі ці види атак одночасно, тому слід заздалегідь поклопотатися про адекватний захист від кожної з них.

Icmp-флуд

Дуже примітивний метод забивання смуги пропускання і створення навантажень на мережевий стек через монотонну посилку запитів ICMP ECHO (пінг). Легко виявляється за допомогою аналізу потоків трафіку в обидві сторони: під час атаки типа Icmp-флуд вони практично ідентичні. Майже безболісний спосіб абсолютного захисту заснований на відключенні відповідей на запити ICMP ECHO:

#sysctl net.ipv4.icmp_echo_ignore_all=1

SYN-флуд

Один з поширених способів не лише забити канал зв'язку, але і ввести мережевий стек операційної системи в такий стан, коли він вже не зможе приймати нові запити на підключення. Заснований на спробі ініціалізації великого числа одночасних TCP-з'єднань через посилку SYN-пакету з неіснуючою зворотньою адресою. Після декількох спроб відіслати у відповідь ACK-пакет на недоступну адресу більшість операційних систем ставлять невстановлене з'єднання в чергу. І лише після n-ої спроби закривають з'єднання. Оскільки потік ACK-пакетів дуже великий, незабаром черга виявляється заповненою, і ядро дає відмову на спроби відкрити нове з'єднання. Найбільш розумні DoS-боти ще і аналізують систему перед початком атаки, щоб слати запити лише на відкриті життєво важливі порти. Ідентифікувати таку атаку просто: досить спробувати підключитися до одного з сервісів. Оборонні заходи зазвичай включають:

Збільшення черги "напіввідкритих" tcp-з'єднань:

sysctl -w net.ipv4.tcp_max_syn_backlog=1024

Зменшення часу утримання "напіввідкритих" з'єднань:

sysctl -w net.ipv4.tcp_synack_retries=1

Включення механізму TCP syncookies:

sysctl -w net.ipv4.tcp_syncookies=1

Обмеження максимального числа "напіввідкритих" з'єднань з одного IP до конкретного порту:

iptables -i INPUT -p tcp --syn --dport 80 -m iplimit --iplimit-above 10 -j DROP

HTTP-флуд

Один з найпоширеніших на сьогоднішній день способів флуда. Заснований на безконечній посилці http-повідомлень GET на 80-й порт з метою завантажити web-сервер настільки, щоб він виявився не в змозі обробляти всі останні запити. Часто метою флуда стає не корінь web-сервера, а один із скриптів, що виконують ресурсоємні завдання або що працює з базою даних. У

будь-якому випадку, індикатором атаки, що почалася, служитиме аномально швидке зростання лігв web-сервера.

Методи боротьби з Http-флудом включають тюнинг web-сервера і бази даних з метою понизити ефект від атаки, а також відсіювання DoS-ботів за допомогою різних прийомів. По-перше, слід збільшити максимальне число з'єднань до бази даних одночасно. По-друге, встановити перед web-сервером Apache легкий і продуктивний nginx – він кешуватиме запити і видавати статику. Це рішення із списку "Must have", яке не лише понизить ефект DoS-атак, але і дозволить серверу витримати величезні нагрузки. Невеликий приклад:

```
# vi /etc/nginx/nginx.conf
# Збільшує максимальну кількість використовуваних файлів
worker_rlimit_nofile 80000;
events {
# Збільшує максимальну кількість з'єднань
worker_connections 65536;
```

Щоб не попасти в безвихідь під час обвалення DDoS-штурму на системи, необхідно ретельно підготувати їх до такої ситуації:

1. Всі сервера, що мають прямий доступ в зовнішню мережу, мають бути підготовлені до простої і швидкої віддаленої передачі. Великим плюсом буде наявність другого, адміністративного, мережевого інтерфейсу, через який можна дістати доступ до сервера в разі затурканості основного каналу.

2. ПЗ, використовуване на сервері, завжди повинно знаходитися в актуальному стані. Всі дірки - пропатчені, оновлення встановлені. Це захистить від DoS-атак, експлуатуючих баги в сервісах.

3. Всі слухаючі мережеві сервіси, призначені для адміністративного використання, мають бути заховані брандмаузером від всіх, хто не повинен мати до них доступу. Тоді той, що атакує не зможе використовувати їх для проведення DoS-атаки або брутфорса.

4. На підходах до сервера (найближчому маршрутизаторі) має бути встановлена система аналізу трафіку (Netflow в допомогу), яка дозволить своєчасно дізнатися про атаку, що починається, і вчасно прийняти заходи по її запобіганню.

Потрібно додати в /etc/sysctl.conf наступні рядки:

vi /etc/sysctl.conf

Захист від спуфінга

net.ipv4.conf.default.rp_filter = 1

Перевіряти TCP-з’єднання кожну хвилину. Якщо на іншій стороні - легальна машина, вона зразу відповість. Значення по замовчуванні - 2 години.

net.ipv4.tcp_keepalive_time = 60

Повторити попитку через десять секунд net.ipv4.tcp_keepalive_intvl = 10

Кількість повірок перед закриттям з’єднання

net.ipv4.tcp_keepalive_probes = 5

4.6 Висновки

Тестування програми показало повну її працездатність та відповідність поставленому технічному завданню. Розроблено інструкцію користувача по розгортання системи на віддаленому сервері та представлено методи захисту системи від DoS атак в Інтернеті.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки. Для проведення технологічного аудиту було залучено 2-х незалежних експертів. Такими експертами будуть Майданюк В.П. та Войтко В.В.

Здійснюємо оцінювання комерційного потенціалу розробки за 12-ма критеріями за 5-ти бальною шкалою.

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.1.

Таблиця 5.1 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта	
	1. Експерт 1	2. Експерт 2
	Бали, виставлені експертами:	
1	4	4
2	4	3
3	3	4
4	4	3
5	3	4
6	4	4
7	3	3
8	4	4
9	4	4
10	4	3
11	3	4
12	3	4
Сума балів	$СБ_1 = 44$	$СБ_2 = 44$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 44$	

Отже, з отриманих даних таблиці 5.1 видно, що нова розробка має високий рівень комерційного потенціалу.

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи.

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою (5.1):

$$З_о = \frac{М}{Т_р} \cdot t, \quad (5.1)$$

де М- місячний посадовий оклад конкретного розробника;

$Т_р$ - кількість робочих днів у місяці, $Т_р$ = 21день;

t - число днів роботи розробника, t = 50 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 5.2.

Таблиця 5.2 – Розрахунки основної заробітної плати

Працівник	Оклад М, грн.	Оплата за робочий день, грн.	Число днів роботи, t	Витрати на оплату праці, грн.
Науковий керівник	5500	261,90	5	1309,5
Інженер-програміст	4000	190,47	50	9523,5
Всього:				10833

Розрахуємо додаткову заробітну плату:

$$З_{дод} = 0{,}1 \cdot 10833 = 1083{,}3 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 37,5…40% від суми їхньої основної та додаткової заробітної плати:

$$Н_{зп} = (З_о + З_р) \cdot \frac{\beta}{100}, \quad (5.2)$$

$$Н_{зп} = (10833 + 1083{,}3) \cdot \frac{36{,}3}{100} = 4325{,}5 \text{ (грн.)}.$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$А=\frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12}, \quad (5.3)$$

де Ц – балансова вартість обладнання, грн;

$Н_а$ – річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 5.3 – Розрахунок амортизаційних відрахувань

Найменування програмного забезпечення	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань,грн
Персональний комп'ютер	9000	25	3	562,5
Всього:				562,5

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою:

$$K = \sum_1^n Н_і \cdot Ц_і \cdot К_і, \quad (5.4)$$

де n – кількість комплектуючих;

Ні - кількість комплектуючих і-го виду;

Ці – покупна ціна комплектуючих і-го виду, грн;

Кі – коефіцієнт транспортних витрат (приймемо Кі = 1,1).

Таблиця 5.4 - Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Флешка	шт.	180	1	180
Пачка паперу	уп.	120	1	120
Ручка	шт.	10	1	10
Всього з урахуванням транспортних витрат				341

Витрати на силову електроенергію розраховуються за формулою:

$$В_е=В\cdot П\cdot Ф\cdot К_п ; \quad (5.5)$$

де В – вартість 1кВт-години електроенергії (В=1,7 грн/кВт);

П – установлена потужність комп'ютера (П=0,6кВт);

Ф – фактична кількість годин роботи комп'ютера (Ф=190 год.);

$К_п$ – коефіцієнт використання потужності ($К_п< 1$, $К_п = 0,7$).

$$В_е= 1,7 \cdot 0,6\cdot 190\cdot 0,7 = 135,66 \text{ (грн.)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$В_{ін} = (1..3) \cdot (З_о + З_р). \quad (5.6)$$

Отже, розрахуємо інші витрати:

$$В_{ін} = 1 * (10833 + 1083,3) = 11916,3 \text{ (грн.)}$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$В = З_о + З_д + Н_{зп} + А + К + В_е + І_в$$

В = 10833+1083,3+4325,5+562,5+341+135,66+11916,3= 29197,26 (грн.)

Розрахуємо загальну вартість наукової роботи $B_{заг}$ за формулою:

$$В_{заг} = \frac{В_{ін}}{\alpha}, \quad (5.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$В_{заг} = \frac{29197,26}{1} = 29197,26 \text{ (грн.)}$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta}, \quad (5.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{29197,26}{0,9} = 32441,4 \text{ (грн.)}$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки.

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості

грошей. Це забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності .

Оцінка зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + П_я \Delta N)_i \quad (5.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

Δ N – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення інформаційної технологіїзменшатьсяна 20 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 20 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 200 користувачів, протягом другого року – на 150 користувачів, протягом третього року – 100 користувачів. Реалізація інформаційної технологіїдо впровадження результатів наукової розробки складала 700 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 200 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_1 = 20 \cdot 700 + (200 + 20) \cdot 200 = 58000 \text{ грн.}$$

Протягом другого року:

$$\Delta П_2 = 20 \cdot 700 + (200 + 20) \cdot (200 + 150) = 91000 \text{ грн.}$$

Протягом третього року:

$$\Delta П_3 = 20 \cdot 700 + (200 + 20) \cdot (200 + 150 + 100) = 113000 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $Е_{абс}$ вкладених інвестицій розраховується за формулою:

$$Е_{абс} = (ПП - PV), \qquad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки 2, 3,4.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, рисунок 5.1.

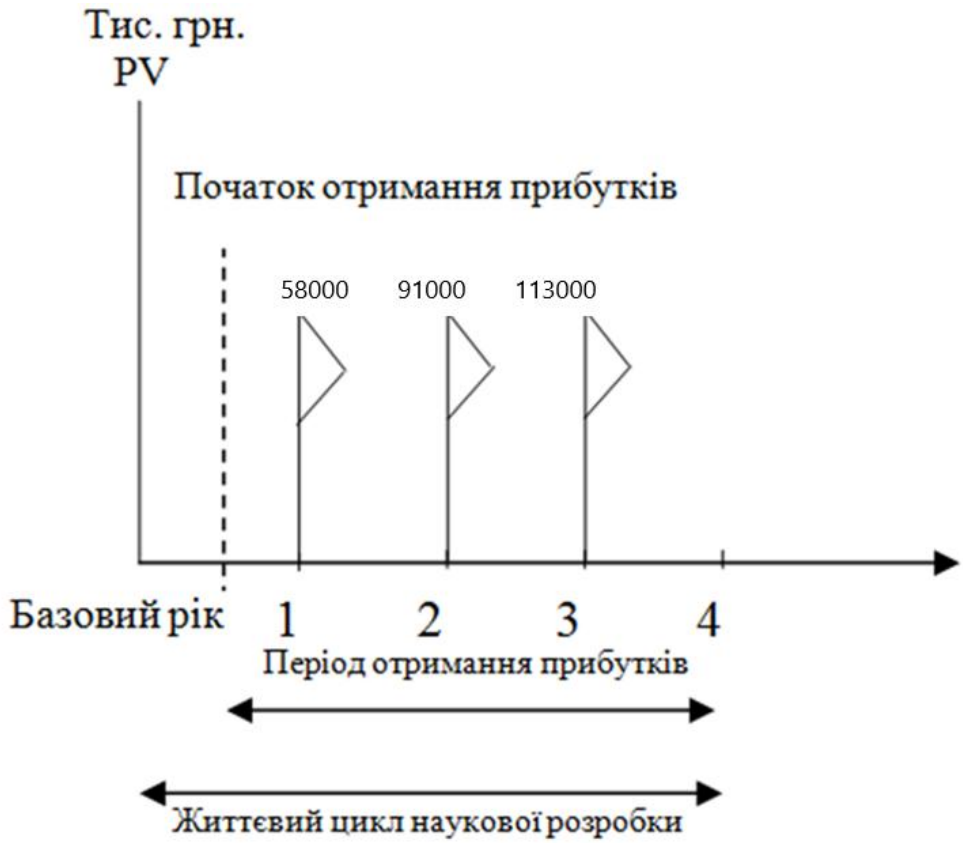


Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою:

$$ПП = \sum_1^m \frac{\Delta П_i}{(1+\tau)^t} \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{32441,4}{(1+0,1)^0} + \frac{58000}{(1+0,1)^2} + \frac{91000}{(1+0,1)^3} + \frac{113000}{(1+0,1)^4} = 225925,44 \text{ (грн.)}$$

Тоді розрахуємо $E_{абс}$:

$$Е_{абс} = 225925,44 - 32441,4 = 193484,04 \text{ грн.}$$

Оскільки $E_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$Е_в = \sqrt[Т]{1 + \frac{Е_{абс}}{PV}} - 1 \qquad (5.12)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$Е_в = \sqrt[3]{1 + \frac{193484,04}{32441,4}} - 1 = 0,90 \text{ або } 90\ \%$$

Далі, розраховану величина $Е_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0{,}2 + 0{,}1 = 0{,}3$$

Оскільки $E_в$ = 90% > $\tau_{мін}$ = 0,3 = 30%, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$розраховується за формулою:

$$T_{ок} = \frac{1}{E_в}$$

$$T_{ок} = \frac{1}{0{,}9} = 1{,}11 \text{ року}$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

5.3 Висновки

Проведено технологічний аудит з залученням двох незалежних експертів та визначено, що рівень комерційного потенціалу розробки знаходиться на рівні вище середнього.

Аналіз комерційного потенціалу розробки показав, що продукт за своїми характеристиками випереджає аналогічні рішення на ринку і є перспективною розробкою. Він забезпечує кращі функціональні показники, а тому є конкурентоспроможним товаром на ринку. Існуючі переваги нової розробки дозволять швидко її поширити та популяризувати.

Згідно із розрахунками всіх статей витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи загальні витрати на розробку складають 32441,4 грн.

Розрахована абсолютна ефективність вкладених інвестицій в сумі 193484,04 грн свідчить про отримання інвестором прибутку від комерціалізації програмного продукту та його зацікавленість в розробці.

Відносна ефективність вкладених в наукову розробку інвестицій складає 90%, що значно вище за мінімальну бар'єрну ставку дисконтування, яка знаходиться на рівні 30%. Це означає потенційну зацікавленість інвесторів у фінансуванні розробки.

Термін окупності вкладених у реалізацію наукового проекту інвестицій складе 1,11 року, що свідчить про швидку окупність вкладених інвестицій та доцільність фінансування нової розробки.

ВИСНОВКИ

Основні результати отримані при виконанні магістерської кваліфікаційної роботи такі:

1. Розроблено методи шифрування даних для WEB-системи дистанційного навчання. Шифрування відбувається на рівні сервера і на рівні бази даних. Також встановлено захищене з’єднання між клієнтом і сервером, що забезпечує безпечну передачу даних між ними. Розроблено засоби захисту від DDoS атак.
2. Подальшого розвитку отримав метод реалізації WEB-додатків з використанням архітектурного шаблону MVC(Model-View-Contrller), відмінністю якого є розбиття системи на 3 сутності – моделі даних, інтерфейс користувача та модулі керування, що надає можливість повторного використання окремих компонентів програми. Використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.
3. З використанням платформи Node.js, мови програмування Javascript та високорівневого фреймворку Sails.js розроблено захищену систему для проведення дистанційного навчання на основі WEB-конференцій.
4. Економічні розрахунки витрат та прибутків від впровадження розробки, термінів окупності та визначення її економічного ефекту показали, що в сучасній технічній ситуації розробка є конкурентоспроможною на ІТ-ринку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Шифрування даних [Електронний ресурс]. Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F
2. Grady B. Object-Oriented Analysis and Design with Applications/ B. Grady. – Amsterdam: Addison-Wesley Professional, 2009. – 720 p.
3. Horton's I. Beginning JavaSctipt/I. Horton's. –Amsterdam: Wrox, 2015. – 988 p.
4. Stellman A. Head First NodeJS / A. Stellman. – Boston: O'Reilly Media, 2014. – 784 p
5. MongoDb Documentation [Електронний ресурс]. Режим доступу до ресурсу: https://www.mongodb.com/
6. Sails.js Documentation [Електронний ресурс]. Режим доступу до ресурсу: https://sailsjs.com/
7. Angular.js Documentation [Електронний ресурс]. Режим доступу до ресурсу: https://angularjs.org/
8. Waterline Documentation [Електронний ресурс]. Режим доступу до ресурсу: http://waterlinejs.org/
9. Nodejs Documentation [Електронний ресурс]. Режим доступу до ресурсу: https://nodejs.org/en/
10. Processing. [Електронний ресурс] // Режим доступу: https://ru.wikipedia.org/wiki/Processing
11. Козловський В.О. Техніко-економічні обґрунтування та економічні розрахунки в дипломних проектах та роботах. Навчальний посібник. / В.О. Козловський – Вінниця: ВДТУ, 2003. – 75 с.
12. Козловський В.О. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт / В.О. Козловський – Вінниця: ВНТУ, 2012. – 22 с.

ДОДАТОК А Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"____ "_____2019 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка методів шифрування даних для WEB-системи дистанційного навчання»

за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

_____ к.т.н., доц. В. П. Майданюк

"____"_____2019 р.

Виконав:

_____студент гр.1ПІ-18м Р.В. Погодич

"____"_____2019 р.

Вінниця – 2019 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів шифрування даних для WEB-системи дистанційного навчання».

Галузь застосування - системи для дистанційного навчання.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № __ ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення рівня захисту даних від несанкціонованого доступу у WEB-системах дистанційного навчання за рахунок застосування криптографічних методів та автоматизації процесу навчання

Призначення роботи – розробка методів і засобів захисту даних у WEB-системі дистанційного навчання.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Шифрування даних [Електронний ресурс]. Режим доступу до ресурсу:https://uk.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F
2. Grady B. Object-Oriented Analysis and Design with Applications/ B. Grady. – Amsterdam: Addison-Wesley Professional, 2009. – 720 p.
3. Horton's I. Beginning JavaSctipt/I. Horton's. –Amsterdam: Wrox, 2015. – 988 p.
4. Stellman A. Head First NodeJS / A. Stellman. – Boston: O'Reilly Media, 2014. – 784 p

5. Технічні вимоги

- Інтегроване середовище розробки - WebStorm;
- Мова програмування - JavaScript;
- База даних – MongoDB;
- Фреймворк – Sails.js

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Обґрунтування вибору методу розробки та постановка задачі дослідження	
2	Розробка системи захисту web-системи для проведення дистанційного навчання	
3	Розробка програмних модулів системи дистанційного навчання на основі web-конференцій	
4	Тестування роботи системи дистанційного навчання на основі web-конференцій	
5	Економічна частина	

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б Акт впровадження

ДОДАТОК В Лістинг програмних модулів

AuthController.js

```
const _ = require('lodash');
const jwt = require('jsonwebtoken');

module.exports = {
    login(req, res) {
        const user = _.pick(req.body, ['login', 'password']);

        if (!user.login || !user.password) {
            return res.send(401, {message: 'Не всі обовязкові поля заповненні'});
        }

        Users.findOne({login: user.login}).then(foundUser => {
            if (!foundUser) {
                return res.send(401, {message: 'Невірний логін або пароль'});
            }

            if (foundUser.password !== user.password) {
                return res.send(401, {message: 'Невірний пароль'});
            }

            foundUser.token = jwt.sign({user: foundUser}, 'TOKEN_SECRET_BBB', {expiresIn: '7d'});

            return res.send(200, {foundUser});
        }).catch((error) => {
            sails.log.error(`Catched error ${error}`)
        });
    },

    ['login-token'](req, res) {
        const token = req.query.token;

        if(!token) {
            return res.send(401, {message: 'Token not found!'});
        }

        jwt.verify(token, 'TOKEN_SECRET_BBB', (error, decoded) => {
            if (error) {
                return res.send(401, {message: 'Invalid token!'});
            }

            return res.send(200, {user: decoded.user});
        });

    }
};
```

ConferenceController.js

```
const bbb = require('nodejs-bigbluebutton');
const randomString = require('randomstring');

module.exports = {
    create(req, res) {
        const newConference = req.body;
        newConference.clientUrl = randomString.generate();
        newConference.userId = req.user.id;
        newConference.adminName = newConference.adminName.replace(/ /g, '_');
        newConference.creatorFullName = `${req.user.firstName}
${req.user.lastName}`;

        Conference.create(newConference, (error, result) => {
            if (error) {
                res.send(error.message)
            }
            res.send(200, result);
        })
    },

    getAll(req, res) {
        const {user} = req;
        let query;
        if (user.role === 'admin') {
            query = Conference.find({});
        } else {
            query = Conference.find({userId: user.id});
        }

        query.sort('createdTime ASC');
        query.exec((error, conferences) => {
            if (error) {
                res.badRequest();
            }

            conferences = conferences.map((conference) => {
                if (conference.clientUrl) {
                    conference.clientUrl = req.headers.host.concat('/',
conference.clientUrl);
                }
                return conference;
            });

            res.send(200, conferences);
        });
    },
    find(req, res) {
        Conference.findOne({clientUrl: req.query.clientUrl, conferenceIsRunning:
true}).exec((err, responce) => {
            if (err) {
                res.badRequest();
            }
```

```
            if (!responce) {
                return res.notFound();
            }

            res.send(200, {meetingID: responce.meetingID});
        });
    },
    deleteSelected(req, res) {
        Conference.destroy({id: {$in: req.body}}).then(success => {
            return res.send(200, 'ok')
        }).catch(error => {
            sails.log.error(error);
            res.send(error);
        })
    },
    join(req, res) {
        const fullName = req.body.fullName.replace(/ /g, '_');
        const meetingId = req.body.conference.meetingID;

        Conference.findOne({meetingID:      meetingId,      conferenceIsRunning:
true}).exec((err, responce) => {
            if (err) {
                return res.badRequest();
            }
            if (!responce) {
                return res.notFound();
            }

            Settings.findOrCreate({userId: responce.userId},
                {
                    userId: responce.userId,
                    salt: '718d4da70b261be47e282f0020e7dc3d',
                    url: 'https://conference-distant.vnmu.edu.ua/bigbluebutton/'
                }).exec((error, data) => {
                bbb.salt = data.salt;
                bbb.url = data.url;

                bbb.join({
                    meetingID: responce.meetingID,
                    fullName: fullName,
                    password: responce.attendeePW,
                    webVoiceConf: ''
                }, (bbbLink) => {
                    return res.json(200, {url: bbbLink.data});
                })

            });
        });
    }
};
```

UsersController.js

```
const _ = require('lodash');
```

```
module.exports = {
    create(req, res) {
        const newUser = _.pick(req.body, ['firstName', 'lastName', 'email', 'login', 'password']);
        if (!newUser.email || !newUser.login || !newUser.password || !newUser.firstName || !newUser.lastName) {
            return res.send(401, {message: 'Не всі обовязкові поля заповненні'});
        }
        if (!(_.isString(newUser.password) && newUser.password.length >= 6 && newUser.password.match(/[a-z]/i) && newUser.password.match(/[0-9]/))) {
            return res.send(401, {message: 'Пароль має бути не менше 6 символів довжиною, містити цифри і літери латинського алфавіту.'});
        }
        if (req.body.id) {
            Users.update({id: req.body.id}, req.body).then(user => {
                return res.send(200, user[0])
            }).catch(error => {
                sails.log.error(error);
                res.send(error);
            })
        } else {
            Users.findOne({login: newUser.login}).then(foundUser => {
                if (foundUser) {
                    return res.send(400, {message: `User with login - ${newUser.login} already exists!`})
                }
                Users.create(newUser).then(user => {
                    return res.send(200, user)
                }).catch(error => {
                    sails.log.error(error);
                    res.send(error);
                })
    },
    getAll(req, res) {
        const {user} = req;
        let query;
        if (user.role === 'admin') {
            query = Users.find({});
        } else {
            console.log('FORBITTEN');
        }
        query.sort('createdTime DESC');
        query.exec((error, users) => {
            if (error) {
                res.badRequest();
            }

            res.send(200, users);
        });
```

ConferenceModel.js

```
module.exports = {
```

```
  attributes: {
      meetingID: {
        type: 'string',
        required: true
      },
      meetingName: {
        type: 'string',
        required: true
      },
      attendeePW:{
        type: 'string',
        minLength: 2
      },
      moderatorPW: {
        type: 'string'
      },
      adminUrl: {
        type: 'string'
      },
      clientUrl:{
        type: 'string'
      },
      createdTime: {
          type: 'datetime'
      },
      conferenceIsRunning: {
          type: 'boolean',
          defaultsTo: false
      },
      userId: {
          type: 'string'
      },
      creatorFullName: {
          type: 'string'
      }
  }
};
```

StartConferenceJob.js

```
const moment = require('moment');
module.exports = (agenda) => {
    const job = {
        frequency: 'every 1 minute',
        run: (job, done) => {
            Conference.find({
                createdTime:     {'>':     moment.utc().toISOString(),     '<':
moment.utc().add(1, 'minutes').toISOString()},
                conferenceIsRunning: false
            }).exec((err, newConference) => {
                if (err) {
                    sails.log.error('Job was not executed properly' + err);
```

```
                }

                if (newConference.length) {
                    newConference.forEach(conference => {
                        sails.log.debug('Start created conference ->',
conference.meetingName);
                        ConferenceService.createConference(conference);
                    });
                }
                done();
            });
        }
    };
    return job;
};
```

isAuthenticated.js

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
    if (req.headers && req.headers.authorization) {
        const token = req.headers.authorization;
         jwt.verify(token, 'TOKEN_SECRET_BBB', (error, decodedData) => {
            if (error) {
                return res.send(401, {message: 'Invalid Token!'})
            }

            req.user = decodedData.user;
            return next();
         })
    } else {
        return res.send(403, {message: 'You don\'t have permission to do
this!'});
    }
};
```

ClientConferenceController.js

```
angular.module('app').controller('ConferenceController', ['$scope', '$mdDialog',
'ConferenceService', 'SettingsService', 'AuthService', 'toastr', '$stateParams',
'$state', '$window', '$rootScope', 'store', '$http', '$timeout',
    function ($scope, $mdDialog, ConferenceService, SettingsService,
AuthService, toastr, $stateParams, $state, $window, $rootScope, store, $http,
$timeout) {

        $scope.selected = [];
        $scope.minDate = moment().add(2, 'minutes');
        $scope.customFullscreen = false;
        let intervalId = null;

        $scope.conference = {
```

```
            meetingName: '',
            meetingID: '',
            attendeePW: '',
            moderatorPW: '',
            adminName: '',
            createdTime: moment().add(2, 'minutes')
        };

        $scope.query = {
            order: '-createdTime',
            limit: 5,
            page: 1
        };

        const getConferences = () => {
            ConferenceService.getAll.query({}).$promise.then((conference) => {
                $scope.conferences = JSON.parse(JSON.stringify(conference));
            });
        };

        if ($state.current.name !== 'userlink') {

            $scope.$watch(() => {
                return AuthService.getIsAthenticated()
            }, (newVal => {
                if (newVal) {
                    $scope.user = AuthService.getCurrentUser();
                    getConferences();
                }
            }));

            // SettingsService.crud.get({id: $rootScope.user.id }, (data) => {
            //     $scope.settings = JSON.parse(angular.toJson(data))[0];
            // });
        } else {

            ConferenceService.setUserLink(true);
            ConferenceService.crud.get({clientUrl:  $stateParams.id},  (responce)
=> {
                $scope.currentConference = responce;
            }, (error) => {
            });
        }

        $scope.$on('newConferenceCreated', (event, data) => {
            getConferences();
        });

        $scope.calcDiffInMinutes = (compareTo) => {
            return moment(compareTo).diff(moment(), 'minutes')
        };

        $scope.user = {};
```

```
        const isRunningConference = (conferences) => {
            let present = false;

            conferences.forEach((conference) => {
                const diff = $scope.calcDiffInMinutes(conference.createdTime);
                if (!conference.isRunning && diff >= 0 && diff < 30) {
                    present = true;
                }
            });

            return present;
        };

        const updateConferencesView = () => {
            intervalId = setInterval(() => {
                getConferences();
            }, 60000)
        };

        $scope.saveSettings = (validForm, selectedUser) => {
            $scope.settings.userId = selectedUser.id;
            if (validForm) {
                SettingsService.crud.save($scope.settings, () => {
                    $mdDialog.hide();
                    toastr.success('Налаштування збережені');
                });
            } else {
                toastr.error('Форма містить помилки');
            }
        };

        $scope.copiedToastr = () => {
            toastr.success('Посилання скопійовано в буфер обміну');
        };

        $scope.conferencesToDelete = [];

        $scope.addToDelete = (id) => {
            if (!$scope.conferencesToDelete.includes(id)) {
                $scope.conferencesToDelete.push(id);
            } else {
                const index = $scope.conferencesToDelete.indexOf(id);
                $scope.conferencesToDelete.splice(index, 1);
            }
        };

        $scope.removeSelectedConferences = () => {
            console.log($scope.conferencesToDelete);
            ConferenceService.deleteSelected.save($scope.conferencesToDelete, ()
=> {
                toastr.success('Конференції видалено');
                getConferences();
```

```
                $mdDialog.hide();
            });
        };

        const randomString = (length) => {
            let text = '';
            const possible = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';

            for (let i = 0; i < length; i++)
                text += possible.charAt(Math.floor(Math.random() * possible.length));

            return text;
        };


        $scope.createConference = (validForm) => {
            if (validForm) {
                $scope.isCloseButton = false;
                $scope.conference.moderatorPW = randomString(8);
                $scope.conference.attendeePW = randomString(8);
                $scope.conference.meetingID = randomString(8);
                $scope.conference.createdTime = moment.utc($scope.conference.createdTime).toISOString();
                ConferenceService.crud.save($scope.conference, (newConference) => {
                    toastr.success('Конференцію створено');
                    $mdDialog.hide();
                });
            } else {
                toastr.error('Не всі поля заповнені');
            }
        };

        $scope.join = (formValid) => {
            if (formValid) {
                $scope.user.conference = $scope.currentConference;
                ConferenceService.join.save($scope.user, (clientUrl) => {
                    $window.location.href = clientUrl.url;
                })
            } else {
                toastr.error('Не всі поля заповнені')
            }
        };

        $scope.deleteConference = (ev, conference) => {
            const confirm = $mdDialog.confirm()
                .title('Видалення конференції')
                .textContent('Ви дійсно бажаєте видалити конференцію?')
                .targetEvent(ev)
                .ok('ТАК')
                .cancel('НІ');
```

ДОДАТОК Г Ілюстративний матеріал
до захисту магістерської кваліфікаційної роботи

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Завідувач кафедри ПЗ, д. т. н., професор _____ О. Н. Романюк

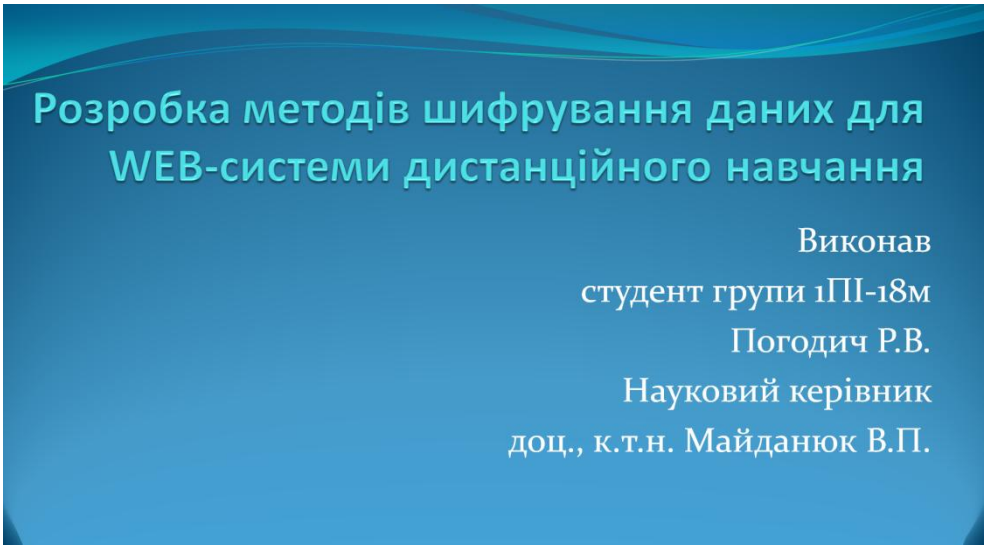
Науковий керівник, к.т.н., доцент кафедри ПЗ _____ В. П. Майданюк

Рецензент, к.т.н, доцент кафедри КН _____ Л. В. Крилик

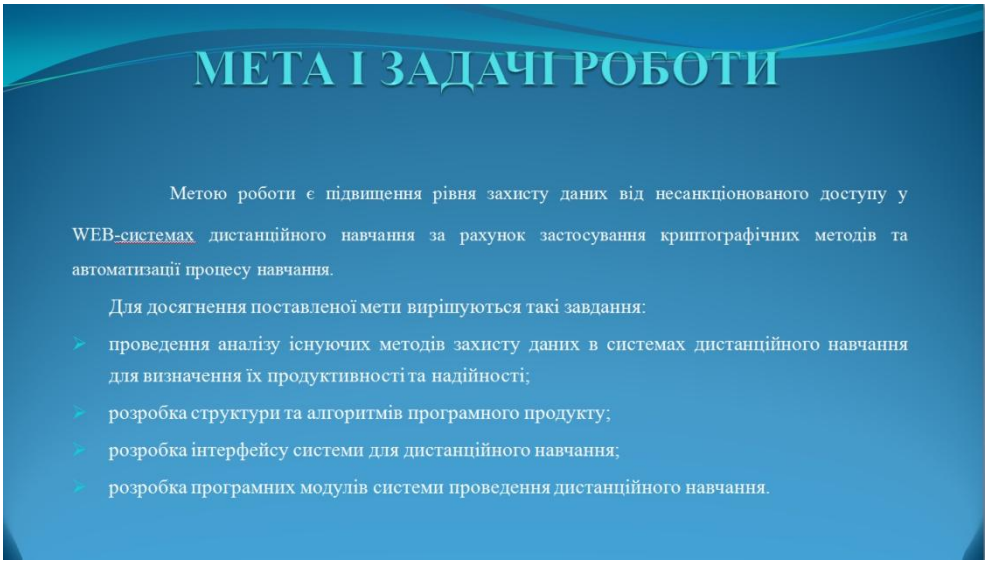
Нормоконтроль, к.т.н., доцент кафедри ПЗ _____ В. П. Майданюк

Виконавець, студент групи 1ПІ-18м _____ Р. В. Погодич

Слайд 1 – Тема і автор МКР



Слайд 2 – Мета і задачі роботи



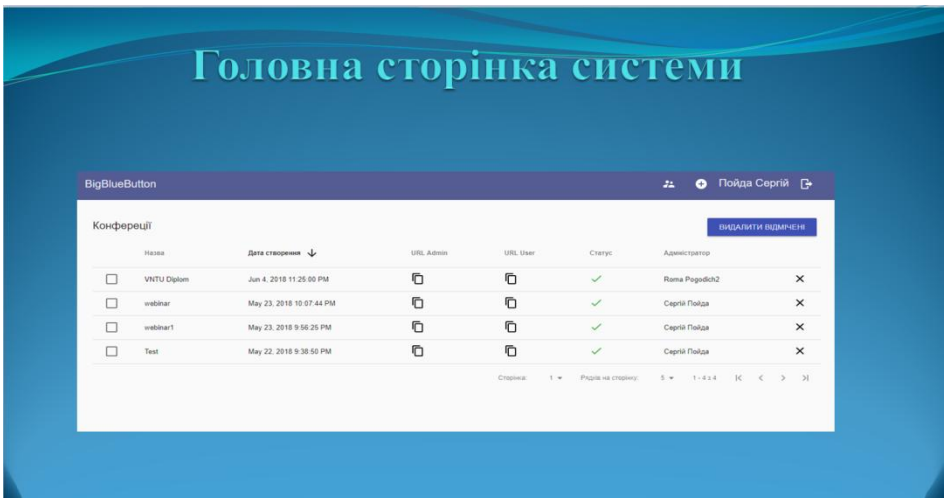
Слайд 3 – Симетричне шифрування даних

Слайд 4 – Асиметричне шифрування даних

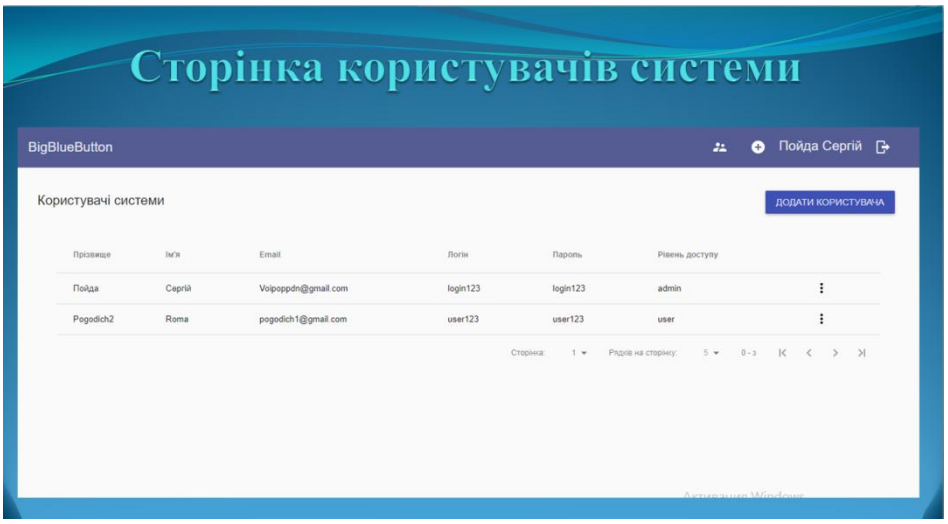
Слайд 5 – Діаграма потоків даних системи

Слайд 6 – ER- діаграма системи

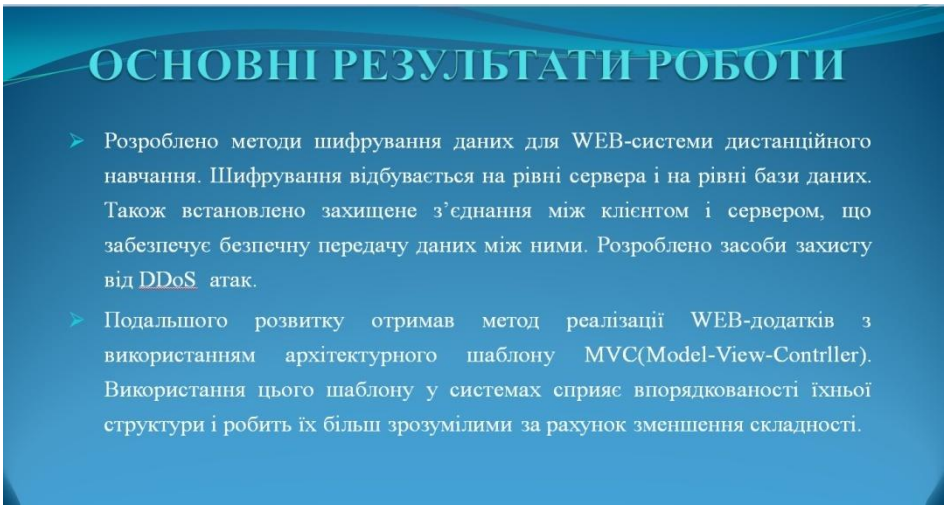
Слайд 7 – Головна сторінка системи



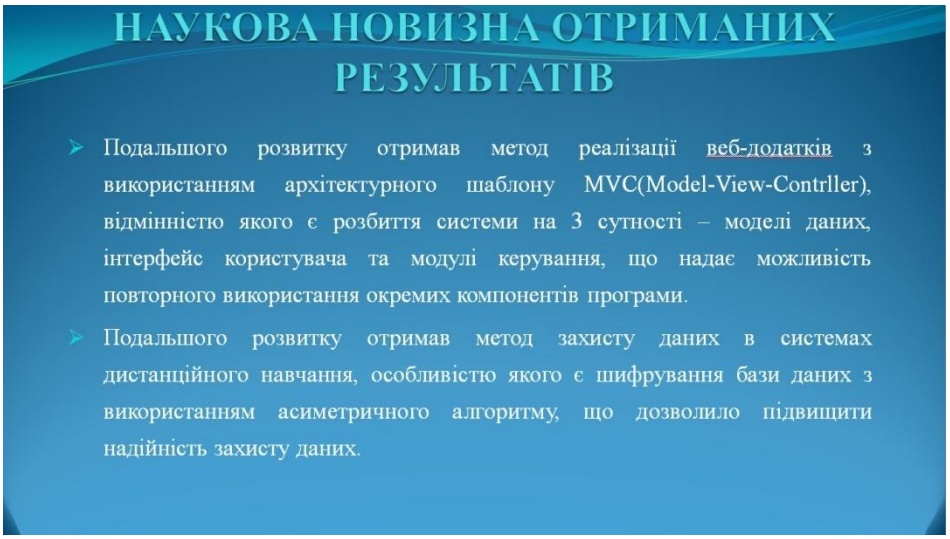
Слайд 8 – Сторінка користувачів системи



Слайд 9 – Основні результати роботи



Слайд 10 - Наукова новизна отриманих результатів



Слайд 11 – Дякую за увагу