МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ

До захисту допускається
Завідувач кафедри
_____ Скарга-Бандурова І.С.
«____»_____ 20__ р.

МАГІСТЕРСЬКА РОБОТА

НА ТЕМУ:

Інформаційні технології в системі управління проектними роботами

Освітньо-кваліфікаційний рівень "Магістр"
Спеціальність 122 "Комп'ютерні науки та інформаційні технології" (освітня програма - "Інформаційні технології проектування")

Науковий керівник роботи: _____ (підпис) Г.Ф.Кривуля (ініціали, прізвище)

Консультант з охорони праці: _____ (підпис) Я.О.Критська (ініціали, прізвище)

Студент: _____ (підпис) О.В.Кравченко (ініціали, прізвище)

Група: ІТП-16зм

Сєвєродонецьк 2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Комп'ютерних наук та інженерії
Освітньо-кваліфікаційний рівень магістр
Напрям підготовки _____
(шифр і назва)
Спеціальність 122 "Комп'ютерні науки та інформаційні технології" (освітня програма-
(шифр і назва)
"Інформаційні технології проектування")

ЗАТВЕРДЖУЮ:

Завідувач кафедри _____
_____ І.С. Скарга-Бандурова
«_____» _____20____р.

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

Кравченку Олексію Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційні технології в системі управління проектними роботами

керівник проекту (роботи) Кривуля Геннадій Федорович, д.т.н., проф.
(прізвище, м.'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «18» 10 2018 р. № 208/48
2. Строк подання студентом роботи 21.01.2018
3. Вихідні дані до роботи Матеріали науково-дослідної практики, сервіси та технології: Google Cloud Services, Amazon Web Services, OpenStack, EC2, ELB, Google Docs, Google Drive, Python Tornado, React, Redux

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Огляд хмарних технологій, система «Кабінет курсового проектування», організація клієнт-серверних додатків, реалізація системи «Кабінет курсового проектування», охорона праці та безпека в надзвичайних ситуаціях, висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Електронні плакати

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці та безпека в надзвичайних ситуаціях	Критська Я.О. ст. викл. кафедри КНІ		

7. Дата видачі завдання 18.10.2017

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Розробка технічного завдання	10.09.2017-15.09.2017	
2	Огляд хмарних технологій та сервісів	16.09.2017-22.09.2017	
3	Організація клієнт-вервисних додатків	23.09.2017-25.09.2017	
4	Проектування ситеми «Кабінет курсового проектування»	26.09.2017-06.10.2017	
5	Розробка частини проекту "Охорона праці та безпеки в надзвичайних ситуаціях"	07.10.2017-25.10.2017	
6	Оформлення пояснювальної записки та презентації	26.10.2017-13.12.2007	
7	Оформлення автореферату	14.12.2017-30.12.2017	

Студент _____ Кравченко О.В.
(підпис) (прізвище та ініціали)

Науковий керівник _____ Кривуля Г.Ф.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кравченко О.В. Інформаційні технології в системі управління проектними роботами.

Метою магістерської роботи є покращення організації та контролю самостійної роботи студентів завдяки розробці системи управління проектною діяльністю студентів з використанням хмарних технологій. В якості хостінгу використано хмарну технологію Amazon Web Services з сервісом EC2. Результатом роботи стала система контролю проектної діяльності студентів.

Ключові слова: AWS, GCS, SAAS, EC2, Google Cloud Services, шаблон MVC, технологія REDUX.

АННОТАЦИЯ

Кравченко А.В. Информационные технологи в системе управления проектными работами

Целью магистерской работы является улучшение организации и контроля самостоятельной работы студентов благодаря разработке системы управления проектной деятельностью студентов с использованием облачных технологий. В качестве хостинга использовано облачную технологию Amazon Web Services с сервисом EC2. Результатом работы стала система контроля проектной деятельности студентов.

Ключевые слова: AWS, GCS, SAAS, EC2, Google Cloud Services, шаблон MVC, технология REDUX.

THE ABSTRACT

Kravchenko O.V. Information technology in project management system.

The aim of the master's thesis is to improve the organization and control of students' independent work through the development of a project management system for students using cloud technologies. Amazon Web Services cloud service with EC2 service is used as hosting. The result of the work was the system of control of the project activity of students.

Key words: AWS, GCS, SAAS, EC2, Google Cloud Services, MVC pattern, REDUX technology.

ЗМІСТ

ВСТУП

З ростом все більшої і більшої зайнятості студентів, а також збільшенням навантаження на викладачів, досить сильно починає страждати навчальний процес, оскільки студенту та викладачу досить важко зустрітися особисто.

З урахуванням того, що технології за останній час дуже активно розвиваються, зараз існує величезна кількість так званих хмарних технологій (обчислень), які дозволяють користувачеві використовувати певні обчислювальні або програмні ресурси без встановлення будь-якого певного програмного забезпечення або ж не маючи потужного комп'ютера з високими обчислювальними можливостями. Передовими в цій області можна вважати такі корпорації як Google і Amazon. Також ще є величезна кількість більш невеликих і менш відомих компаній, але нам цілком вистачить тих можливостей які нам надають ці два гіганти.

Використання хмарних технологій є дуже зручним, оскільки сам провайдер гарантує їх працездатність і надійність, а в разі неполадок - досить швидко особисто усуває їх. Хоча зіткнутися з тими чи іншими проблемами досить складно, оскільки все реалізовано на дуже високому рівні.

Ряд технологій надаються Google: Google Docs, Google Sheets, Google Slides - повноцінно замінюють Microsoft Office і при цьому поширюються абсолютно безкоштовно. Якщо врахувати що ми також отримуємо і безкоштовне сховище даних Google Drive, то можна вважати, що у нас вже є повноцінна хмарна платформа для реалізації віддаленого освітнього процесу.

Кожен провайдер хмарних технологій також надає певне програмне API для програмної взаємодії з його технологіями. Це створює відмінну платформу для розробників, оскільки у них з'являється можливість інтегруватися з вищезгаданими технологіями і реалізовувати ще більш технологічні і корисні додатки.

Об'єкт дослідження - проектна діяльність студентів в рамках роботи в LMS.

Предмет дослідження - моделі, технології та архітектурні рішення, використовувані при розробці клієнт-серверних освітніх систем з використанням хмарних технологій.

Мета дослідження – покращення організації та контролю самостійної роботи студентів завдяки розробці системи управління проектною діяльністю студентів з використанням хмарних технологій.

Для досягнення поставленої мети **необхідно вирішити такі завдання.**

1) розробити загальну структуру системи, описати систему у вигляді ряду модулів, описати взаємодія між ними;

2) розробити принципи взаємодії зі сторонніми сервісами, провести дослідження інших рішень;

3) розробити систему аутентифікації, пов'язану з доменом snu.edu.ua;

4) сформулювати вимоги до системи засновані на користувальницьких історіях;

5) розробити архітектуру системи, а також архітектуру бази даних на основі MongoDB;

6) реалізувати систему і провести інтеграцію зі сторонніми сервісами.

Публікації. Основні результати магістерської роботи доповідались на Міжнародній науково-практичній конференції «Майбутній науковець – 2017», та на Всеукраїнській науково-практичній конференції «Електронні апарати та системи. Проблеми створення. Перспективи розвитку».

Структура і обсяг роботи. Магістерська атестаційна робота складається з вступу, 5 розділів, висновків, переліку джерел посилань, 2 додатків. Загальний обсяг роботи становить 94 сторінок, 1 таблиця, 12 рисунків.

1.1 ОГЛЯД ХМАРНИХ ТЕХНОЛОГІЇ

Існує досить багато сервісів, що надають різні хмарні технології, яких у світі існує величезна безліч. Вони все більше і більше входять в повсякденне життя людей і не дивлячись на це багато людей навіть не мають на увазі, що використовують ті чи інші хмарні технології. Перш за все варто зрозуміти, що таке хмарні технології (обчислення). Хмарні обчислення - інформаційно-технологічна концепція, що передбачає забезпечення повседенного та зручного мережевого доступу на вимогу до загального пулу конфігурованих обчислювальних ресурсів (наприклад, мереж передачі даних, серверів, пристроїв зберігання даних, додатків і сервісів - як разом, так і окремо), які можуть бути оперативно надані та звільнені з мінімальними експлуатаційними витратами або зверненнями до провайдера.

Первісний поділ хмар можна зробити за моделлю розгортання хмари [1].

Приватна хмара (англ. Private cloud) - інфраструктура, призначена для використання однією організацією, що включає кілька споживачів (наприклад, підрозділів однієї організації), можливо також клієнтами і підрядниками даної організації. Приватна хмара може перебувати у власності, управління та експлуатації як самої організації, так і третьої сторони (або будь-якої їх комбінації), і вона може фізично існувати як всередині, так і поза юрисдикцією власника.

Публічна хмара (англ. Public cloud) - інфраструктура, призначена для вільного використання широкою публікою. Публічна хмара може перебувати у власності, управлінні та експлуатації комерційних, наукових і урядових організацій (або будь-якої їх комбінації). Публічна хмара фізично існує в юрисдикції власника - постачальника послуг.

Суспільна хмара (англ. Community cloud) - вид інфраструктури, призначений для використання конкретною спільнотою споживачів з організацій, що мають спільні завдання (наприклад, місії, вимог безпеки, політики, і відповідності різним вимогам). Громадська хмара може перебувати в кооперативній (спільної) власності, управлінні та експлуатації однієї або більше з організацій спільноти або третьої сторони (або будь-якої їх комбінації), і вона може фізично існувати як всередині, так і поза юрисдикцією власника.

Гібридна хмара (англ. Hybrid cloud) - це комбінація з двох або більше різних хмарних інфраструктур (приватних, публічних або суспільних), що залишаються унікальними об'єктами, але пов'язаних між собою стандартизованими або приватними

технологіями передачі даних і додатків (наприклад, короткочасне використання ресурсів публічних хмар для балансування навантаження між хмарами).

Наступним кроком буде розуміння, що ж саме може надати нам ту чи іншу хмару, звідси з'являється ще одна класифікація хмарних технологій (за моделлю обслуговування) [2].

Програмне забезпечення як сервіс (Software as a Service / SaaS) - модель, в якій споживачеві надається можливість використання прикладного програмного забезпечення провайдера, який працює в хмарної інфраструктурі і доступного з різних клієнтських пристроїв або за допомогою тонкого клієнта, наприклад, з браузера (наприклад, Gmail, Google Docs, Google Calendar, etc.) або за допомогою інтерфейсу програми. Контроль і управління основної фізичної і віртуальної інфраструктури хмари, в тому числі мережі, серверів, операційних систем, зберігання, або навіть індивідуальних можливостей додатку (за винятком обмеженого набору призначених для користувача налаштувань конфігурації програми) здійснюється хмарним провайдером.

Платформа як сервіс (Platform as a Service / PaaS) - модель, коли споживачеві надається можливість використання хмарної інфраструктури для розміщення базового програмного забезпечення для подальшого розміщення на ньому нових або існуючих додатків (власних, розроблених на замовлення або придбаних тиражованих додатків). До складу таких платформ входять інструментальні засоби створення, тестування і виконання прикладного програмного забезпечення - системи управління базами даних, сполучна програмного забезпечення, середовища виконання мов програмування - надаються хмарним провайдером. Контроль і управління основною фізичною і віртуальною інфраструктурою хмари, в тому числі мережі, серверів, операційних систем, зберігання здійснюється хмарним провайдером, за винятком розроблених або встановлених додатків, а також, по можливості, параметрів конфігурації середовища (платформи). Найбільш придатними прикладами є Google App Engine і Heroku.

Інфраструктура як сервіс (Infrastructure as a Service / IaaS) - надається як можливість використання хмарної інфраструктури для самостійного управління ресурсами обробки, зберігання, мережами та іншими фундаментальними обчислювальними ресурсами, наприклад, споживач може встановлювати і запускати довільне програмне забезпечення, яке може включати в себе операційні системи, платформне і прикладне програмне забезпечення. Споживач може контролювати операційні системи, віртуальні системи зберігання даних і встановлені програми, а також володіти обмеженим контролем за набором доступних мережевих сервісів (наприклад,

фаєрволом, DNS). Контроль і управління основною фізичною і віртуальною інфраструктурою хмари, в тому числі мережі, серверів, типів використовуваних операційних систем, систем зберігання здійснюється хмарним провайдером. До цього типу можна віднести AWS EC2. Хмарні технологію значно спрощують життя людей, причому, як простих користувачів, так і розробників різного роду програмного забезпечення. Серед безлічі позитивних сторін хмарних технологій можна виділити:

а) Доступність. Доступ до інформації, що зберігається на хмарі, може отримати кожен, хто має комп'ютер, планшет, будь-який мобільний пристрій, підключений до мережі інтернет. З цього випливає наступна перевага. При цьому найчастіше немає необхідності в установці якогось специфічного програмного забезпечення.

б) Мобільність. Користувач не має постійної прихильності до одного робочого місця. З будь-якої точки світу менеджери можуть отримувати звітність, а керівники - стежити за виробництвом.

в) Економічність. Одним з важливих переваг називають зменшену витратність. Користувачеві не треба купувати дорогі, великі за обчислювальної потужності комп'ютери та комплектуючі, ПЗ, а також він звільняється від необхідності наймати фахівця з обслуговування локальних IT-технологій.

г) Орендної. Користувач отримує необхідний пакет послуг тільки в той момент, коли він йому потрібен, і платить, власне, тільки за кількість придбаних функцій.

д) Гнучкість. Всі необхідні ресурси надаються провайдером автоматично.

є) Висока технологічність. Великі обчислювальні потужності, які надаються в розпорядження користувача, які можна використовувати для зберігання, аналізу і обробки даних.

ж) Надійність. Деякі експерти стверджують, що надійність, яка забезпечує сучасні хмарні обчислення, набагато вище, ніж надійність локальних ресурсів, аргументуючи це тим, що мало підприємств можуть собі дозволити придбати і містити повноцінний дата-центр.

1.1 Хмарні технології Google

Google надає багато різних хмарних технологій, ми зупинимося лише на деяких [3].

– Gmail (SaaS) – поштовий агент;
– Google Calendar (SaaS) – календар;

– Google Drive (SaaS) – файлове сховище;
– Google Docs (SaaS) – аналог MS Word, LibreOffice Writer;
– Google Sheets (SaaS) – аналог MS Excel, LibreOffice Calc;
– Google Forms (SaaS) – створення кастомних опитувань.

З даним набором технологій найбільш часто стикаються пересічні користувачі, до речі можна помітити, що моделлю обслуговування цих технологій є програмне забезпечення (SaaS). Далі ми розглянемо інші хмарні технології від Google, з якими найчастіше зустрічаються розробники програмного забезпечення.

– Compute Engine (IaaS) - віртуальні машини на інфраструктурі Google;
– App Engine - платформа для створення масштабованих веб-додатків і мобільних API;
– Container Engine - платформа для запуску докер-контейнерів під Kubernetes;
– Container Registry - приватна сховище для докер образів;
– Cloud Functions - платформа для створення подієво-орієнтованих мікросервісів;
– Cloud Storage - хмарне сховище;
– Cloud SQL - реляційна база даних від Google..

Ще не можна залишити без уваги певний розділ Google сервісів, наданих безпосередньо для освіти Google Apps Education Edition. Серед них є такі.

– Gmail;
– Google ClassRoom;
– Google Drive;
– Google Docs;
– Google Sheets;
– Google Calendar;
– Google Forms;
– Google Slides;
– Hangouts.

Можна помітити, що багато з даних сервісів аналогічні тим, що були згадані раніше. Але є одна невелика відмінність - повна відсутність реклами і більш зручне використання групою людей. Також варто трохи зупинитися на деяких з цих сервісів, оскільки ми не обговорили їх раніше:

– Google ClassRoom - свого роду навчальне середовище для організації навчального процесу. Допомагає значно заощадити час викладача і студентів;

– Google Forms - сервіс, який дозволяє створювати форми (опитувальники). Може бути дуже зручним, в разі, коли необхідно швидко прийняти якесь рішення і при цьому врахувати думку більшості, але відсутня можливість зібрати всіх студентів;

– Google Slides - аналог MS PowerPoint;

– Hangouts - відео-чат з можливістю створення конференцій. Може бути дуже зручний для віддаленого проведення лекцій та вебінарів.

Варто ще помітити, що календар можна дуже зручно організувати як розклад, особливо якщо врахувати, що він має хорошу систему оповіщень. Позитивні сторони:

– мінімальні вимоги до апаратного забезпечення (обов'язкове умовою - наявність доступу в Інтернет);

– хмарні технології не вимагають витрат на придбання і обслуговування спеціального програмного забезпечення (доступ до додатків можна отримати через вікно веб браузера);

– Google Apps підтримують всі операційні системи і клієнтські програми, які використовуються студентами та навчальними закладами;

– робота з документами можлива за допомогою будь-якого мобільного пристрою, що підтримує роботу в Інтернеті;

– всі інструменти Google Apps Education Edition безкоштовні.

Підводячи підсумки можна помітити, що Google надає величезну кількість самих різних хмарних технологій для самих різних людей. Деякі з них можуть бути актуальні для людей, які займаються наукою, деякі - для програмістів, для рядових користувачів і навіть для груп людей, орієнтовані на освітній процес. На жаль Амазон не має такого величезного розмаїття сервісів для різного типу користувачів, але він має ще більшу кількість сервісів орієнтовану на розробників програмного забезпечення. Багато з них будуть розглянуті в наступному розділі.

1.2 Хмарні технології Amazon

Відразу варто відзначити, що більшість сервісів Amazon навряд чи знадобляться рядовому користувачеві, частіше з ними стикаються розробники програмного

забезпечення. Amazon Web Services пропонує широкий спектр глобальних хмарних продуктів, включаючи обчислювальні додатки, сховища даних, бази даних, засоби аналітики, мережі, мобільні технології, інструменти для розробників, інструменти управління, технології «Інтернету речей», кошти забезпечення безпеки і корпоративні програми [4] . Ці сервіси допомагають різним організаціям розвиватися швидше, знижувати витрати в сфері IT і забезпечувати масштабування. Найбільші корпорації і стрімко зростаючі стартапи довіряють AWS підтримку роботи широкого спектру робочих навантажень, включаючи мобільні та інтернет-додатки, розробку ігор, обробку та зберігання даних, сховища, архівацію і багато іншого.

Дуже зручно, що управління AWS здійснюється як за допомогою веб інтерфейсу (AWS console), так і за допомогою Command Line Tools. В консолі зібрані всі сервіси AWS, але функціональність налаштування кілька обрізана. У командному рядку ж можна більш гнучко налаштувати той чи інший сервіс, також доступні закриті в консолі функції. Далі більш детально будуть описані найпопулярніші сервіси Amazon [5].

EC2 - це хмарний сервіс, що надає віртуальні сервера (Amazon EC2 Instance), 2 видів сховищ даних, а також балансувальник навантаження (Load Balancer).

EC2 може чимось нагадувати VPS, але це не що інше, як сервіс, що надає VPS в цьому хмарному сервісі, де сервер може легко мігрувати між нодами, а сховище легко може бути розширено до майже безрозмірного. Тому-то в назві і звучить слово Elastic - Еластичний. EC2 дозволяє запускати вже заздалегідь сконфігуровані сервери з попередньо встановленими ОС: Amazon Linux, Red Hat EL, Suse ES, Windows 2008, Oracle EL, etc. Також можливо створювати свої образи (AMI - Amazon Machine Image) і використовувати будь-який Linux. Є можливість налаштувати захист доступу до серверів. EC2 інстанси об'єднуються в групи безпеки (Security Groups) з можливістю обмеження доступу по портам з IP або підмереж. Балансування навантаження і автомасштабування є дуже важливими функціями EC2. Є можливість створити правила при яких стане можливо автоматично збільшити кількість серверів, наприклад, якщо один або декілька серверів не справляються з навантаженням. Контроль за здоров'ям серверів веде ще один сервіс AWS - Amazon Cloud Watch. За допомогою цього сервісу можна створювати різного роду перевірки - checks - за допомогою яких контролюються найважливіші показники роботи ОС. Додавання майже нескінченної кількості дисків з майже нескінченним об'ємом зберігання. EBS (Elastic Block Storage) - це один з типів сховища в EC2. Особливість його така, що диски, створювані за цією технологією незалежні від VPS-ноди і розташовані на спеціальних Storage серверах, на відміну від Instance сховищ, які розташовані

безпосередньо на серверах віртуалізації. Створення миттєвих образів (Snapshot) дозволяє створити зліпок диска і використовувати його в якості ісходника для AMI (Amazon Machine Image), а також для простої резервної копії ОС.

Amazon S3 - це об'єктне сховище, оснащене простим веб-інтерфейсом. Сервіс дозволяє зберігати будь-який обсяг даних та видавати дані через Інтернет, де б ви не знаходилися. Позитивні сторони наступні. Веб-інтерфейс консолі управління сервісом і мобільний додаток роблять роботу з Amazon S3 простою і зручною. S3 також підтримує повний набір API REST і пакетів SDK, що дозволяє без праці інтегрувати сховище з технологіями сторонніх розробників.Стійка інфраструктура Amazon S3 забезпечує надійне зберігання +99,999999999% об'єктів. Резервні копії даних розподіляються між декількома об'єктами і кількома пристроями на кожному об'єкті. Amazon S3 дозволяє самостійно вибрати обсяг збережених даних і метод доступу до них. Більше не потрібно вираховувати обсяг необхідних ресурсів: вільне масштабування в міру необхідності допоможе домогтися найвищої гнучкості. Amazon S3 підтримує передачу даних по протоколу SSL з автоматичним шифруванням по завершенню завантаження. Сервіс AWS Identity and Access Management (IAM) дозволяє налаштувати політику кошиків, щоб керувати дозволами на рівні об'єктів і доступом до даних.

Стандартне сховище Amazon S3 розроблено для забезпечення доступності об'єктів на рівні 99,99% протягом року і регулюється Угодою про рівень обслуговування Amazon S3, яке забезпечує клієнтові необхідну підтримку і гарантії. Також є можливість вибрати певний регіон AWS, щоб оптимізувати час затримки, знизити витрати або привести систему у відповідність з нормативними вимогами. Amazon S3 - це зберігання величезної кількості даних за дуже низькою ціною. Використовуючи політики життєвого циклу, можна налаштувати автоматичне перенесення своїх даних у міру старіння в стандартне сховище нечастого доступу або в Amazon Glacier для ще більшої економії.

Amazon надає різні варіанти міграції хмарних даних, які дозволяють просто і економічно переміщати великі обсяги даних в Amazon S3 і з нього. Для імпорту даних в S3 і експорту даних з нього можна використовувати методи підключення на базі фізичних дисків, оптимізовані для роботи в мережі, а також штекера від сторонніх розробників. Amazon S3 глибоко інтегрований з іншими сервісами AWS, що спрощує розробку комплексних рішень з використанням відразу декількох сервісів AWS. Підтримується інтеграція з Amazon CloudFront, Amazon CloudWatch, Amazon Kinesis, Amazon RDS, Amazon Glacier, Amazon EBS, Amazon DynamoDB, Amazon Redshift, Amazon Route 53, Amazon EMR, Amazon VPC, Amazon KMS і AWS Lambda.

Можливості S3 Storage Management дозволяють підійти до питань оптимізації зберігання, забезпечення безпеки даних і ефективності управління з позиції оптимальної роботи з даними. Ці можливості корпоративного рівня надають клієнтам дані о збережених даних і дозволяють управляти сховищем за допомогою індивідуально налаштованих метаданих.

Amazon Relational Database Service (Amazon RDS) дозволяє легко налаштовувати, використовувати і масштабувати реляційні бази даних в хмарі. Цей сервіс надає економічні та масштабовані ресурси і одночасно керує виконанням трудомістких завдань адміністрування баз даних. Завдяки цьому ви можете зосередитися на своїх додатках і веденні бізнесу. Amazon RDS підтримує шість популярних ядер баз даних: Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle і Microsoft SQL Server.

Позитивні сторони наступні:

а) Сервіс Amazon RDS дозволяє легко перейти від концепції проекту до його розгортання. Скористайтеся Консоллю управління AWS, інтерфейсом командного рядка AWS RDS або простими викликами API, щоб всього за кілька хвилин отримати доступ до можливостей реляційної бази даних, повністю готовою до роботи. При цьому не потрібно турбуватися про виділення інфраструктури або установці і обслуговуванні програмного забезпечення для баз даних.

б) Користувач має можливість масштабувати обчислювальні ресурси і ресурси зберігання своєї бази даних за допомогою декількох клацань миші або викликів API, найчастіше без простоїв. Багато типів двигунів Amazon RDS дозволяють запускати одну або кілька реплік читання для розвантаження основного інстанси баз даних від трафіку читання.

в) Amazon RDS працює на тій же надійнії інфраструктуриі що і інші сервіси Amazon Web Services. При створенні інстансів БД в декількох зонах доступності Amazon RDS синхронно репліцює дані на резервні інстанси в іншій зоні доступності. Сервіс Amazon RDS має різні можливості, які підвищують надійність критично важливих робочих баз даних, серед яких автоматичне резервне копіювання, знімки стану БД і автоматична заміна хоста.

г) Сервіс Amazon RDS дозволяє легко перейти від концепції проекту до його розгортання. Скористайтеся Консоллю управління AWS, інтерфейсом командного рядка AWS RDS або простими викликами API, щоб всього за кілька хвилин отримати доступ до можливостей реляційної бази даних, повністю готовою до роботи. При цьому не потрібно

турбуватися про виділення інфраструктури або установці і обслуговуванні програмного забезпечення для баз даних.

д) Сервіс Amazon RDS дозволяє легко керувати мережевим доступом до бази даних. Amazon RDS також дозволяє запускати інстанси баз даних в хмарі Amazon Virtual Private Cloud (Amazon VPC), щоб ізолювати інстанси баз даних і підключатися до існуючої IT-інфраструктури з використанням VPN зі стандартним шифруванням IPsec. Багато типів двигунів Amazon RDS пропонують шифрування збережених і переданих даних.

є) Користувач сплачує за дуже низькими тарифами і тільки за ресурси, які реально використовуєте. Крім того, він отримує перевагу від оплати на вимогу, без передоплати або довгострокових контрактів, або навіть нижчих погодинних тарифів при використанні цін на зарезервовані інстанси.

Route53 - це високодоступних і масштабований хмарний веб-сервіс системи доменних імен (DNS). Використовується як дуже надійний і ефективний метод перенаправлення кінцевих користувачів до інтернет-додатків, переводячи доменні імена (наприклад, www.example.com) в формат цифрових IP-адрес (наприклад, 192.0.2.1), зрозумілих для комп'ютерів. Amazon Route 53 також повністю сумісний з протоколом IPv6. Сервіс Amazon Route 53 направляє запити користувачів до інфраструктури AWS, наприклад до інстанси Amazon EC2, балансувальник навантаження Elastic Load Balancing або кошиках Amazon S3. Крім того, він може використовуватися для перенаправлення користувачів в інфраструктуру за межами AWS. Amazon Route 53 можна використовувати як для організації підключень тільки за «здоровим» адресами (з використанням перевірок DNS), так і для незалежного моніторингу стану програми і його кінцевих точок. З сервісом Amazon Route 53 Traffic Flow це зручний спосіб упорядкування глобальним трафіком за допомогою різних типів маршрутизації (таких як маршрутизація на базі затримки, DNS з урахуванням географічного положення та циклічний зважений алгоритм), які можна поєднувати з можливістю перенесення сервісу DNS, створюючи в результаті відмовостійкі архітектури з низькою затримкою. Використовуючи нескладний візуальний редактор Amazon Route 53 Traffic Flow, це зручний спосіб упорядкування маршрутизацією кінцевих користувачів до кінцевих точок ваших додатків як в рамках одного регіону AWS, так і при розподілі трафіку по всьому світу. Крім того, в сервісі Amazon Route 53 можна зареєструвати доменне ім'я: при покупці доменів (наприклад, example.com) і управлінні ними Amazon Route 53 автоматично налаштує для них параметри DNS.

Amazon Simple Queue Service (SQS) - це швидкий, надійний, що масштабується, і повністю керований сервіс черг повідомлень. Amazon SQS дозволяє легко і економічно розділити компоненти хмарного додатка. Amazon SQS можна використовувати для передачі будь-якого обсягу даних без втрати повідомлень і необхідності постійного доступу до інших сервісів. Amazon SQS пропонує, як стандартні черги з високою пропускною здатністю і гарантованої обробкою повідомлень за принципом «хоча б один раз», так і черги FIFO з доставкою повідомлень за алгоритмом FIFO («першим отримано - першим відправлено») і строго одноразової обробкою. Позитивні сторони:

а) Amazon SQS працює на базі високодоступних ЦОД Amazon, тому черги постійно доступні для ваших додатків. З метою запобігання втрат повідомлень або відсутності доступу до них всі повідомлення зберігаються в кількох копіях на серверах декількох ЦОД.

б) Розробники можуть почати роботу з Amazon SQS, використовуючи всього три API: SendMessage, ReceiveMessage і DeleteMessage. Для розширення функціоналу доступні додаткові API.

в) Можливість вибору черзі в залежності від потреб програми. Стандартні черзі забезпечують максимальну пропускну здатність, «найкраще з можливого» впорядкування і доставку повідомлень за принципом «хоча б один раз» .Очереді FIFO забезпечують суворе впорядкування і строго одноразову обробку повідомлень, при цьому пропускна здатність черзі обмежена.

г) Amazon SQS забезпечує динамічне масштабування відповідно до потреб додатки, тому попереднє виділення ресурсів не потрібно. Кількість використовуваних черг і повідомлень не обмежена, крім того, стандартні черзі забезпечують практично необмежену пропускну здатність.

д) Сервіс пропонує механізми аутентифікації, що дозволяють надійно захистити повідомлення, що зберігаються в Amazon SQS, від несанкціонованого доступу.

є) Amazon SQS не передбачає авансових або фіксованих платежів. Передбачена лише невелика плата за кожен запит API. При передачі даних з іншого регіону AWS або в інший регіон AWS стягується додаткова плата.

Amazon Simple Email Service (SES) - це сервіс надає інфраструктуру по відправці великого числа поштової кореспонденції. SES дозволяє надсилати листи через API - безпосередньо з програми. Існують десятки бібліотек, плагінів, що дають можливість надсилати листи обходячи SMTP методи. Для тих додатків, які не можуть бути інтегровані з SES через API - існує опція включення SMTP сервера з авторизацією по

зв'язці логін-пароль. Не варто забувати, що, як і інші сервіси Amazon має величезну надійність і дуже легко масштабується.

Amazon CloudWatch - це сервіс моніторингу хмарних ресурсів AWS і додатків. Amazon CloudWatch можна використовувати для збору і відстеження метрик, накопичення та аналізу файлів журналів, створення попереджень, а також автоматичного реагування на зміни ресурсів AWS. Amazon CloudWatch може використовуватися для моніторингу наступних ресурсів AWS: інстанси Amazon EC2, таблиць Amazon DynamoDB, інстанси Amazon RDS DB, а також для моніторингу призначених для користувача метрик додатків і сервісів і будь-яких логів ваших додатків. Можна використовувати Amazon CloudWatch для отримання зведеної інформації про систему, що включає в себе інформацію про використовувані ресурси, продуктивності додатків і загальний стан системи. Ці дані застосовуються для оперативного реагування та забезпечення стабільної роботи додатків.

AWS Identity and Access Management (IAM) - сервіс дозволяє безпечно керувати доступом користувачів до сервісів та ресурсів AWS. Використовуючи IAM, можна створювати користувачів і групи AWS і управляти ними, а також використовувати дозволу, щоб надати або заборонити їм доступ до ресурсів AWS.

Були розглянуті далеко не всі сервіси від Amazon, але цього достатньо, щоб зробити висновок, що серед них достатня кількість технологій, які можуть у величезній мірі полегшити життя розробникам програмного забезпечення завдяки своїй багатофункціональності та гнучкості.

1.3 Технологія OpenStack

OpenStack - комплекс проектів вільного програмного забезпечення, який може бути використаний для створення інфраструктурних хмарних сервісів і хмарних сховищ, при тому як публічних, так і приватних. OpenStack складається з ряду окремих компонентів (приведена основна частина) [6].

- Compute (Nova)
- Networking (Neutron / Quantum)
- Identity Management (Keystone)
- Object Storage (Swift)
- Block Storage (Cinder)

- Image Service (Glance)
- User Interface Dashboard (Horizon)

Модуль OpenStack Compute (Nova) управляє "фабрикою" хмарних обчислень (це базовий компонент інфраструктурних сервісів). Модуль OpenStack Compute, написаний на мові Python, утворює рівень абстрагування для віртуалізації ресурсів масових серверів (таких як процесори, пам'ять, мережеві адаптери і жорсткі диски) і підтримує відповідні функції для підвищення коефіцієнта використання і для автоматизації.

OpenStack Compute забезпечує активне управління віртуальними машинами із застосуванням таких функцій, як запуск, зміна розмірів, припинення, зупинення та перезавантаження, за допомогою інтеграції з набором підтримування гіпервізора. Крім того, є механізм для кешування образів віртуальних машин на вузлах Compute з метою прискорення ініціалізації цих машин. В процесі виконання цих образів підтримується можливість програмованого збереження файлів і управління ними через API-інтерфейс.

Модуль Networking (Neutron), раніше носив ім'я Quantum, забезпечує управління локальними мережами з підтримкою віртуальних мереж (VLAN), протоколу DHCP і протоколу IP v6. Користувачі можуть визначати мережі, підмережі та маршрутизатори з метою конфігурації їх внутрішньої топології, а потім призначати цим мережам IP-адреси і віртуальні мережі. Механізм плаваючих IP-адресів дозволяє користувачам призначати (і перепризначувати) віртуальним машинам фіксовані зовнішні IP-адреси.

Модуль OpenStack Identity Management (Keystone) управляє каталогом користувачів, а також каталогом сервісів OpenStack, до яких ці користувачі можуть звертатися. Його мета полягає в підтримці механізму централізованої аутентифікації для всіх компонентів OpenStack. Замість безпосередньої підтримки аутентифікації модуль Keystone здатний інтегруватися з різними іншими сервісами каталогів, включаючи Pluggable Authentication Module, Lightweight Directory Access Protocol (LDAP), OAuth. За допомогою відповідних плагінів забезпечується підтримка декількох форм аутентифікації - від простих засобів входу з використанням логіна / пароля до складних багатофакторних систем.

OpenStack Identity дозволяє адміністраторам конфігурувати централізовані політики, що застосовуються до користувачів і до систем. Адміністратори можуть створювати проекти і користувачів, приписувати їх до адміністративних доменах, визначати повноваження для доступу до ресурсів на основі ролей, здійснювати інтеграцію з іншими каталогами, такими як LDAP. Каталог містить список всіх розгорнутих сервісів в єдиному реєстрі. Користувачі і інструменти можуть витягувати список доступних їм

сервісів за допомогою програмних запитів або за допомогою панелі інструментів, яку також можна використовувати для створення ресурсів і для приписування їх до свого облікового запису.

Модуль OpenStack Object Storage (Swift) заснований на продукті Rackspace Cloud Files і являє собою резервування системи зберігання, яка ідеально підходить для горизонтального масштабування ресурсів зберігання. OpenStack гарантує реплікацію даних і їх розподіл між пристроями в своєму пулі, завдяки чому користувачі можуть використовувати масові жорсткі диски і сервери замість більш дорогого обладнання. У разі відмови будь-якого компонента система OpenStack здатна заповнити контент з інших активних систем і перенести його на нові елементи кластера. Ця архітектура підтримує горизонтальне масштабування завдяки легкості розширення кластерів зберігання додатковими серверами в міру необхідності.

Swift - це розподілена система зберігання, орієнтована переважно на статичні дані, такі як образи віртуальних машин, резервні копії і архіви. Програмне забезпечення Swift записує файли і інші об'єкти на набір дискових накопичувачів, який може бути розподілений між декількома серверами в одному або декількох центрах обробки даних, що гарантує можливість реплікації даних і їх цілісність в масштабі кластера.

Модуль OpenStack Block Storage (Cinder) керує сховищем блочного рівня, яке використовують екземпляри Compute. Блочне сховище добре підходить для сценаріїв зі строгими вимогами до продуктивності (бази даних і файлові системи).

Спільно з модулем Cinder найчастіше використовується сховище на основі Linux-сервера, проте є плагіни і для інших платформ, включаючи Ceph, NetApp, Nexenta і SolidFire. Користувачі хмарних ресурсів можуть управляти своїми ресурсами зберігання за допомогою інструментальної панелі. Система надає інтерфейси для створення блокових пристроїв, для приєднання їх до серверів і для від'єднання їх від серверів. Крім того, підтримується можливість створення резервних копій томів Cinder за допомогою механізму моментальних знімків.

Модуль OpenStack Image Service (Glance) забезпечує підтримку образів віртуальних машин, зокрема, системних дисків, призначених для використання при запуску примірників віртуальних машин. Крім сервісів виявлення, реєстрації та активації даний модуль підтримує можливості створення моментальних знімків і резервних копій.

Glance-образи можуть служити шаблонами для швидкого та узгодженого розгортання нових серверів. Як серверного API застосовується RESTful-інтерфейс (Representational State Transfer), за допомогою якого користувачі можуть переглядати і

вибирати на віртуальному диску образи, приписані до ширший набір внутрішніх сховищ (включаючи OpenStack Object Storage).

Користувачі можуть надавати цьому сервісу приватні та публічні образи в різних форматах, в тому числі в наступних: VHD (Microsoft (Hyper-V), VDI (VirtualBox), VMDK (VMware), qcow2 (Qemu / Kernel-based Virtual Machine), Open Virtualization Format. Є функції для реєстрації нових образів віртуальних дисків, для запиту інформації про публічно доступних образах дисків і для потокової доставки образів віртуальних дисків.

Типова реалізація OpenStack містить більшість або навіть всі перераховані проекти.

Три елемента архітектури взаємодіють з усіма компонентами системи. Horizon - це графічний інтерфейс користувача, який дозволяє адміністраторам з максимальною легкістю управляти всіма проектами. Keystone здійснює управління авторизованими користувачами, а Neutron дозволяє визначати мережі, які забезпечують зв'язок між компонентами.

Nova - це, поза всяким сумнівом, центральний елемент OpenStack. Nova здійснює оркестровку робочих навантажень. Compute-екземплярів зазвичай потрібно той або інший різновид персистентного сховища, яке може бути або блоковим (Cinder), або об'єктним (Swift). Крім того, Nova потрібно образ для запуску екземпляра. Glance обробляє цей запит і в якості опції може використовувати Swift в якості внутрішнього сховища.

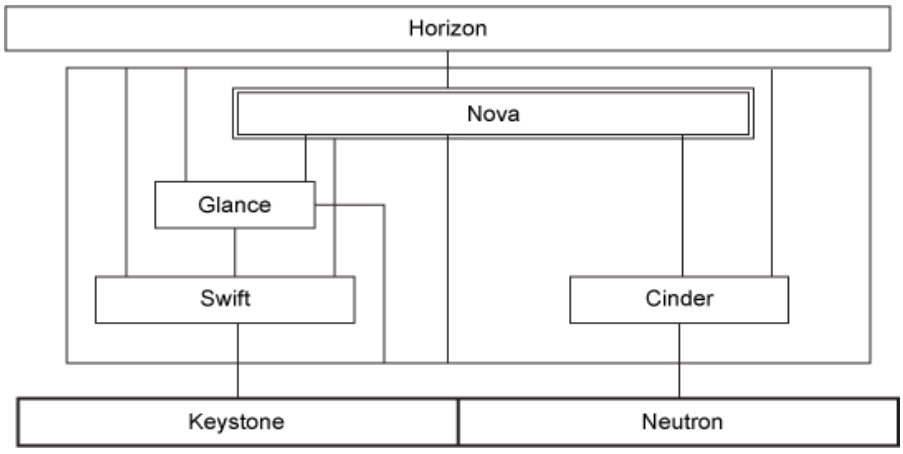


Рисунок 1.1 − Архітектура OpenStack

Архітектура OpenStack заохочує максимальну незалежність кожного проекту. Це дозволяє користувачам розгортати лише необхідну їм підмножину функціональності та інтегрувати цю підмножину з іншими системами і технологіями, які пропонують подібні або доповнюють функції. Проте ця незалежність не повинна маскувати той факт, що для

безперешкодної діяльності загальнофункіональній приватній хмарі , швидше за все, буде потрібно практично вся функціональність, при цьому всі елементи повинні бути тісно інтегровані.

1.4 Постановка завдання дослідження

Задача контролю діяльності студента була актуальною абсолютно завжди. А з огляду на те, що на даний момент багато студентів починають працювати ще, навчаючись в університеті, їх комунікація з викладачем сильно порушується, що в подальшому може сильно позначитися на якості курсових і дипломних робіт. Саме в зв'язку з цим було прийнято рішення розробити систему, що дозволяє проводити розробку курсових і дипломних проектів віддалено, при цьому викладач завжди буде мати можливість подивитися, що саме робить студент і на якій стадії розробки він знаходитися.

Для досягнення поставленої задачі необхідно коректно вибрати хмарне сховище, оскільки надавати доступ до комп'ютера студента є не дуже простим і коректним завданням, а постійний обмін файлами по електронній пошті, або якимось іншим чином, не дуже зручний. Виходячи з цього, найбільш відповідним рішення може бути Google Drive, оскільки він уміє не тільки зберігати документи, але також представляє дуже зручний інтерфейс для взаємодії з ними і їх редагування. Взагалі можна досить довго перераховувати різні позитивні моменти GCS, а також сервіси, які він надає, розглянемо лише такі:

- Google Drive.
- Google Docs.
- Google Translate.
- Google Sheets.
- Google Calendar.

Варто зауважити, що я не згадував сервіси, з яким рідко стикаються пересічні користувачі, які не є розробниками клієнт-серверних і мобільних додатків.

Ще одним позитивним моментом є те, що Google надає технологію OAuth і наявність у кожного нашого студента власної електронної пошти наданою університетом в єдиному домені snu.edu.ua.

В якості хостингу найкращим рішенням є AWS (Amazon Web Services). Він також надає величезну кількість різних хмарних сервісів, наприклад, EC2 (виділений сервер) або ж S3 (хмарне сховище). Деякі з них також будуть розглянуті в подальшому.

Об'єкт дослідження - проектна діяльність студентів в рамках роботи в LMS.

Предмет дослідження - моделі, технології та архітектурні рішення, використовувані при розробці клієнт-серверних освітніх систем з використанням хмарних технологій.

Мета дослідження - покращення організації та контролю самостійної роботи студентів завдяки розробці системи управління проектною діяльністю студентів з використанням хмарних технологій.

Для досягнення поставленої мети необхідно вирішити такі завдання. \

- розробити загальну структуру системи, описати систему у вигляді ряду модулів, описати взаємодія між ними;
- розробити принципи взаємодії зі сторонніми сервісами, провести дослідження інших рішень;
- розробити систему аутентифікації, пов'язану з доменом snu.edu.ua;
- сформулювати вимоги до системи засновані на користувальницьких історіях;
- розробити архітектуру системи, а також архітектуру бази даних на основі MongoDB;
- реалізувати систему і провести інтеграцію зі сторонніми сервісами.

2 СИСТЕМА “КАБІНЕТ КУРСОВОГО ПРОЕКТУВАННЯ”

2.1 Користувальницькі історії

Користувальницькі історії - спосіб опису вимог до розроблюваної системі, сформульованих як одне або більше пропозицій на повсякденному або діловою мовою користувача. Призначені для користувача історії використовуються гнучкими методологіями розробки програмного забезпечення для специфікації вимог.

В системі присутні три основні суб'єкти: студент, викладач і адміністратор. Від цих ролей і будуть формуватися користувальницькі історії.

Користувальницькі історії адміністратора розглянуті нижче.

– як адміністратор я хотів би мати зручну панель адміністрування;

– як адміністратор я хотів би мати завантажувати інформацію, отриману про суб'єктів системи використовуючи CSV або JSON;

– як адміністратор я хотів би мати можливість редагувати інформацію про суб'єктів в разі, якщо була допущена помилка;

– як адміністратор я хотів би мати можливість додати в систему ще одного суб'єкта (адміністратора, студента або викладача).

Користувальницькі історії викладача розглянуті нижче:

– як викладач я хотів би мати можливість створення тем для курсових (дипломних) проектів;

– як викладач я хотів би надати студенту можливість самому придумати тему курсової (дипломної) роботи;

– як викладач я хотів би мати можливість схвалити або спростувати тему запропоновану студентом;

– як викладач я хотів би мати можливість контролювати роботу студентів;

– як викладач я хотів би мати можливість залишати студентам коментарі до їхньої роботи;

– як викладач я хотів би мати доступ до напрацювань студентів;

– як викладач я хотів би мати можливість переглядати, історію змін напрацювань студентів;

– як викладач я хотів би мати можливість створити голосування для всієї групи студентів;

– як викладач я хотів би бути впевненим, що студенти отримали сповіщення щодо створеного голосування;

– як викладач я хотів би мати можливість розіслати оповіщення всієї групі студентів;

– як викладач я не хотів би постійно відвідувати університет з метою консультації студентів.

Користувальницькі історії студента розглянуті нижче:

– як студент я хотів би мати вибрати тему курсового (дипломного) проекту не відвідуючи університет;

– як студент я хотів би мати можливість самостійно запропонувати тему курсової (дипломної) роботи;

– як студент я хотів би мати особистий навчальний простір, до якого викладач також має доступ;

– як студент я не хотів би встановлювати деяке програмне забезпечення (MS Word, MS PowerPoint, etc.) для виконання курсової (дипломної) роботи.

2.2 Бізнес-процеси в системі

Використання системи можна розбити на 4 кроки, кожен з яких може містити в собі ще певні (незалежні) підпроцеси. Під використанням я маю на увазі повний шлях від створення групи студентів і прив'язки до них викладача до збереження в архів готового курсового (дипломного) проекту.

Рівень 1. Підготовка до курсового проектування. На даному етапі відбувається синхронізація бази даних університету оскільки нам необхідно мати актуальні дані по студентському і викладацькому складу, а також за складом груп. Але останнім вже не так важливо, оскільки формування груп і прив'язка до них викладачів може проводитися вручну. Після формування груп і прив'язка до них викладача. викладач приступає до формування тем курсового проекту, а студенти вибирають відповідні їм теми.

Рівень 2. Виконання курсового проекту. На даному етапі студенти приступають до виконання курсового проекту, при цьому варто зауважити, що необхідне постійне підтримання контакту між викладачем і студент, для отримання позитивних результатів. Не варто забувати, що деякі студенти можуть не мати доступу до особистого кабінету і

виконувати роботу локально на своєму комп'ютері. В такому випадку є можливість завантажити вже готову записку, щоб викладач мав можливість ознайомитися з нею.

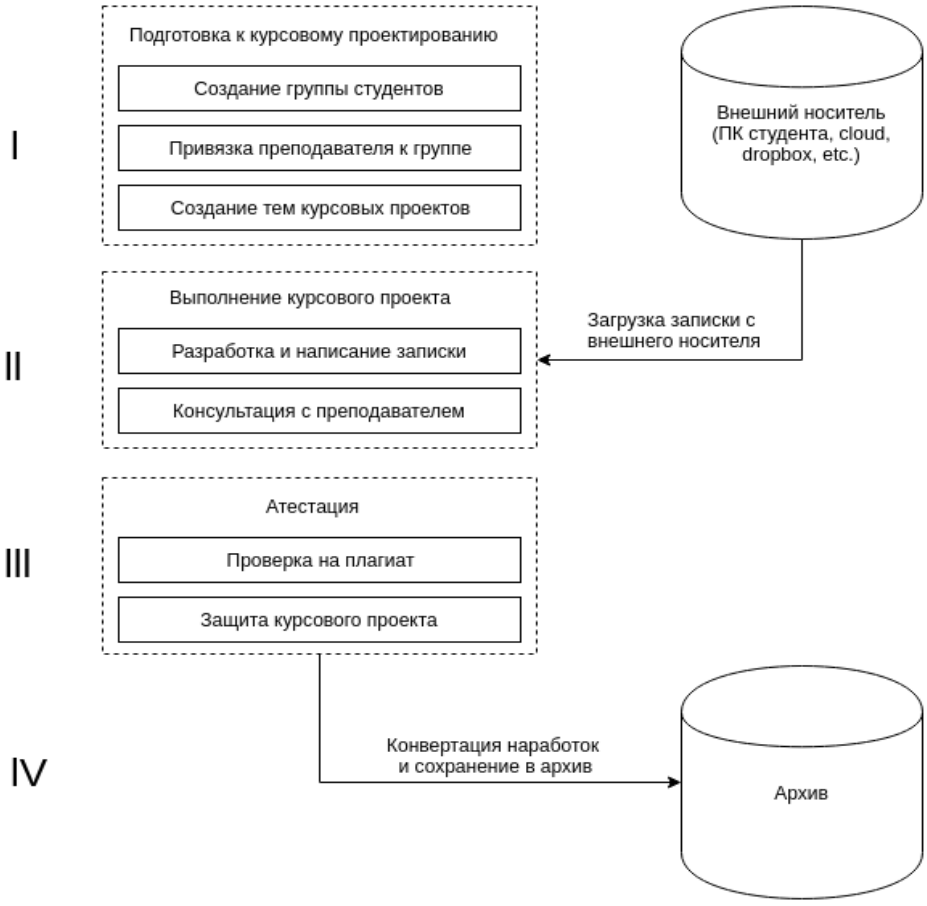


Рисунок 2.1 - Бізнес-процеси в системі

Рівень 3. Атестація. Атестація включає в себе перевірку на плагіат і захист курсового проекту. Це відбувається вже з найменшим взаємодією з системою. Але варто зауважити, що процес захисту може також проходити з використанням хмарних технологій, наприклад, за допомогою Hangouts. За перевірку на плагіат відповідає сторонній сервіс, який взагалі може не взаємодіяти з системою безпосередньо.

Рівень 4. Архівація. Під архівацією мається на увазі процес збереження напрацювань в архів університету, який також є стороннім ресурсом, але оригінал буде продовжувати зберігатися в хмарному сховищі Google. Також перед процесом збереження може відбуватися конвертація пояснювальної записки з метою її збереження в коректному форматі.

Система надає досить великі можливості як студентам, так і викладачам:. Можливості студентів:

– можливість демонструвати свій курсовий проект викладачеві віддалено, що дасть можливість людям, віддаленим у просторі і часі від аудиторій, отримувати відгуки від викладачів про свій проект, що не приїжджають в університет;

– можливість отримати тему свого курсового проекту віддалено, можна буде пропонувати свої теми, або ж довіриться викладачеві і після обопільної згоди тема закріпиться за студентом і її можна буде подивитися онлайн в будь-який момент часу, що дозволить уникнути спірних моментів;

– загальний робочий простір з іншими студентами в разі якщо ми маємо загальний курсовий проект. Це просто необхідно так як комплексний курсовий проект можуть виконувати студенти, об'єднані в групи. Тепер буде видно, хто з групи робить роботу, а хто просто відпочиває, так як всі маєтки в проекті будуть входити до системи і їх можна буде переглянути викладачеві, який зможе використовувати отриману інформацію для більш точно оцінювання студента;

– шаблон пояснювальної записки з зумовленими гостами. Тепер до викладача буде менше питань по оформленню, що дасть йому час займатися більш корисною роботою, ніж кожному студенту з групи пояснювати, як оформляти курсової;

– календарний план курсового проекту. Велике завдання завжди краще розбивати на більш дрібні, це буде мотивувати студентів зробити курсової нема за останній тиждень, забувши про сні, а невеликими порціями протягом усього семестру;

– доступ до матеріалів необхідним для виконання проекту наданих викладачем;

– повідомлення про проведення консультацій, а також вибір часу їх проведення на основі голосування студентів;

– можливість розміщувати свої напрацювання, щоб вони були доступні викладачеві;

– можливість листуватися з викладачем в самому додатку. Зайшовши в свій особистий кабінет можна безпосередньо буде написати викладачеві і отримати своєчасну відповідь, що заощадить купу часу;

– контроль версій, який дозволить повертатися до старих більш правильним версіями проекту і бачити, хто і коли вносив зміни в проект;

– можливість переглядати проекти інших студентів. Це дозволить студентам переймати хороший стиль оформлення, беручи приклад з лідерів групи, адже в групі завжди є ті, хто знає трохи більше, або краще розбирається в форматування, або в чомусь

іншому. Також це створить атмосферу здорового змагання, що підвищить стимул виконувати проекти швидше;

– можливість листуватися з іншими студентами. Якщо, щось не виходить завжди можна буде попросити допомоги у своїх одногрупників.

Можливості викладача:

– можливість переглядати і редагувати роботу студента. Це дозволить викладачеві своєчасно оцінювати роботу студентів і допомагати їм в оформленні;

– можливість проаналізувати роботу студента на наявність плагіату. Це функціональність дозволить виявити недобросовісних студентів і змусити їх працювати;

– можливість роздати студентам теми курсових проектів віддалено. Це дасть можливість відповідальним викладачам не займатися пошуком студентів, які не прийшли взяти тему, а заощадивши свій час, зробити це віддалено, при цьому витративши всього пару хвилин;

– можливість переглянути статистику активності студента. Для викладача це буде хорошим показником того, що студент витрачав багато часу на виконання курсового проекту, або ж взагалі майже нічого не робив, що збільшить ступінь точності оцінювання роботи;

– можливість повідомити студентам про майбутню (ймовірно позапланової) консультації. Коли з міністерства приходить нова інформація викладачеві важливо вчасно донести її до студентів, в цьому і допоможе ця функціональність. А також вона дасть можливість обирати зручний час для проведення консультації за допомогою голосування студентів;

– можливість переглянути готовність курсового проекту студента на поточний момент. Що дасть можливість квапити неорганізованих студентів і добре оцінювати тих, хто виробляє виконання в строки;

– можливість переглядати дату останнього візиту студента. Тепер студент не зможе збрехати, що він недавно заходив і все робив;

– можливість спілкуватися з кожним студентом особисто. Як викладачеві, так і студенту ця функціональність дасть можливість обговорити всі аспекти проектування курсового, задати питання, попросити допомоги, і іноді просто душевно поспілкуватися;

– можливість мати чат з керуємою групою. Щоб ні писати новини кожному студенту, досить буде написати цю новину в загальний чат, що відразу ж дозволить подивитися реакцію студентів на неї;

– електронний журнал. Дуже корисна функціональність, що дасть викладачеві можливість, проставляти ступінь готовності виконання і ставити оцінки за захист;

– статистика виконання курсового проекту курує групи.

2.3 Структурна модель системи

У даній системі можна виділити певні частини, що відповідають за виконання різних завдань.

Модуль адміністрування. Даний модуль є почасти унікальним для кожного суб'єкта. Оскільки кожен з них має свого роду особистий кабінет, який він може адмініструвати. для адміністратора - це безпосередньо адмінка, в якій він може вивантажувати дані про інших суб'єктах системи, отримані зі сторонніх сервісів, наприклад, бази даних університету. Також там може проводитися певне коригування даних суб'єкта (у разі якщо була допущена якась помилка). Основне завдання адміністратора системи в підтримці цілісності даних і їх синхронізації між базою даних університету і базою даних системи. Що стосується викладача, то він має можливість переглядати роботу студентів, які до нього прив'язані, залишати коментарі, також робити розсилки і створювати опитування для всієї групи студентів. Хоча варто врахувати, що спочатку викладач створює теми робіт, які згодом вибирають студенти. Викладач може абсолютно повністю контролювати роботу своїх студентів (але тільки своїх), в той час як студенти не мають доступу до робіт один одного і тим більше до них не має ніякого доступу адміністратор. Модуль адміністрування для студента представляє з себе свого роду навчальний простір, де студент зберігає свої напрацювання щодо проекту, а також може отримувати оповіщення про дедлайни і переглядати коментарі викладача до своєї роботи. Даний кабінет є унікальним для кожного студента, і вони не мають доступу до кабінетів один одного.

Модуль авторизації. З даним модулем взаємодіють абсолютно всі суб'єкти системи і навіть не зареєстровані \ авторизовані користувачі, оскільки модуль блокує їх доступ до практично всієї системи. Також він розмежовує доступ суб'єктів до деякої інформації.

Модуль взаємодії з навчальної середовищем. Даний модуль можна вважати взагалі стороннім сервісом, точніше навіть так і є. Це Google Docs. За коректність доступу відповідає технологія oauth і сам гугл. Це обумовлено тим, що доступ до системи студент може отримати тільки в разі, якщо в базі даних міститься його адреса електронної пошти в

домені snu.edu.ua. Точно також і доступ до свого Google диску він отримує через ту саму пошту. Отже, відповідальність за збереження даних перекладена на Google.

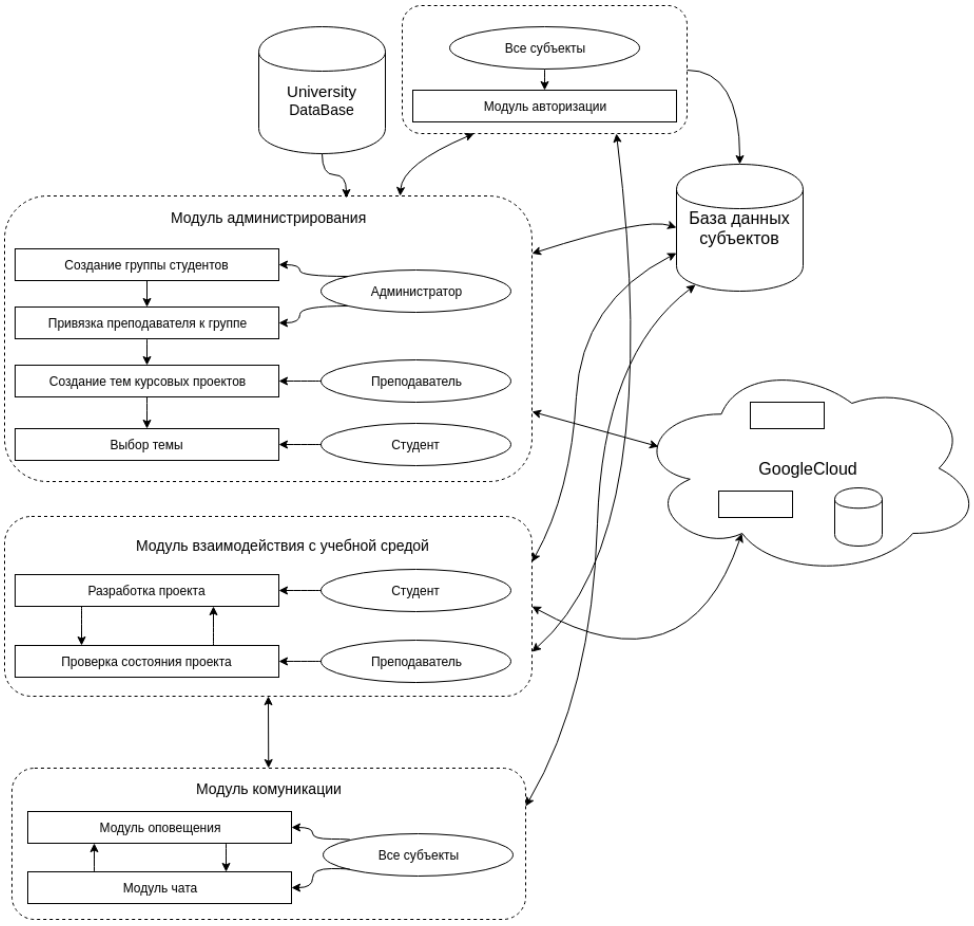


Рисунок 2.2 - Модульна структура системи

Модуль комунікації. Даний модуль відповідає за взаємодію між суб'єктами. Комунікація відбувається шляхом обміну повідомленнями або ж листами. Листи відправляються автоматично, без використання сторонніх поштових клієнтів. Технічна сторона питання буде викладена в наступних розділах.

Варто приділити певну увагу технологіям, які нам надає Google. Як уже згадувалося раніше, ми використовуємо технологію oauth - відкритий протокол аутентифікації. Таким чином відповідальність за аутентифікацію перекладена на Google. Також використовується Google Drive і Google Docs, які є файловим сховищем і редактором документів. Всі маніпуляції з ними відбуваються через API.

2.4 Робота з контентом в системі «Кабінет курсового проектування»

Для того щоб система «Кабінет курсового проектування» могла ефективно працювати, потрібен спосіб для забезпечення актуальності, корисності і організованості інформації, яка присутня в системі. В системі буде присутній функціональні можливості, що забезпечують автоматичну обробку цієї інформації, такі як, «модуль перевірки курсових проектів», автоматична відправка пошти при завантаженні курсового проекту та інші. Але для повного забезпечення впорядкованості та актуальності інформації цього недостатньо. Для цих цілей краще всього підійде спеціально призначена людина, якого називають куратор контенту.

2.4.1 Контент, який бере участь в роботі системи

В системі «Кабінет курсового проектування» (ККП) присутня велика кількість контенту, для того, щоб ефективно з ним працювати, для початку, потрібно визначити види контенту, які будуть присутні в системі.

Види контенту, присутні в системі «Кабінет курсового проектування»:

- Контент присутній в пояснювальній записці до курсового проекту:
    1) зображення;
    2) посилання на використані джерела;
    3) таблиці;
    4) текст;
    5) вирізки з програмного коду.
- Контент безпосередньо присутній в самій системі:
    1) питання і відповіді;
    2) відгуки;
    3) Google Документи;
    4) листування;
    5) календар;
    6) новини;
    7) журнал.

В системі «Кабінет курсового проектування» присутня велика кількість інформації, так званого контенту. У вузі вчитися приблизно 11000 чоловік, і кожен семестр десь 5000 пишуть курсової проект. Так як записка до курсового проекту пишеться не за один день, при організованій роботі, контрольних версій цієї записки буде близько 10, в залежності від того, скільки контрольних точок призначить викладач, провідний проект.

Така велика кількість пояснювальних записок потребує грамотної і націленої обробки. На основі обробки цієї інформації надається актуальна і зручна у використанні інформація.

Незважаючи на те, що в системі будуть присутні функціональні можливості, що забезпечують автоматичну обробку цієї інформації, такі як, «модуль перевірки курсових проектів», автоматична відправка пошти при завантаженні курсового проекту та інші. Система так само потребує ручну обробку цієї інформації. В ролі суб'єкта, що виконує цю роль буде виступати куратор контенту.

У ролі куратора контенту, в окремому випадку, буде виступати викладач, провідний групи, він буде створювати новини, призначати контрольні точки, заповнювати журнал, призначати консультації, давати відгуки про роботу. Все це буде тримати контент в системі в актуальному стані. Також частина обов'язків куратора візьмуть на себе студенти, які будуть завантажувати актуальні версії своїх робіт і залишати коментарі в роботах.

Для системи в цілому добре було б мати людину, яка займається обробкою і актуалізуванням інформації в цілому. Ця людина повинна буде давати права доступу, структурувати пояснювальні записки шляхом розміщення вже закінчених і захищених пояснювальних записок до архіву, а також сортувати проекти за певними тематиками.

Основні діючі ролі, які будуть брати участь в роботі з контентом в системі:

- головний адміністратор (університет);
- адміністратор;
- викладач;
- студент.

Найвпливовішу роль матиме головний адміністратор, роль якого буде брати на себе керівництво деканату. Керівництво деканату буде вирішувати, чи буде працювати і застосуються система «Кабінет курсового проектування» взагалі, а також буде припиняти і знову відновлювати її роботу, виділяти гроші на її обслуговування.

Головний адміністратор буде призначати адміністратора, який може бути однією людиною, або групою осіб. Адміністратор системи буде виконувати такі функції:

- створення користувачів;
- видалення користувачів;
- роздача прав доступу;
- об'єднання користувачів в групи;
- закріплення групи за викладачем;
- підтримка працездатності системи;
- консультування про питання, що виникають при роботі з системою;
- архівування даних;
- оповіщення користувачів про роботу системи;
- обслуговування системи.

Основну роботу з контентом в системі здійснюватимуть студент і викладач.

Для зручної роботи з контентом в системі, першим кроком буде створення персональної навчального середовища. Викладач разом з групою студентів, в ході обговорення, повинні вибрати такі інструменти ПУС, які дозволять зручно взаємодіяти між собою. Один інструмент ПУС з самого початку буде визначено, це буде система «Кабінет курсового проектування». Далі в ході обговорення потрібно вибрати засіб для зручного спілкування, так як деякі можливості системи «Кабінет курсового проектування» будуть обмежені, то інші засоби спілкування будуть більш звичні і зручні, як для студента, так і викладача. Наприклад, в групі ІТП-16дм для швидкого оповіщення студентів використовується соціальна мережа Вконтакті. Для швидкого перенесення лекції, консультації або лабораторної роботи викладач дзвонить старості, до слова телефон - це теж необхідний елемент ПУС, і вона в свою чергу пише повідомлення про зміни в створений груповий діалог ВК. Це є більш зручним засобом спілкування, так як в наш час майже кожен студент проводить багато годин в соціальних мережах, таких як ВК і отримує цю інформацію майже так само швидко, як би йому це сказали особисто.

2.4.2 Організація роботи куратора контенту в системі ККП

Останнім часом (з 2008 року) в мережі Інтернет з'явилися фахівці користувачі, яких називають «кураторами змісту» (content curator) або «майстрами новин» (master news).

Термін куратор змісту використовується в музейну справу, в Україні таких людей називають мистецтвознавцями або експертами.

Музейні куратори не створюють контент, вони тримають руку на пульсі тенденцій, прислухаються до того, що гості обговорюють, і знаходять такі ресурси, які добре резонують з інтересами відвідувачів. Вони шукають артефакти, пов'язані з обраною темою, і організують тематичні виставки.

Сучасні інформаційні технології дозволяють будь-якому користувачеві Інтернет бути куратором різного кваліфікаційного рівня в залежності від теоретичних і практичних навичок. Професійні навички керування дозволяють уникнути інформаційного перевантаження.

Куратор змісту - це користувач мережі, який постійно контролює стан мережі, збирає, фільтрує, обробляє інформацію по заданій темі, структурує, організовує її зберігання і коментує її значимість для інших користувачів мережі.

Можна навести інші визначення керування змісту. Ось як експерти визначають керування змісту:

1) заняття змісту - це процес збору контенту, створеного іншими і цінного для вашої аудиторії і публікація його на вашій платформі (Michael Brenner);

2) у керування змісту входить вибір контенту, створеного іншими користувачами мережі, для пред'явлення співтовариству (C. Chapman);

3) заняття змісту - це процес вибору достовірної інформації відповідно до потреб читачів на певну тему фахівцем рівня редактора або куратора музею. Заняття змісту потрібно більше, ніж просто вибір інформації. Це монтаж, категоризація, коментування та подання кращого контенту (Heidi Cohen);

4) куратор змісту - це той, хто постійно знаходить, групує, організовує і поширює кращий і надійний контент з конкретного питання. Це людина, яка є експертом предметної області та підвищує цінність процесу. Хороший куратор змісту постійно і послідовно знаходиться на вершині тематичної області в якості надійного ресурсу для аудиторії, вибирає і коментує кращий і актуальний зміст. Куратор фокусується на конкретному питанні (Pawan Deshpande);

5) хороші куратори змісту знаходять, фільтрують і представляють контент своєї аудиторії в такому вигляді, який є актуальним і корисним для користувача (Barry Feldman);

6) заняття змісту - це відбір і спільне використання контенту в даній аудиторії, починаючи з агрегації (як правило, автоматизованої) курирування (ручне або напівавтоматичне) і закінчуючи аналізом (Barry Graubert);

7) заняття змісту - це процес визначення адекватного змісту по темі, просіювання і сортування до такого стану, яке буде забезпечувати найбільшу інформаційну цінність для вашої аудиторії (Kelly Hungerford);

8) заняття змісту - це процес знаходження кращого і релевантного контенту для аудиторії, для обміну з учасниками досвідом (Dave Kerpen);

9) заняття змісту - це спосіб дивитися на світ очима експерта (Michael Kolowich);

10) заняття змісту - це агрегація і узагальнення конкретного онлайн-контенту. Це часто означає сортування великих обсягів інформації та публікація її для вашої аудиторії в структурованому вигляді (Arnie Kuenn);

11) заняття змісту використовує стратегію і судження для збірки, спільного використання та поширення контенту з різних джерел для обраної аудиторії (Rebecca Lieb);

12) заняття змісту - це просто відповідна фільтрація вмісту соціальних каналів і додавання коментарів до нього для певної аудиторії (Jason Miller);

13) заняття змісту - це процес сортування великої кількості контенту в мережі та подання його користувачам (Neil Patel);

14) заняття змісту - це організація і уявлення зовнішнього, цінного змісту предметної області для певної групи користувачів в привабливому вигляді (Joe Pulizzi);

15) заняття змісту - це процес використання технологій для виявлення джерел змісту, які потім куратор фільтрує до редакційної значущості для обраної аудиторії, а потім розміщує в мережі і зберігає цю інформацію в часі (Nate Riggs);

16) заняття змісту - це процес пошуку, організації та анотування змісту. Заняття використовує інструменти обробки природної мови для фільтрації контенту на основі ключових слів, до яких потім додаються власні ідеї, проводиться аналіз і спостереження до анотацій оригінального контенту (Jake Sorofman);

17) заняття змісту - швидкий і простий спосіб переконатися, що ви ніколи не закінчите працювати з вмістом (Waynette Tubbs).

Заняття не має нічого спільного з думкою, зі збором посилань, приміток в соціальних мережах або блогах, які можуть бути цікавими. Заняття допомагає вашій аудиторії зануритися в сенс конкретної теми, питання, події або новини. Це збір і пояснення, що ілюструють різні точки зору і їх сутність.

Заняття новин в реальному часі - це мистецтво знаходження, фільтрації, відбору та перевидання високоякісних новин по конкретній темі або для конкретної аудиторії, висвітлення проблеми.

Керування змістом (content curation) це процес категоризації великої кількості контенту і подання його в організаційній функції для конкретної предметної області (ніші). Особливо ця робота важлива при організації масових відкритих дистанційних курсів, які повинні базуватися на новітній інформації, яка ще не пройшла етап узагальнення. Саме тут важливу роль відіграє якість інформації, і куратор робить цей процес обробки прозорим і якісним.

Заняття змісту є однією з форм маркетингу змісту. Воно включає в себе збір змісту, має відношення до вашої ніші, додану вартість у вигляді особистих думок і досвіду. Чим більше змісту в мережі, тим більше організація повинна взяти на себе роль її осмислення.

Заняття змісту - акт постійного виявлення, відбору та поширення кращого і релевантного онлайн-контенту та інших інтернет-ресурсів по конкретній темі, щоб відповідати потребам конкретної аудиторії.

Цифрове керування, можливо, буде новою діяльністю вчених в галузі вищої освіти, але необхідно при цьому розглянути наступні питання:

– які навички необхідні для ефективних кураторів змісту?

– як куратори повинні готуватися до цієї діяльності?

– як готувати системи в різних навчальних закладах (системи управління навчанням, обладнання, програмне забезпечення доступності і т.д.) для підтримки вчених в їхній творчості?

Перший шар керування змісту - це те, що ми отримуємо з блогів, соціальних мереж та інших незалежних джерел. Ці матеріали можуть бути дуже цінними для користувача і зможуть кожному організувати і відчути інформацію і значно скоротити час на її пошук, але без системного підходу це виглядає трохи хаотично.

Використання блогів мало для повноцінного керування з кількох причин, якщо:

– зміст організовано спонтанним і несподіваним чином;

– потрібно стежити за багатьма джерелами, щоб бути в змозі проявити цікавість і важливість новин;

– неможливо уникнути надмірності змісту новин;

– деякий вміст інформації змішується з особистим змістом;

– зміст часто вирвано з контексту, і він не може бути зрозумілим без першоджерела;

– важко виявити першоджерело після перелінковки блогу або перейменування матеріалу.

Тому наступним кроком стало автоматичне керування змісту за допомогою соціальних сервісів, наприклад, paper.li, Google News. Це була суттєва допомога для куратора, але інформація все одно вимагала персоналізації.

Таким чином:

– кількість цифрової інформації, яка доступна, вражає і знайти те, що вам потрібно в Інтернеті стає все більш складним завданням;

– ті особи, які в змозі знайти інформацію і донести до цільової аудиторії, яка працює з такою інформацією, мають величезну цінність;

– кількість інформативних даних і необхідність знайти людину, щоб курирувати їх, зростає з кожним днем.

Заняття змісту складається з трьох кроків:

– Відкриття. Для того щоб займатися зміст, ви повинні мати, чим поділитися в першу чергу. На стадії відкриття ви знайдете зміст, яким можна поділитися з аудиторією.

– Аналіз. Це етап, коли ви вирішите, що щось дійсно гідно обміну. Це дуже суб'єктивно. Але ви куратор. Ви самі вирішуєте, що є правильним для Вас і Вашої аудиторії.

– Заняття. Це етап, коли ви насправді виділяєте зміст, яке вважаєте за потрібне для спільного використання.

Але обмін контентом може йти в багатьох формах, і можна зустріти багато різних платформ, на різних носіях, особливо в умовах маркетингу.

П'ять законів керування викладені далі.

Перший закон: Люди не хочуть великої кількості інформації. Суспільство перевантажено нефільтроване, контекстно-вільними даними. Люди хочуть, щоб це припинилося.

Другий закон: куратори бувають трьох типів. Є куратори-експерти, куратори-редактори, які керують колекціями публікацій і сайтами і захоплені куратори, які люблять свою конкретну галузь.

Третій закон: керування - це не хобі, це професія і покликання. Куратори повинні бути частиною формування екосистеми і їхня праця має оплачуватися. Економічна база необхідна і неминуча.

Четвертий закон: керування вимагає технологій і інструментів для пошуку, фільтрації і перевірки вмісту в режимі реального часу. Заняття не може здійснюватися тільки людиною або програмою.

П'ятий закон: керування в вузьких, цілеспрямованих, високоякісних категоріях з'явиться, щоб конкурувати з мас-медіа.

Розглянемо ключові причини важливості кураторів змісту в навчанні:

1) Перелік інформації, яку необхідно організувати.

Студента необхідно навчити вчитися, щоб він знав, де шукати інформацію і що є актуальним для навчання або досягнення певної мети. Саме тому цифрова грамотність має таке велике значення. Вона забезпечує інструменти для оцінки, фільтрації і впорядкування інформації найбільш ефективними способами.

2) Зростаюче число відкритих ресурсів. Кількість відкритих ресурсів для навчання зростає і студентам знадобляться рекомендації по вибору найбільш якісного ресурсу для ефективного досягнення своїх цілей.

3) Світ інформації з статичного перетворюється в динамічний.

Кількість і складність доступної інформації швидко зростає, найчастіше старі поняття змінюються, тому важливо:

– направляти зусилля на пошук, моніторинг та оновлення, які є найбільш актуальними «джерелами інформації»;

– оснащувати студентів інструментами для виконання таких завдань.

Куратор змісту використовує ці навички та підходи для досягнення своїх цілей. Ось чому навички керування, ймовірно, будуть ключовими рисами майбутніх викладачів.

4) Підготовка студентів до реальної роботи. У той час як академічний світ від початкової школи до університетів в основному організований навколо теорій, реальний світ являє собою складну мережу ситуацій. Заняття вносить в навчання елементи підготовки до реальної роботи, показуючи нові відносини між різними інформаційними елементами.

5) Заняття - новий пошук.

Результати пошуку стають все більш незадовільними, оскільки вони містять велику кількість «шуму», достовірність якого складно перевірити. Іншими словами, дослідники,

викладачі та керівники все частіше вважають за краще звертатися до довірених кураторам з конкретних галузей інформації, а не покладатися на традиційний пошук.

6) Ринок праці швидко змінюється.

Факт: (в США) 17 млн. Випускників коледжів займають робочі місця, які не потребують вищої освіти. Це «більше 30 відсотків випускників коледжів в США». Заняття змісту пропонує практичні підходи тренування нових навичок для студентів.

7) Альтернативні системи сертифікації. Альтернативні системи сертифікації можуть демонструвати і оцінювати навички людини без необхідності відвідувати навчальні курси, платити за навчання і нові підручники, за іспити. Вартість навчання для сертифікованих навчальних програм дуже висока, в той час як справжнє значення цих курсів на ринку праці продовжує швидко зменшуватися.

Для розвитку організацій доцільно переходити від навчання і сертифікації до:

– керування талантів;
– керування освітніх ресурсів;
– формування навичок куратора змісту.

8) Вчителі та викладачі можуть тепер куруватиме свою сферу інтересів.

Викладачі та тренери, сьогодні починають займатися свої навчальні ресурси за рахунок використання як кількості відкритих навчальних матеріалів в Інтернеті, так і зростаючого числа інструментів.

9) Ринок освіти відкритий для конкурентів.

Сьогодні існує велика кількість вільних і доступних цифрових інструментів, веб-сервісів і додатків для створення, пошуку, редагування та публікації курсів, навчальних посібників, довідників по будь-якій темі. Куратори змісту можуть отримати репутацію фахівця з виявлення, відбору та організації кращих відкритих ресурсів для конкретних потреб.

Хтось повинен буде зібрати, організувати величезну кількість відкритих навчальних курсів і навчальних матеріалів через створення методу навчання для конкретної аудиторії або потреб.

10) Зростаючий попит на надійне керівництво в навчанні і в змісті навчальних програм.

Коли освітні пропозиції стають широкими, зростає необхідність знайти підходящі і надійні ресурси. Тому стає важливим пошук і вибір якісних відкритих освітніх ресурсів.

3 ОРГАНІЗАЦІЯ КЛІЄНТ-СЕРВЕРНИХ ДОДАТКІВ

Перш ніж почати цей розділ, варто внести деяке уточнення: під архітектурою додатки мається на увазі архітектура системи в цілому (система може складатися з ряду окремих самостійних додатків-мікросервісов). Архітектура окремих додатків буде розглянута в наступних підрозділах [7].

Як правило, комп'ютери і програми, що входять до складу інформаційної системи, не є рівноправними. Деякі з них володіють ресурсами (файлова система, процесор, принтер, база даних і т.д.), інші мають можливість звертатися до цих ресурсів. Комп'ютер (або програму), керуючий ресурсом, називають сервером цього ресурсу (файл-сервер, сервер бази даних, обчислювальний сервер ...). Клієнт і сервер будь-якого ресурсу можуть знаходиться як в рамках однієї обчислювальної системи, так і на різних комп'ютерах, пов'язаних мережею.

Основний принцип технології "клієнт-сервер" полягає в поділі функцій додатка на три групи:

- введення і відображення даних (призначений для користувача інтерфейс);
- прикладні функції, характерні для даної предметної області;
- функції управління ресурсами (файловою системою, базою даних і т.д.).

Тому, в будь-якому додатку виділяються наступні компоненти:

- компонент представлення даних;
- прикладної компонент;
- компонент управління ресурсом.

Зв'язок між компонентами здійснюється за певними правилами, які називають "протокол взаємодії".

Дволанкова архітектура клієнт-серверних додатків. Дволанкова (two-tier, 2-tier) вона називається через необхідність розподілу трьох базових компонентів між двома вузлами (клієнтом і сервером).

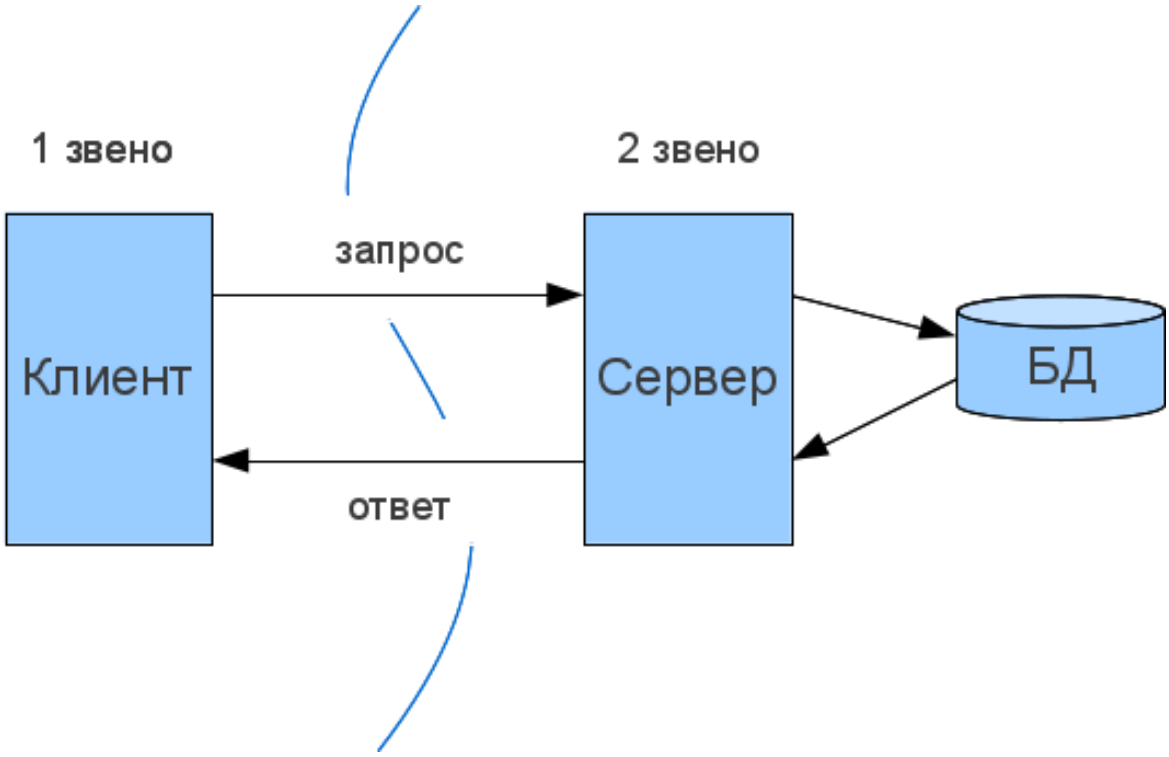


Рисунок 3.1 - Дволанковий архітектура клієнт-серверного додатка

Дволанкова архітектура використовується в клієнт-серверних системах, де сервер відповідає на клієнтські запити безпосередньо і в повному обсязі, при цьому використовуючи тільки власні ресурси, тобто сервер не викликає сторонні мережеві програми, але не звертається до сторонніх ресурсів для виконання будь-якої частини запиту.

Розташування компонентів на стороні клієнта або сервера визначає наступні основні моделі їх взаємодії в рамках дволанкової архітектури:

- сервер терміналів - розподілене представлення даних;
- файл-сервер - доступ до віддаленої бази даних і файлових ресурсів;
- сервер БД - віддалене уявлення даних;
- сервер додатків - віддалений додаток.

Історично першою з'явилася модель розподіленого представлення даних (модель сервер терміналів). Вона реалізовувалася на універсальній ЕОМ (мейнфрейми), що виступала в ролі сервера, з підключеними до неї алфавітно-цифровими терміналами. Користувачі виконували введення даних з клавіатури терміналу, які потім передавалися на мейнфрейм і там виконувалася їх обробка, включаючи формування «картинки» з результатами. Ця «картинка» і поверталася користувачеві на екран терміналу.

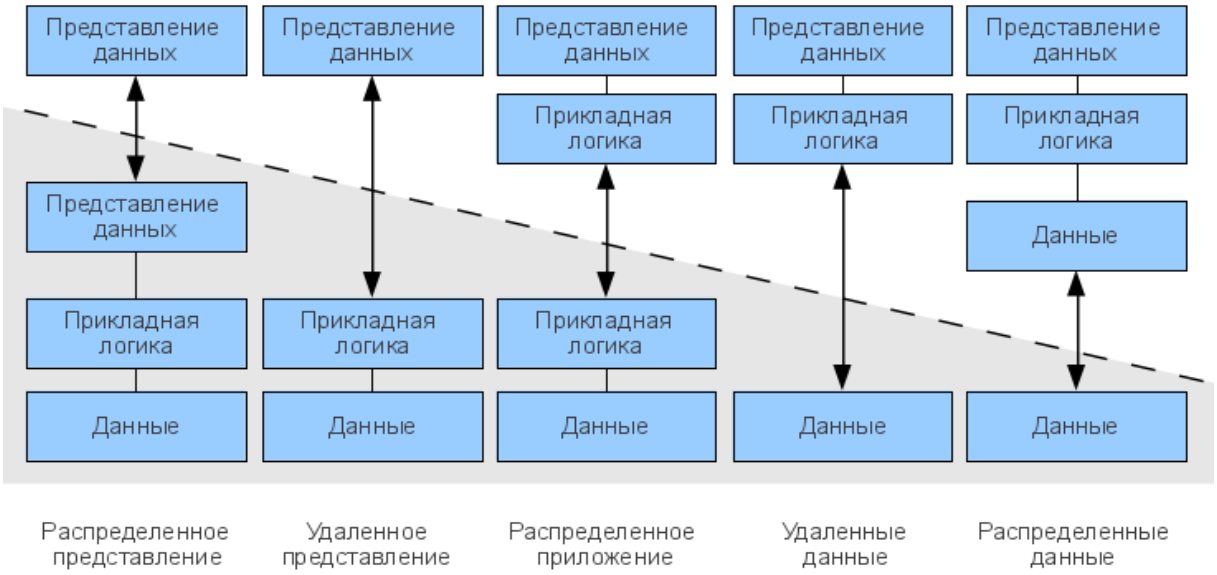


Рисунок 3.2 - Моделі клієнт-серверної взаємодії

З появою персональних комп'ютерів і локальних мереж, була реалізована модель файлового сервера, який представляв доступ файлових ресурсів, в т.ч і до віддаленої базі даних. В цьому випадку виділений вузол мережі є файловим сервером, на якому розміщені файли бази даних. На клієнтах виконуються додатки, в яких поєднані компонент уявлення та прикладної компонент (СУБД і прикладна програма), що використовують підключену віддалену базу як локальний файл. Протоколи обміну при цьому представляють набір низькорівневих викликів операцій файлової системи.

Така модель показала свою неефективність з огляду на те, що при активній роботі з таблицями БД виникає велике навантаження на мережу. Частковим вирішенням цієї проблеми є підтримка тиражування (реплікації) таблиць і запитів. В цьому випадку, наприклад, при зміні даних, оновлюється не вся таблиця, а тільки модифікована її частина.

З появою спеціалізованих СУБД з'явилася можливість реалізації іншої моделі доступу до віддаленої базі даних - моделі сервера баз даних. В цьому випадку ядро СУБД функціонує на сервері, прикладна програма на клієнті, а протокол обміну забезпечується за допомогою мови SQL. Такий підхід в порівнянні з файловим сервером веде до зменшення завантаження мережі й уніфікації інтерфейсу "клієнт-сервер». Однак, мережевий трафік залишається досить високим, крім того, як і раніше неможливо задовільний адміністрування додатків, оскільки в одній програмі поєднуються різні функції.

З розробкою і впровадженням на рівні серверів баз даних механізму збережених процедур з'явилася концепція активного сервера БД. У цьому випадку частина функцій прикладного компонента реалізовані у вигляді збережених процедур, що виконуються на стороні сервера. Решта прикладна логіка виконується на стороні клієнта. Протокол взаємодії - відповідний діалект мови SQL.

Переваги такого підходу:

– можливо централізоване адміністрування прикладних функцій;

– зниження вартості володіння системою (TOC, total cost of ownership) за рахунок оренди сервера, а не його покупки;

– значне зниження мережевого трафіку (тому що передаються не SQL-запити, а виклики збережених процедур).

Основний недолік - обмеженість коштів розробки збережених процедур у порівнянні з мовами високого рівня.

Реалізація прикладного компонента на стороні сервера надає наступну модель - сервер додатків. Перенесення функцій прикладного компонента на сервер знижує вимоги до конфігурації клієнтів і спрощує адміністрування, але представляє підвищені вимоги до продуктивності, безпеки і надійності сервера.

В даний час намічається тенденція повернення до того, з чого починалася клієнт-серверна архітектура - до централізації обчислень на основі моделі термінал-сервера. У сучасній реінкарнації термінали відрізняються від своїх алфавітно-цифрових предків тим, що, маючи мінімум програмних і апаратних засобів, представляють мультимедійні можливості (в т.ч. графічний користувальницький інтерфейс). Роботу терміналів забезпечує високопродуктивний сервер, куди винесено все, аж до віртуальних драйверів пристроїв, включаючи драйвери відеопідсистеми.

Триланкова архітектура клієнт-серверних додатків. Дана тенденція в клієнт-серверних технологіях пов'язана з дедалі більшим використанням розподілених обчислень. Вони реалізуються на основі моделі сервера додатків, де мережевий додаток розділене на дві і більше частин, кожна з яких може виконуватися на окремому комп'ютері. Виділені частини додатка взаємодіють один з одним, обмінюючись повідомленнями в заздалегідь узгодженому форматі. В цьому випадку дволанкова клієнт-серверна архітектура стає триланковою (three-tier, 3-tier).

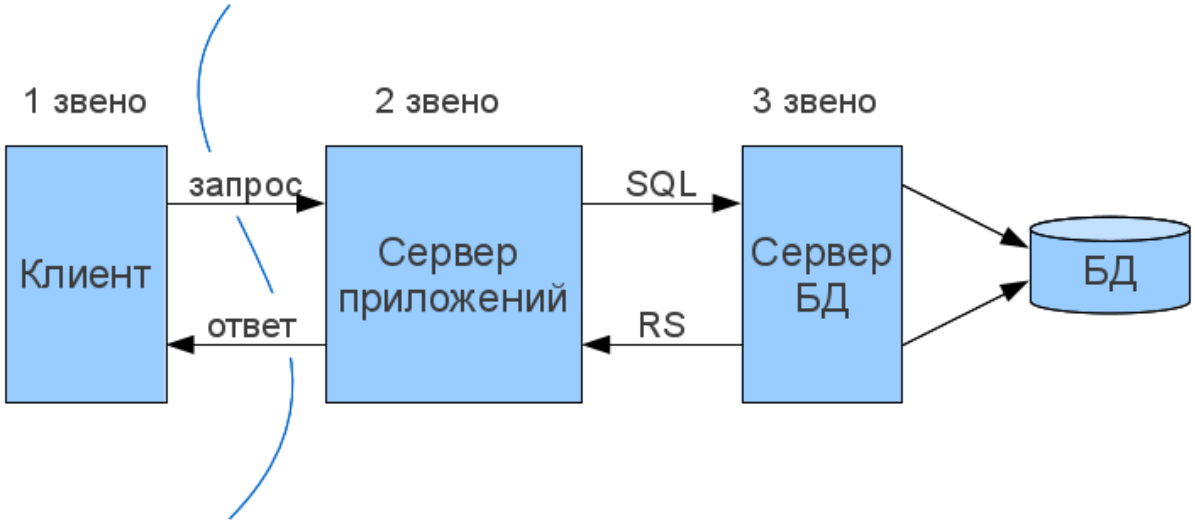


Рисунок 3.3 - Триланкова архітектура клієнт-серверного додатка

Як правило, третьою ланкою в триланкової архітектурі стає сервер додатків, тобто компоненти розподіляються наступним чином:

– представлення даних - на стороні клієнта;

– прикладний компонент - на виділеному сервері додатків (як варіант, що виконує функції проміжного ПО);

– управління ресурсами - на сервері БД, який і представляє запитувані дані.

Триланкова архітектура може бути розширена до багатоланкової (N-tier, Multi-tier) шляхом виділення додаткових серверів, кожен з яких представлятиме власні сервіси і користуватися послугами інших серверів різного рівня.

Підводячи підсумки можна сказати, що триланкова архітектура складніше, але завдяки тому, що функції розподілені між серверами другого і третього рівня, ця архітектура являє:

– високий ступінь гнучкості і масштабованості;

– високу безпеку (тому що захист можна визначити для кожного сервісу або рівня);

– високу продуктивність (тому що завдання розподілені між серверами).

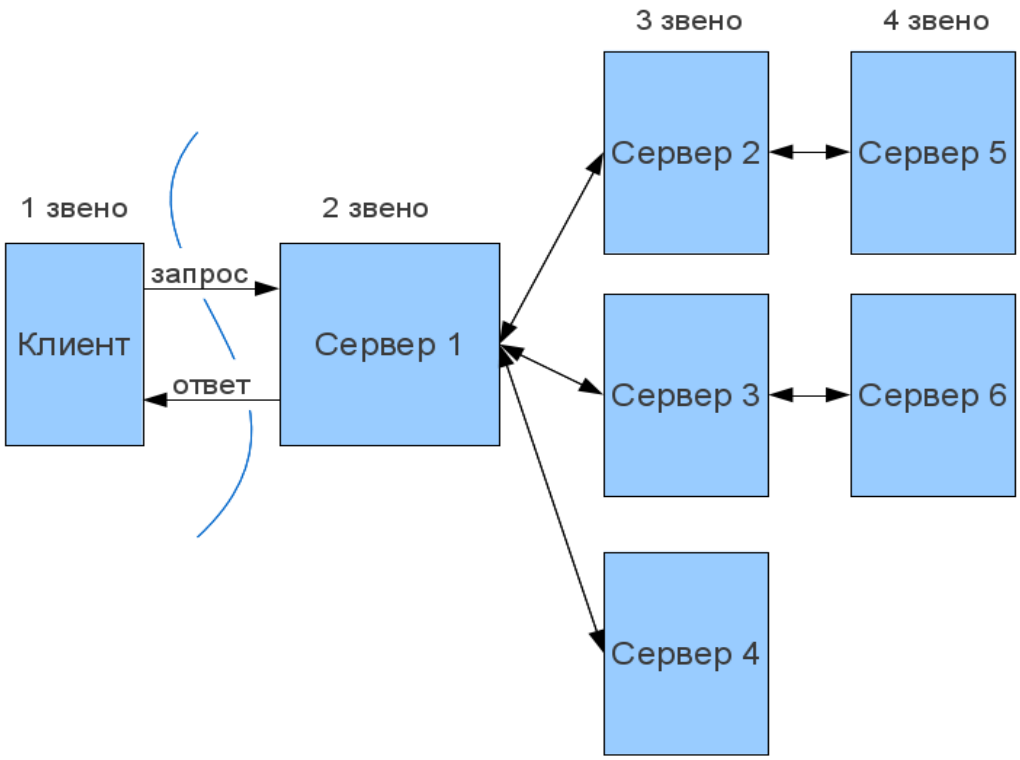


Рисунок 3.4 - Багатоланкова (N-tier) клієнт-серверна архітектура

3.1 Архітектура клієнт-серверних додатків. Шаблони MVC і MVVM

Шаблон проектування або патерн в розробці програмного забезпечення - повторювана архітектурна конструкція, що представляє собою рішення проблеми проектування в рамках деякого часто виникає контексту.

Зазвичай шаблон не є закінченим зразком, який може бути прямо перетворений в код; це лише приклади розв'язання задач, який можна використовувати в різних ситуаціях. Об'єктно-орієнтовані шаблони показують відносини і взаємодії між класами або об'єктами, без визначення того, які кінцеві класи або об'єкти додатки будуть використовуватися.

«Низькорівневі» шаблони, що враховують специфіку конкретної мови програмування, називаються ідіомами. Це хороші рішення проектування, характерні для конкретної мови або програмної платформи, і тому не універсальні.

На найвищому рівні існують архітектурні шаблони, вони охоплюють собою архітектуру всієї програмної системи.

Алгоритми за своєю суттю також є шаблонами, але не проектування, а обчислення, так як вирішують обчислювальні завдання.

Model-View-Controller (MVC, «Модель-Представлення-Контролер», «Модель-Вид-Контролер») - схема поділу даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: модель, уявлення і контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно [8].

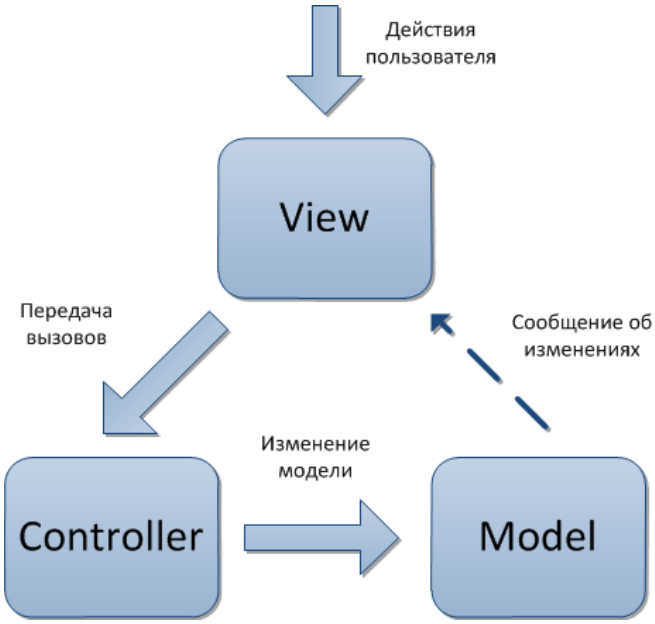


Рисунок 3.5 - Шаблон MVC

В архітектурі MVC модель надає дані і правила бізнес-логіки, уявлення відповідає за користувальницький інтерфейс, а контролер забезпечує взаємодію між моделлю і представленням.

Типову послідовність роботи MVC-додатки можна описати таким чином:

– додаток отримує запит від користувача і визначає запитані контролер і тип запиту;

– відпрацьовує необхідний контролер, в якому, наприклад, міститися виклики моделі, що зчитують інформацію з бази даних;

– після цього, дія формує уявлення з даними, отриманими з моделі і виводить результат користувачеві.

Модель - містить бізнес-логіку додатка і включає методи вибірки (це можуть бути методи ORM), обробки (наприклад, правила валідації) і надання конкретних даних, що часто робить її дуже товстою, що цілком нормально.

Модель не повинна безпосередньо взаємодіяти з користувачем. Всі змінні, що відносяться до запиту користувача повинні оброблятися в контролері.

Модель не повинна генерувати HTML або інший код відображення, який може змінюватися в залежності від потреб користувача. Такий код повинен оброблятися в видах.

Одна і та ж модель, наприклад, модель аутентифікації користувачів може використовуватися як в призначеній для користувача, так і в адміністративній частині програми. В такому випадку можна винести загальний код в окремий клас і успадковуватися від нього, визначаючи в спадкоємців специфічні методи.

Вид - використовується для завдання зовнішнього відображення даних, отриманих з контролера і моделі. Види містять HTML-розмітку і невеликі вставки PHP-коду для обходу, форматування і відображення даних. Види зазвичай поділяють на загальний шаблон, що містить розмітку, загальну для всіх сторінок (наприклад, шапку і підвал) і частини шаблону, які використовують для відображення даних, що виводяться з моделі або відображення форм введення даних.

Контролер - сполучна ланка, що з'єднує моделі, види і інші компоненти в робоче додаток. Контролер відповідає за обробку запитів користувача. Контролер не повинен містити SQL-запитів. Їх краще тримати в моделях. Контролер не повинен містити HTML і інший розмітки. Її варто виносити в види.

У добре спроектованому MVC-додатку контролери зазвичай дуже тонкі і містять тільки кілька десятків рядків коду. Логіка контролера досить типова і велика її частина виноситься в базові класи.

Моделі, навпаки, дуже товсті і містять велику частину коду, пов'язану з обробкою даних, тому що структура даних і бізнес-логіка, що міститься в них, зазвичай досить специфічна для конкретного додатка.

Варто зауважити, що все описане вище може варіюватися в залежності від використовуваної технології (веб-фреймворку), оскільки підходи можуть бути дуже різні, в деяких ситуаціях контролери не містять багато логіки, вона виноситься в моделі, а в деяких навпаки. Також для деяких фреймворків характерна наявність такого поняття як сервіси, які містять основну логіку і можуть бути використані як в моделях, так і в контролерах.

Патерн MVVM (Model-View-ViewModel) дозволяє відокремити логіку додатки від візуальної частини (подання). Даний патерн є архітектурним, тобто він задає загальну

архітектуру програми. Спочатку з'явився для обходу обмежень патернів MVC і MVP, і об'єднання деякі з їх сильних сторін [9].

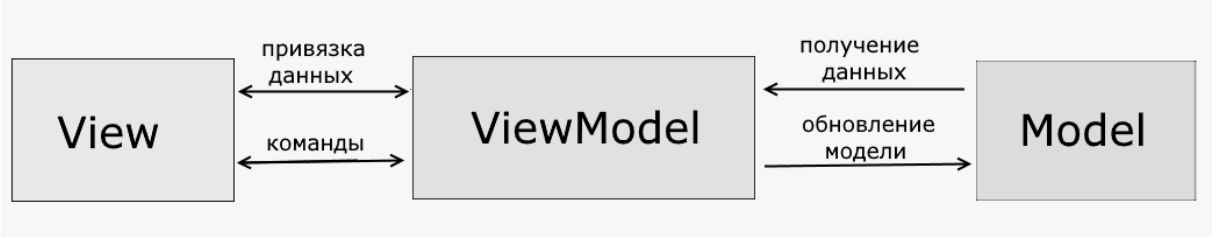


Рисунок 3.6 - Шаблон MVVM

MVVM складається з трьох компонентів: моделі (Model), моделі подання (ViewModel) і уявлення (View).

Модель описує використовувані в додатку дані. Моделі можуть містити логіку, безпосередньо пов'язану цими даними, наприклад, логіку валідації властивостей моделі. У той же час модель не повинна містити ніякої логіки, пов'язаної з відображенням даних і взаємодією з візуальними елементами управління.

Нерідко модель реалізує інтерфейси INotifyPropertyChanged або INotifyCollectionChanged, які дозволяють повідомляти систему про зміни властивостей моделі. Завдяки цьому полегшується прив'язка до подання, хоча знову ж пряму взаємодію між моделлю і представленням відсутня.

View або подання визначає візуальний інтерфейс, через який користувач взаємодіє з додатком. Стосовно до WPF уявлення - це код в xaml, який визначає інтерфейс у вигляді кнопок, текстових полів та інших візуальних елементів.

Хоча вікно (клас Window) в WPF може містити як інтерфейс в xaml, так і прив'язаний до нього код C #, проте в ідеалі код C # не повинен містити якийсь логіки, крім хіба що конструктора, який викликає метод InitializeComponent і виконує початкову ініціалізацію вікна . Вся ж основна логіка додатки виноситься в компонент ViewModel.

Однак іноді в файлі пов'язаного коду все може знаходитися певна логіка, яку важко реалізувати в рамках паттерна MVVM у ViewModel.

Подання і не виконує жодних події за рідкісним винятком, а виконує дії в основному за допомогою команд.

ViewModel або модель уявлення пов'язує модель і уявлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації

моделлю інтерфейсу INotifyPropertyChanged автоматично йде зміна відображуваних даних в поданні, хоча безпосередньо модель і уявлення не пов'язані.

ViewModel також містить логіку по отриманню даних з моделі, які потім передаються в уявлення. І також VewModel визначає логіку по оновленню даних в моделі.

Оскільки елементи уявлення, тобто візуальні компоненти типу кнопок, не використовують події, то уявлення взаємодіє з ViewModel за допомогою команд.

Наприклад, користувач хоче зберегти введені в текстове поле дані. Він натискає на кнопку і тим самим відправляє команду під ViewModel. А ViewModel вже отримує передані дані і відповідно до них оновлює модель.

Підсумком застосування патерну MVVM є функціональний розподіл програми на три компонента, які простіше розробляти і тестувати, а також в подальшому модифікувати і підтримувати.

3.2 Архітектура клієнтських додатків

З дедалі більшим розвитком різних технологій і серверної частини, клієнтські програми також дуже ускладнилися. На світ з'явилися такі різновиди мови JavaScript як CoffeeScript, TypeScript, asm.js. Я називаю їх різновидами лише з тієї причини, що на клієнтської частини вони виконують ту саму функцію. Також не варто забувати те, що сам JS дуже активно розвивається, ще не всі браузери повноцінно підтримують EcmaScript 5, з EcmaScript 6 підтримуються лише деякі можливості (функції), а спільнота вже чекає виходу в світ EcmaScript 7.

Жодна мова не може похвалитися таким різноманіттям різних технологій, бібліотек і фреймворків, яким може похвалитися JavaScript. Навіть якщо взяти звичайний менеджер пакетів, то JS має як мінімум 3 таких:

- Bower - менеджер пакетів для клієнтської частини.
- NPM - менеджер пакетів для node.js.
- Yarn - покращений варіант NPM.

Хоча останнім часом ходять bower перестає бути затребуваним і ймовірно, що скоро він взагалі не буде використовуватися.

Ще існує величезна кількість технологій, які полегшують життя розробників (я не буду згадувати тут Flow, Babel і подібне):

– Grunt - task-runner. має величезну кількість плагінів для Less, Babel, etc.

– Gulp - більш сучасний аналог Grunt.

– Webpack - складальник пакетів. Має величезну кількість плагінів для різних цілей і навіть тестовий сервер.

Останнє, про що варто згадати - це величезна різноманітність фреймворків. Будуть вказані тільки декілька з них (які, на мою думку, є (були) найактуальнішими:

– React.js.

– Angular.js.

– Backbone.

– Knockout.js.

В даному списку немає jQuery оскільки цю бібліотеку не можна вважати повноцінним фреймворком для розробки клієнтської частини. Та й взагалі останнім часом багато розробників намагаються її уникати в зв'язку з її ваговитістю.

Далі мова піде про сам React.js і про деякі нюанси розробки на ньому. React.js - бібліотека (фреймворк) для створення користувацьких інтерфейсів, яка вирішує проблему часткового оновлення сторінки.

React має ряд основних концепцій, без яких складно розібратися в тому, як він взагалі працює і почати його використовувати:

Елементи - це об'єкти JavaScript, які представляють HTML-елементи. Їх не існують в браузері. вони описують DOM-елементи, такі як h1, div, або section.

Компоненти - це елементи React, написані розробником. Зазвичай це частини призначеного для користувача інтерфейсу, які містять свою структуру і функціональність. Наприклад, такі як NavBar, LikeButton, або ImageUploader.

JSX - спеціально розроблений тип файлів, що значно спрощує опис React компонентів.

Virtual DOM - це дерево React елементів на JavaScript. React отрісовиває Virtual DOM в браузері, щоб зробити інтерфейс видимим. React стежить за змінами в Virtual DOM і автоматично змінює DOM в браузері так, щоб він відповідав віртуальному. Причому відбувається не повна зміна DOM, а лише тих елементів, де це необхідно. Хоча не можна сказати, що все дуже швидко і добре. Існують деякі ситуації, коли ReactDOM перерендерівает зайві компоненти, але це вже окрема історія і ймовірно, в майбутньому розробники це виправлять.

Спочатку при ініціалізації сторінки React будує віртуальне DOM дерево, після чого проводить порівняння з DOM деревом сторінки і добудовує недостающіі компоненти.

Кожен компонент має так звані props - це такі собі дані, які були передані йому <SomeComponent defaultValue = "Test" />, таким чином defaultvalue потрапить в словник this.props нашого компонента SomeComponent. Якщо ми маємо композицію компонентів і в одному з батьків з якихось причин відбувається зміна стан, яке впливає на props дочірніх компонентів, то ми можемо це відстежити в дочірніх компонентах за допомогою перевизначення методу componentWillReceiveProps (newProps), який приймає, нові, тільки що змінені props.

Також компонент має state. Це ні що інше як стан компонента і саме при його зміні відбувається перерендер віртуального DOM дерева і зміна реально DOM, в разі, якщо між ними було виявлено невідповідність. Установка стану відбувається за допомогою виклику методу this.setState ({someValue: 'Test'}) в контексті React компонента. Варто знати. що дана методу виконується асинхронно і зміна стану компонента відбудеться не миттєво, тому, якщо у нас є необхідність відразу ж використовувати тільки що встановлений стан, то краще записати його безпосередньо в словник this.state, після чого викликати метод this.setState ({}) з порожнім словником для виклику механізму перерендера DOM.

Ще однією важливою складовою компонентів є контексти. Завдяки їм можна більш зручно зв'язувати дочірні компоненти з батьківськими без використання методу componentWillReceiveProps, при цьому навіть контролювати тип одержуваних даних завдяки механізмам React (також є можливість контролювати тип одержуваних props). Але з появою технологій, призначених для так званого, State Management-а контексти починають використовуватися набагато рідше.

Є одна основна проблема при розробці React додатків: відстежити зміну дочірнього компонента або батьківського нескладно, але в разі глибокої вкладеності або, коли компоненти взагалі не мають загального контейнера (у вигляді React компонента), обмін станами ставати дуже непростим або взагалі неможливим, а реалізація незрозумілою і заплутаною. Дану проблему вирішують шаблони Flux і Redux, які будуть описані в наступному розділі.

3.3 State-management. Шаблони Flux і Redux.

Flux - архітектурний підхід або набір шаблонів програмування для побудови призначеного для користувача інтерфейсу веб-додатків, що поєднується з реактивним програмуванням і побудований на односпрямованих потоках даних [10]. Згідно із задумом творців і незважаючи на те, що Facebook надав реалізацію Flux на додаток до React, Flux не є ще одним веб-фреймворком, а є архітектурним рішенням і може бути застосований абсолютно де завгодно, в разі, якщо подібний підхід може вирішити поставлену задачу. Основною відмінною рисою Flux є односторонняя спрямованість передачі даних між компонентами Flux-архітектури. Архітектура накладає обмеження на потік даних, зокрема, виключаючи можливість поновлення стану компонентів самими собою. Такий підхід робить потік даних передбачуваним і дозволяє легше простежити причини можливих помилок в програмному забезпеченні

Перш ніж перейти до опису компонентів Flux варто розповісти про такий шаблон як Dispatcher (диспетчер), оскільки він є основною частиною Flux. Диспетчер - це шаблон проектування, який має API для підписки на певні події, а також API для виклику певних подій. Механізм роботи банально проста, при виклику певної події відпрацьовують всі функції, підписані на дану подію.

Основними компонентами Flux є наступні [11].

– Actions (Дії) - вираз подій (часто для дій використовуються просто імена - рядки, що містять деякий «дієслово»). Диспетчери передають дії нижчого компонентів (сховищ або диспетчеру) по одному. Нова дія не передається поки попереднє повністю не оброблено компонентами. Дії через роботу джерела дії, наприклад, користувача, надходять асинхронно, але їх диспетчеризація є синхронним процесом. Крім імені, дії можуть мати корисне навантаження, що містить пов'язані з дією дані.

– Dispatcher (Диспетчер) - призначений для передачі дій сховищ. У спрощеному варіанті диспетчер може взагалі не виділятися, як єдиний на весь додаток. У диспетчері сховища реєструють свої функції зворотного виклику (callback) і залежності між сховищами.

– Stores (Сховища) - є місцем, де зосереджено стан додатки. Інші компоненти, згідно Flux, не мають значного (з точки зору архітектури) стану. Зміна стану сховища відбувається строго на основі даних дії і старого стану сховища.

– Views (Уявлення) - компонент, звичайно відповідає за видачу інформації користувачеві. У Flux-архітектурі, яка може технічно не торкатися внутрішнього облаштування уявлень взагалі, це - кінцева точка потоків даних. Для інформаційної архітектури важливо тільки, що дані потрапляють в систему (тобто, назад в сховища) тільки через дії.

Основні відмінні риси Flux архітектури:

– синхронність - все методи зворотного виклику, зареєстровані для кожної дії, синхронні у виконанні, саме ж дія може викликатися джерелом асинхронно;

– інверсія управління потік управління передається соответвуют сховища і цільової функції зворотного виклику;

– семантичні дії - дія, що викликається джерелом, містить смислове інформацію, що дозволяє відповідному сховища вибрати правильний метод виконання;

– відсутність каскадів дій - Flux забороняє каскадні (вкладені) дії.

Можна помітити, що проблема передачі властивостей дочірнім компонентів залишається, але вирішується проблема зв'язування компонентів, які не мають загального батька. Оскільки ці компоненти можуть повністю зберігати свої статки в Store, який в свою чергу буде змінюватися при виклику певних подій в диспетчері. Таким чином ми отримуємо можливість будувати ще більш гнучкі клієнтські програми, так як у нас буде можливість просто визначити певний набір компонентів, який ми будемо використовувати на кожній сторінці, при цьому ми маємо можливість робити кожен раз різну зв'язку за допомогою Flux, а всі компоненти будуть перевикористати.

Архітектура Flux досить проста для розуміння. Саме завдяки цьому її воліє велика кількість розробників. Тим більше можна кілька переробити архітектуру під свої конкретні потреби, якщо є така необхідність.

Виходячи з рис. 3.7 можна помітити, що подія може бути пов'язано з викликом, будь-якого API. Отже, обробка події буде передана в Диспетчер в той момент, коли буде отримано відповідь від API, або в разі невдачі буде передано повідомлення про помилку, або повідомлення з нульовим результатом. У будь-якому випадку це вже буде регулюватися розробником.

Архітектура Redux дуже схожа з Flux. Навіть можна сказати, що вона є деякою надбудовою над Flux. Також вона набагато складніше для розуміння і набагато заплутаніше (заплутаніше саме в процесі розробки, оскільки в подальшому відстежити потік даних досить просто), але в той же час і кілька функціональніша.

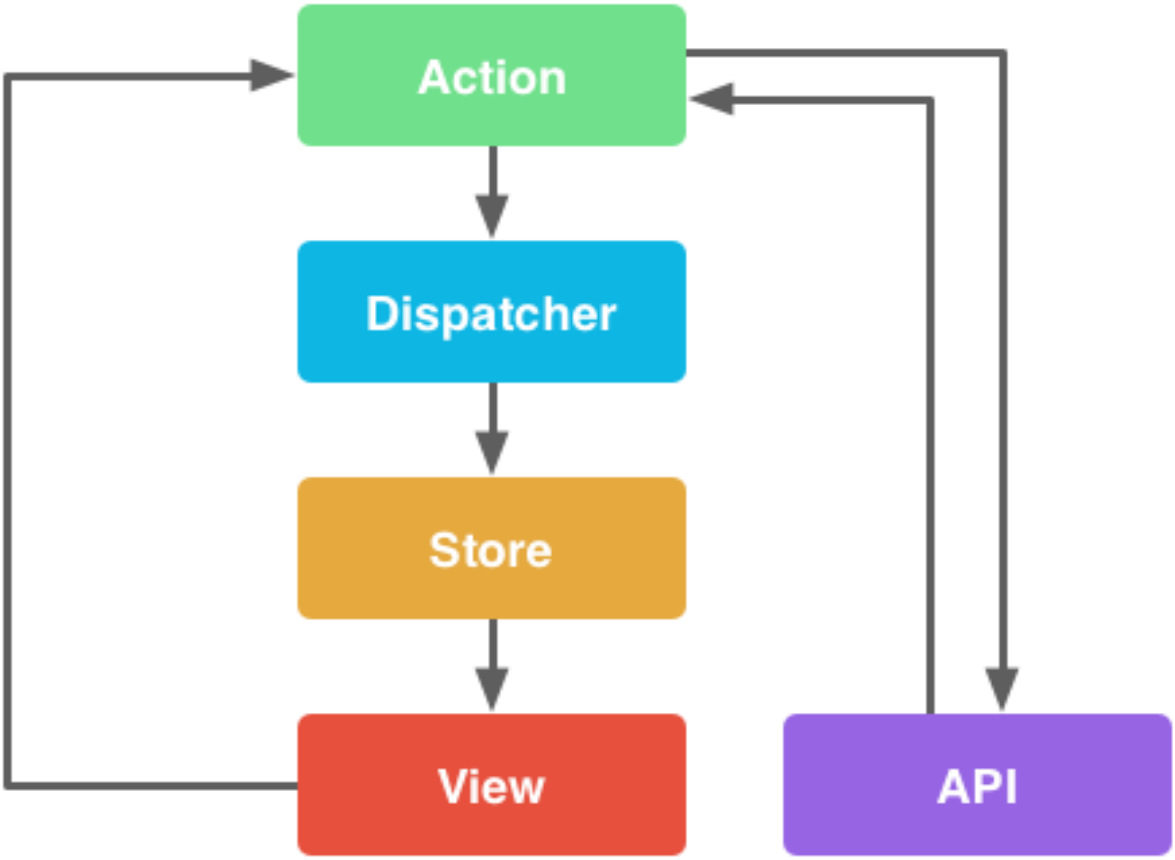


Рисунок 3.7 - Механізм роботи Flux

Redux - це бібліотека, яка реалізує контейнер для даних додатка, завдяки об'єднанню ідей від таких бібліотек, як Flux і Elm. За допомогою Redux, є можливість керувати будь-яким станом даних, за умови дотримання таких принципів [12]:

– стан даних зберігається в одиночному сховище;

– дані змінюються за допомогою дій (actions).

Ядром сховища Redux, є функція, яка приймає поточний стан даних програми, потрібну дію і комбінує їх для створення нового стану програми. Така функція називається - редуктором.

React-компоненти відповідають за відправку дій в сховище, і, в свою чергу, сховище оповіщає компоненти, коли їм потрібно змінити свій стан.

Основними частинами Redux-додатку є:

– Дії (Actions) - визначають типи дій (скоріше навіть подій). За фактом є просто строковими константами;

– Фабрики дій - функції повертають об'єкт дії. Містять тип і деякі дані мають відношення до дії. Фабрики дії повинні бути «чистими» функціями - не повинні вносити ніякі зміни в дані;

– Редуктори (Reducers) - основний компонент Redux додатки. Зберігає стан і виробляє все його зміни.

За фактом тіло редуктора складається і switch який містить ряд функцій змінюють стан і спрацьовують в залежності від типу дії.

– сховище (Store) - містить набір редукторів;

– контейнер (Container) - «розумний» React-компонент.

Дана термінологія введена розробником Redux Деном Абрамовим. Розумними є компоненти, які зберігають свій стан і можуть його змінювати, в той момент як «дурні» компоненти є чисто презентаційними та все, що вони входить до їхніх обов'язків це відображення їх props, які передаються їм з розумних «компонентів». Цей поділ не дуже коректно (як в подальшому сказав Ден Абрамов), точніше воно коректно, проблема в розробників, не можна зводити це висловлювання в абсолют, оскільки навіть «дурні» компоненти цілком можуть зберігати свій стан і в деяких ситуаціях це необхідно і правильно. Оскільки якщо більш поглиблено вдуматися в суть технології Redux, то розумними компонентами є саме ті, які обгорнуті функцією connect. І саме це є їх основною ознакою, а не те, що вони зберігають свій стан. Також дуже часто «дурні» компоненти називають презентаційними, хоча таку назву дуже характерно для багатьох клієнтських фреймворків.

Перед розбором того як саме працює Redux варто зауважити, що для його використання знадобляться як мінімум дві бібліотеки: сам redux і react-redux. React-redux містить набір корисних функцій таких як:

– Connect - створює прив'язку state, отриманого з store, і actions до контейнера;

– Provider - React-компонент, який виступає в ролі обгортки, який дістає зі сховища стан і прокидає його connect-у;

В цілому механізм роботи Redux досить простий:

– спочатку відбувається ініціалізація програми та створюються всі компоненти Редуктора ініціалізуються початковим станом;

– стан з редукторів передаються контейнерів, від них презентаційним компонентів і відбувається рендеринг сторінки;

– користувач виробляє якісь дії і діспатчітся якась подія;

– в фабриці подій генерується об'єкт події і передається в сховище, яке передає його всім редукторів, що зберігаються в ньому;

– Редуктор приймає об'єкт події і по типу визначає яка саме функція його обробить;

– Редуктор виконує функцію обробник, змінює свій стан і повертає його копію в сховище;

– Провайдер зауважує зміна сховища і передає його компоненту через connect.

– цикл замикається;

В цілому все досить просто, складність полягає в розумінні, того, що у нас може бути величезна кількість редукторів, що зберігаються в одному сховищі, і подія викликане для цього сховища змінить стан кожного такого редуктора. Redux - це та технологи, яка вимагає досить багато часу на початкове написання, але в подальшому, при необхідності внести будь-які зміни, тільки спрощує життя розробникам.

Ще однією значною відмітною особливістю Redux від Flux є можливість виробляти dispatch асинхронних подій, при цьому повноцінно їх обробляючи. Для цього необхідно підключити таку бібліотеку як redux-thunk (є й інші варіанти). Вона додається як middleware в сховище (Store), шляхом виклику функції applyMiddleware.

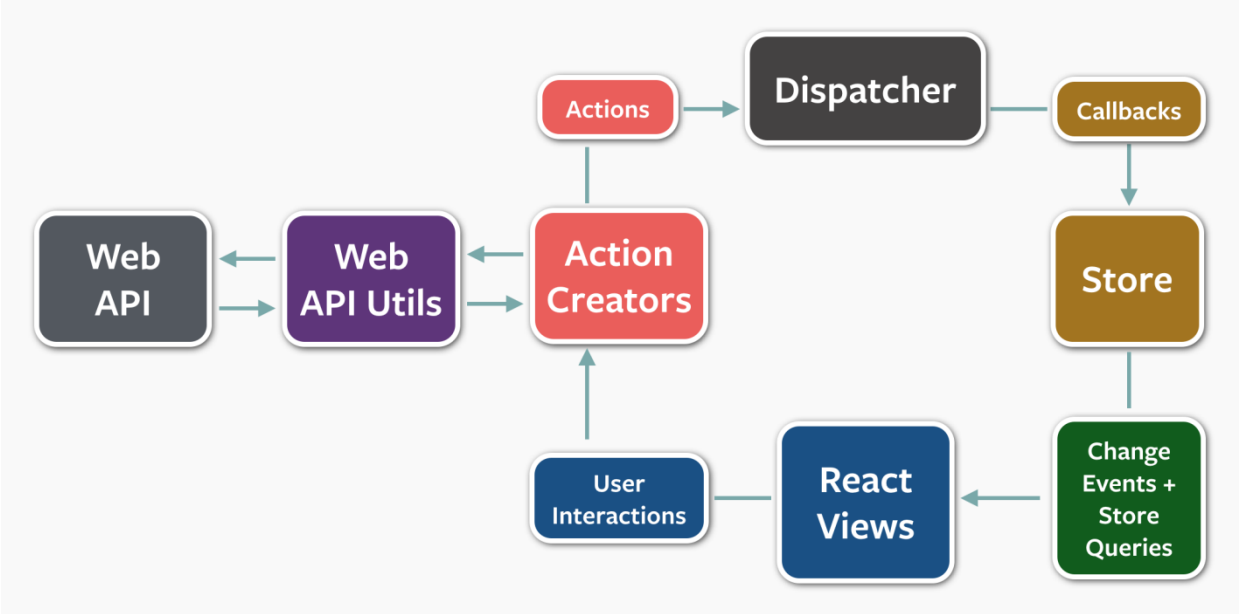


Рисунок 3.8 - Механізм роботи Redux

Механізм визначення фабрик подій дещо ускладнюється, а точніше доповнюється, оскільки тепер фабрика подій може повернути не тільки об'єкт, а й функцію. Отже, щоб коректно обробляти асинхронні події нам необхідно (допусти в прикладі нам по кліку на

кнопку треба запитувати у сервера список послуг і відображати його, а поки він грузиться - відображати Спінор):

1. Розбити нашу асинхронну подія на ряд синхронних подій. В даному випадку це початок завантаження, кінець завантаження (нам немає необхідності обробляти окремо отримання даних, оскільки ми можемо передати їх в закінчення завантаження).

```
export function startFetchingProducts () {
    return {
        type: START_FETCHING_PRODUCTS // it is a constant
    }
}

export function finishFetchingProducts (products, errorMessage) {
    return {
        type: FINISH_FETCHING_PRODUCTS,
        products: products,
        errorMessage: errorMessage
    }
}
```

2. Визначити фабрику для асинхронного події, де спочатку задіспатчити початок завантаження, а по отриманню результату - кінець.

```
export function fetchProducts (dispatch) {
    return dispatch => {
        dispatch(startFetchingProducts());
        fetch('/api/v1/get-products')
            .then(function(response) {
                if (response.status >= 400) {dispatch(finishFetchingProducts([],
"Something went wrong!")); }
                else {
                    response.json().then(data => {
                        dispatch(finishFetchingProducts(data.products, ""));
                    });
                }
            },
            function(error){
                dispatch(finishFetchingProducts([], "Something went wrong!"));
            })
    }
}
```

3. Таким чином наші редуктори спочатку оброблять початок завантаження і будуть відображати Спінор (ймовірно заблокують частина інтерфейсу). А по завершенні завантаження буде відмалювали результат і робота інтерфейсу відновиться.

3.4 Допоміжні технології

Доброю практикою в сучасній Web-розробці є застосування інструментів, які допоможуть спростити і прискорити процес розробки, а також підготувати файли проекту для продакшн сервера. Для цих цілей, ви можете використовувати такі системи для автоматизації завдань, як Grunt або Gulp, створюючи ланцюжок таких перетворення, які, наприклад, мінімізують ваш CSS або JavaScript код, з'єднувати в єдиний файл, трансплітіровать код і т.д. Подібні інструменти вкрай популярні і дуже корисні. Є, проте, інший спосіб зробити подібні дії, які реалізується за допомогою Webpack.

Webpack є збирачем модулів. Він бере JavaScript модулі з необхідними залежностями, і потім з'єднує їх разом якомога ефективнішим способом, на виході створюючи єдиний JS-файл. Наприклад, такі інструменти, як RequreJS дозволяють робити подібні речі ось вже багато років. Весь трюк полягає ось у чому. Webpack не обмежений у використанні тільки JavaScript модулів. Застосовуючи спеціальні Завантажники, Webpack розуміє, що JavaScript модулів можуть знадобитися для їх роботи, наприклад, CSS файли, а їм, в свою чергу, зображення. При цьому, результат роботи Webpack буде містити тільки ті ресурси, які дійсно потрібні для роботи програми. Webpack дастаточно складний в конфігурації, в разі, якщо необхідно.

4 РЕАЛІЗАЦІЯ СИСТЕМИ "КАБІНЕТ КУРСОВОГО ПРОЕКТУВАННЯ"

Дана система написана на пітон-фреймворку Tornado - розширюваний, неблокірующій веб-сервер і фреймворк, написаний на Python. Він був створений для використання в проекті FriendFeed, який в 2009 році придбала компанія Facebook, після чого вихідні коди Tornado були відкриті. Основною відмінністю фреймворка від інших є те, що він асинхронний (неблокірующій), що дає відчутну прискорення. Завдяки цьому, а також завдяки використанню epoll, він може обробляти тисячі одночасних з'єднань, а це значить, що цей фреймворк ідеальний для створення веб сервісів реального часу. Також він є досить легковажним, так як не навантажений власним ORM і набором великої кількості вбудованих Middleware класів (як наприклад Django), н проте має вбудований шаблонизатор.

Це також дає максимум гнучкості, оскільки розробник може сам вибрати необхідні йому технології.

Також при розробці системи був використаний GitHub (github.com) - веб-сервіс хостингу проектів з використанням системи контролю версій git, а також як соціальна мережа для розробників. Користувачі можуть створювати необмежену кількість репозиторіїв, для кожного з яких надається wiki, система issue tracking-а, є можливість проводити code review і багато іншого. GitHub на даний момент є найпопулярнішим сервісом такого роду, обігнавши Sourceforge і Google Code. Для open-source проектів використання сайту безкоштовно. При необхідності мати приватні репозиторії, є можливість використовувати платний тарифний план.

Як фреймвок для клієнтської частини був використаний React.js.

Приклад реалізації панелі адміністрування викладача можна побачити в Додатку А.

4.1 Архітектура БД і реалізація моделей

В якості бази даних використовується MongoDB - документоорієнтованих система управління базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць (написана на мові C ++). Має дуже зручну консольную утиліту, написану на JS, що дозволяє легко взаємодіяти з БД безпосередньо з консолі.

Документоорієнтованих СУБД (document-oriented database) - СУБД, спеціально призначена для зберігання ієрархічних структур даних (документів) і зазвичай реалізована за допомогою підходу NoSQL. В основі документоорієнтованих СУБД лежать документні сховища (document store), що мають структуру дерева. Структура дерева починається з кореневого вузла і може містити кілька внутрішніх і листових вузлів. Листові вузли містять дані, які при додаванні документа заносяться в індекси, що дозволяє навіть при досить складну структуру знаходити місце (шлях) шуканих даних. API для пошуку дозволяє знаходити за запитом документи і частини документів. На відміну від сховищ типу ключ-значення, вибірка за запитом до документному сховищі може містити частини великої кількості документів без повного завантаження цих документів в оперативну пам'ять. Документи можуть бути організовані (згруповані) в колекції. Їх можна вважати віддаленим аналогом таблиць реляційних СУБД, але колекції можуть містити інші колекції. Хоча документи колекції можуть бути довільними, для більш ефективного індексування краще об'єднувати в колекцію документи зі схожою структурою

Архітектура бази даних представлена на рис.4.1. Варто зазначити, що вона досить проста, є всього 6 різних сутностей даних, причому зі зв'язками 1 до 1. Серед сутностей ми можемо помітити (варто зазначити, що на увазі використовуваної БД MongoDB кожна сутність містить в собі поле _id з типом ObjectId, тому в надалі я не буду їх вказувати):

- Users (Користувачі) - користувачі системи:
  1) email - строкове значення E-mail користувача;
  2) firstName - строкове значення імені;
  3) lastName - строкове значення прізвища;
  4) group - ObjectId групи;
  5) type - ObjectId типу користувача.
- UserTypes (Типи користувачів) - типи користувачів системи (студент, викладач або адміністратор):
  1) type - строкове значення типу;
- Groups (Групи) - групи студентів%
  1) tile - строкове значення номера групи;
  2) lecturer - ObjectId користувача є викладачем;
  3) faculty - ObjectId факультету, до якого прив'язана група.
- Departments (Факультети) - факультети університету:
  1) title - строкове значення назви факультету;

– Sessions (Сесії) - сесії користувача. Використовуються для авторизації (не мають відношення до студентської сесії):

1) userId - ідентифікатор користувача, якому належить сесія;
2) token - токен сесії;

– Tasks (Завдання) - теми курсових проектів, створені викладачем:

1) title - строкове значення назви теми;
2) text - строкове значення завдання;
3) lecturer - ObjectId викладача;
4) group - ObjectId групи, до якої прив'язані завдання.

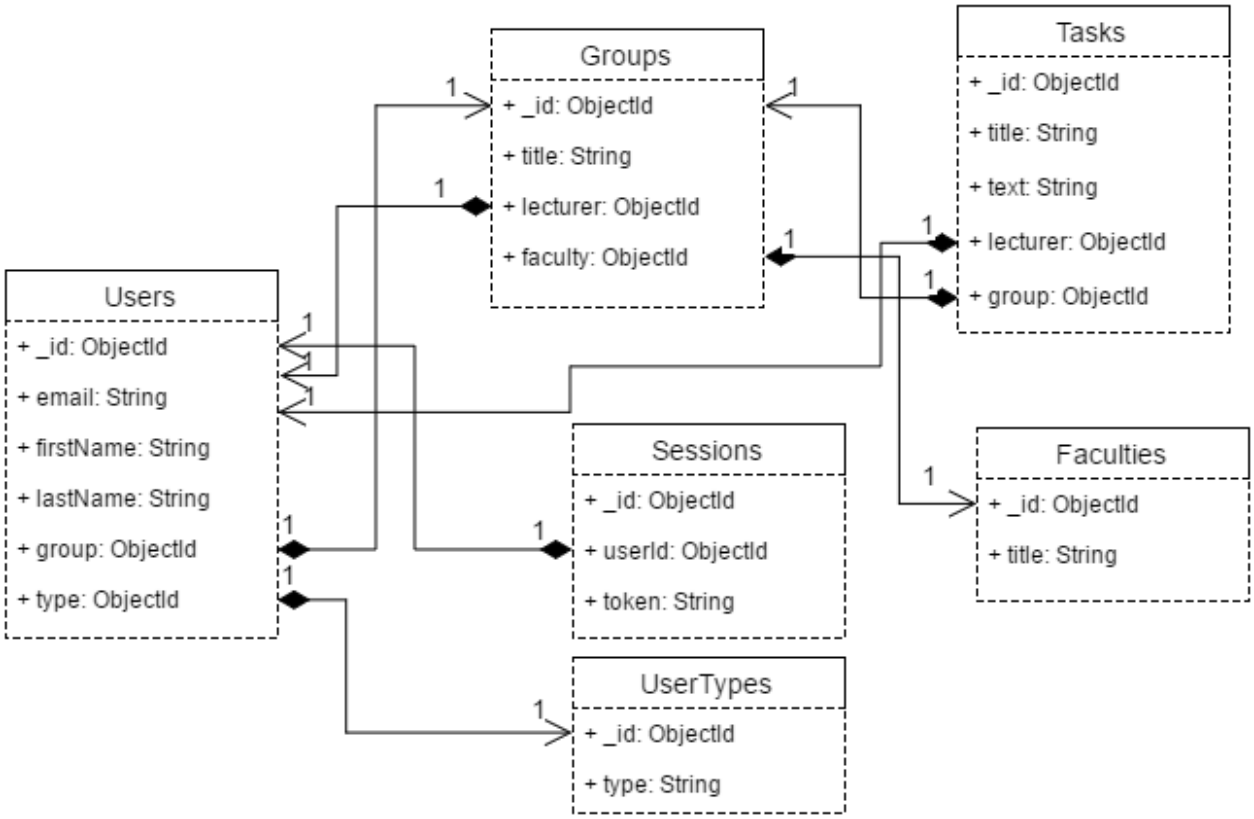


Рисунок 4.1 - UML діаграма БД

Варто зауважити, що в діаграмі відсутня сутність чатів і повідомлень. Це зроблено з метою полегшення діаграми, оскільки дані сутності Ніяк не будуть фігурувати в подальшому описі системи.

Як ORM для БД даних був обраний motor - асинхронний драйвер для MongoDB. Перш ніж почати його використання, слід створити об'єкт підключення, це робиться шляхом створення об'єкта класу MotorClient (host, port), який приймає два аргументи:

– хост, на якому розташована БД.

– порт, на якому запущений сервер БД.

У разі успіху, клієнт буде містити бази даних розташовані на сервері, і ми зможемо звертатися до них.

Моделі являють собою звичайні Python класи, просто для зручності в них виноситися ряд методів з метою подальшого полегшення коду контролерів.

```
class User:
    @classmethod
    @gen.coroutine
    def create(cls, user_dict):
        user_dict['_id'] = str(uuid.uuid4())
        result_id = yield db_client.User.insert(user_dict)
        return result_id
```

Варто також відзначити, що ми самі відповідаємо за генерацію полів _id, для цього використовується модуль uuid.

4.2 Реалізація моделі MVC в модулі авторизації

У цьому підпункті буде розглянуто модуль авторизації який раніше був описаний в пункті 2.2. Це робиться з метою пояснення як саме це пов'язано з моделлю MVC.

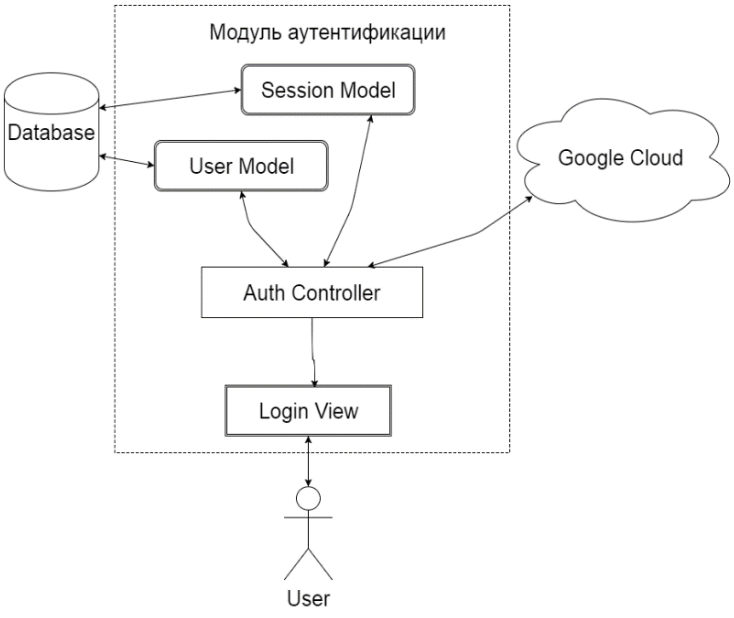


Рисунок 4.2 - Модуль аутентифікації

Модуль аутентифікації був обраний з тієї причини, що він найбільш наочно реалізує модель MVC (хоча контролер взаємодіє з рядом моделей, але це не критично). Інші модулі складаються з досить великої кількості контролерів в зв'язку з чим модель проглядатися не надто явно.

З рисунка 4.2 можна помітити, що модуль аутентифікації містить дві моделі, контролер, а також одне подання. Також, контролер взаємодіє з хмарою Google, а моделі з нашою базою - даних. Як хмара, так і БД є сторонніми сервісами, отже, вони винесені за межі нашого модуля.

Контролер в Tornado вдає із себе об'єкт класу RequestHandler, який міститься в модулі tornado.web. Для того, щоб обробляти запити, нам необхідно визначити у контролера набір методів, імена яких співпадають з назвами типів запитів.

– GET - отримання ресурсу;
– POST - створення ресурсу;
– PUT - оновлення ресурсу;
– DELETE - видалення ресурсу.

У контролері дані методи оголошуються з маленької літери. Варто зауважити, що завдяки цьому фреймворк Tornado дуже зручний для створення REST архітектури, хоча на даний момент всі сучасні веб-фреймворки мають таку можливість. Кожен з методів приймає як аргумент об'єкт самого себе (self, що характерно для всіх класів в Пітоні). Також він може приймати інші аргументи, які визначені в роутинг і передаються контролерам у вигляді аргументів. Об'єкт self містить об'єкт типу request, через який можна отримати різну необхідну інформацію:

– заголовки запиту;
– Get-параметри;
– Post-параметри;
– додаткову інформацію про запит.

Також серед корисних методів об'єкта self існують:

– Render - метод приймає в якості аргументу ім'я шаблону і словник з даними. які передаються в шаблонизатор і надалі повертаються користувачу в якості відповіді;

– Redirect - метод переадресації, як аргумент приймає шлях, на який буде перенаправлений запит.

Також існує досить велика кількість методів для роботи з cookies (наприклад set_secure_cookie, get_secure_cookie).

Аналогічна ситуація як і з cookies йде з аргументами.

Контролер торнадо обов'язково повинен повернути об'єкт відповіді, оскільки інакше він буде марним. відповідь може бути проведений різними способами, один з них це redirect, хоча це більше делегація обробки запиту іншого методу, повноцінну відповідь можна повернути шляхом виклику методу render або write (самі часто використовувані варіанти, також є можливо кинути виняток).

У Tornado є можливість обертати методи об'єкта RequestHandler (точніше ця можливість підтримується на рівні мови), а точніше це навіть є необхідним, якщо ми хочемо домогтися асинхронної обробки запитів. До речі асинхронність в Tornado домагається не за рахунок обробки запитів в різних потоках, а завдяки власним веб-сервера і циклу обробки повідомлень (event loop), оскільки якщо настає виконання блокує операції, то починає оброблятися інший запит, поки операція не буде завершена. Для досягнення асинхронности необхідно в першу чергу звертати все обробники спеціальним декоратором (на даний момент їх існує кілька варіантів) coroutine, який міститься в модулі tornado.gen. Ще одним обов'язковим моментів є використання асинхронного драйвера для бази даних, інакше очікуваний результат не буде отриманий.

4.3 Інтеграція з Google OAuth

Модуль аутентифікації використовує Google OAuth (рис. 4.3). Це дуже зручне рішення з тим урахуванням, що кожен студент має особистий E-mail в домені snu.edu.ua. Єдиною складністю є, що необхідно зареєструвати свій додаток в консолі Google для отримання певних облікових даних (credentials), які використовується для взаємодії з API. Також є певні труднощі при локальному тестуванні (розробці) технології OAuth, не можна створити облікові дані (credentials) для локального хоста (127.0.0.1).

Для взаємодії з Google OAuth API використовується модуль oauth2client. Спочатку йому передаються credentials для ініціалізації, інакше взаємодія з API буде неможливо. Надалі відбувається запит для отримання URL аутентифікації, в запиті міститься так званий callback_url - це адреса на який Google перенаправляє користувача після авторизації, в запиті буде певний токен (code) який передасть Google. Завдяки йому може бути запрошена інформація про користувача. Найважливішим в ній для нас є E-mail оскільки по ньому ми ідентифікуємо користувача в нашій системі. Приклад вікна авторизації можна побачити на рисунку 4.4.

Якщо підсумувати вищесказане, то механізм працює приблизно таким чином:

– користувач робить запит авторизації;

– контролер додатків обробляє його і запитує URL для авторизації користувача в Google. Після чого користувач перенаправляється на отриманий URL.

Після авторизації Google перенаправляє користувача на callback_url, де контролер дістає E-mail користувача з запиту, після чого відбувається перевірка чи є в базі даних студент з таким E-mail-ом і в разі якщо успіху йому чіпляється певний токен для його подальшого визначення (токен також сохраняється в базу даних в модель Sessions).

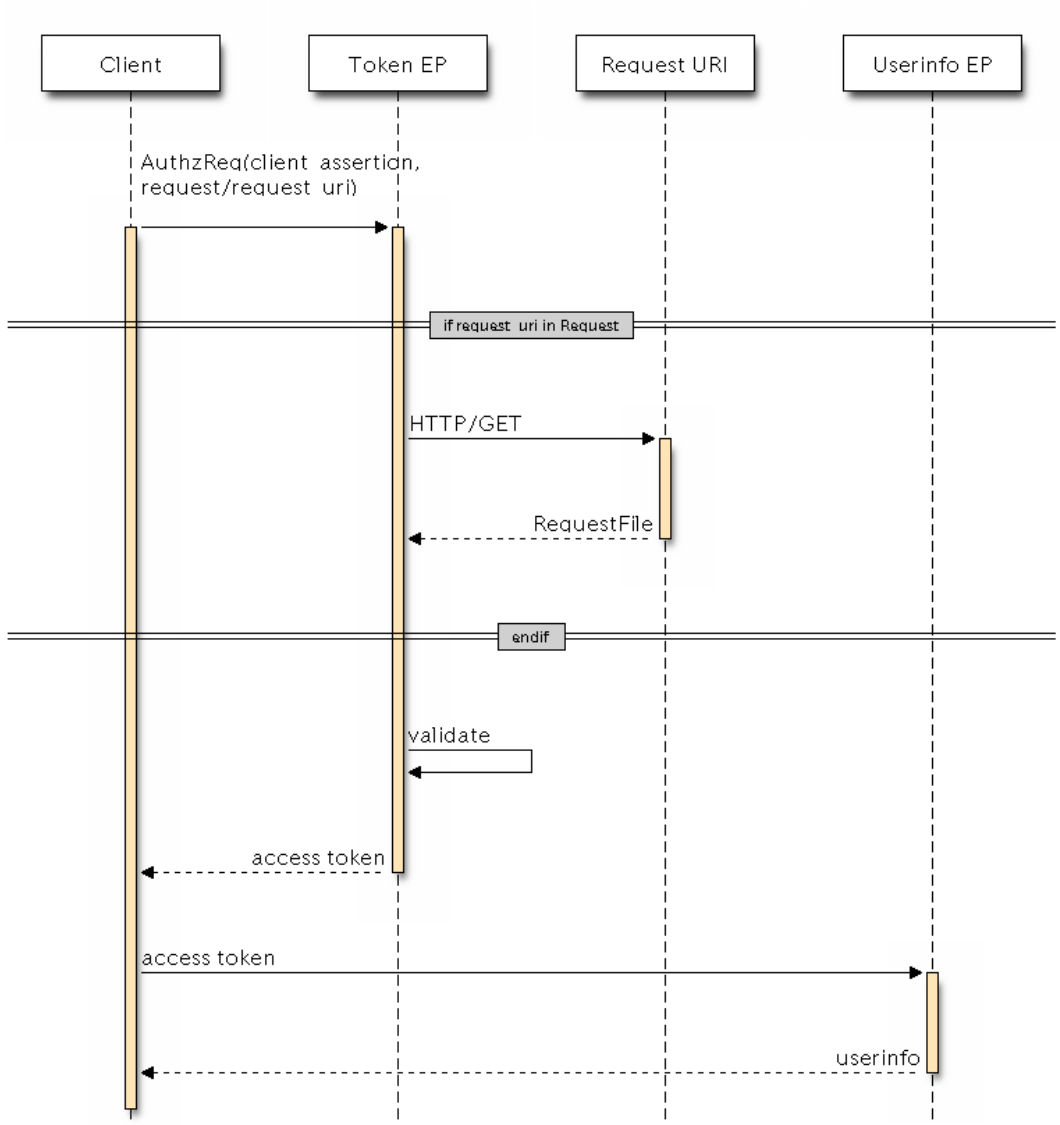


Рисунок 4.3 - Принцип роботи технології OAuth

Позитивною стороною використання технології OAuth є те, що не має сенсу зберігати паролі користувача і піклуватися про їх безпеки, оскільки за це відповідає Google. При цьому наша система також не взаємодіє з паролями студентів від їх Google акаунтів, що також може гарантувати певну безпеку і збереження їх даних.

На рисунку 4.4 зображено процес авторизації через Google OAuth, ф саме процес отримання дозволу на доступ до інформації користувача. Технологія OAuth не може гарантує, що користувач буде сповіщений щодо всіх запитуваних про нього даних в момент аутентифікації.

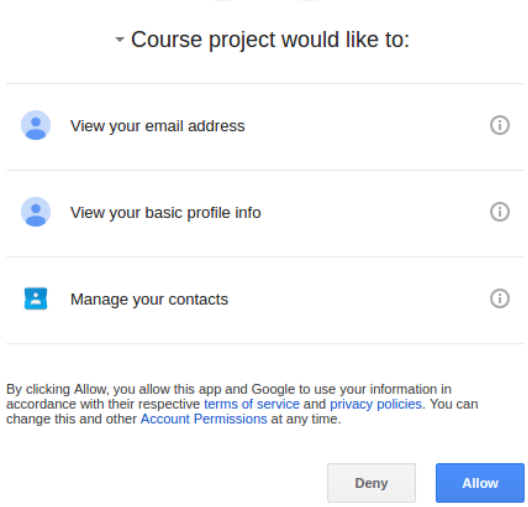


Рисунок 4.4 - Приклад авторизації через Google OAuth

На рисунку 4.5 зображено процес аутентифікації і її результат, якщо користувач намагається аутентифікуватися, використовуючи E-mail, який не внесено до бази даних.

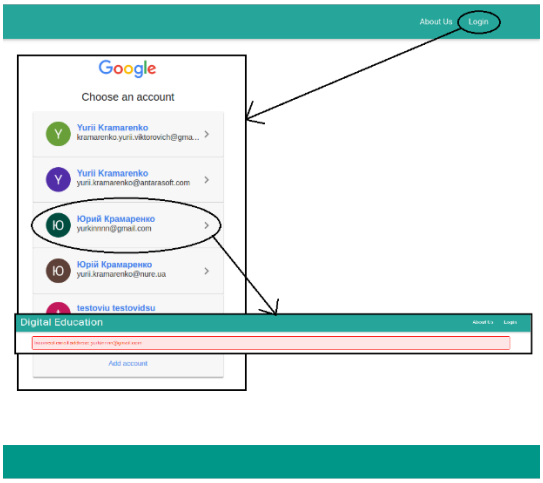


Рисунок 4.5 - Приклад аутентифікації через незареєстрований в системі E-mail

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1 Аналіз потенційних небезпечних і шкідливих виробничих чинників проектованого об’єкту, що мають вплив на персонал

У даному дипломному проекті розробляється програмне забезпечення навчального призначення. Розроблене програмне забезпечення орієнтоване на роботу з персональним комп'ютером. Експлуатовані для вирішення внутрішньовиробничих завдань ПЕОМ типу IBM PC мають наступні характеристики:

споживана потужність	220 Вт;
робоча напруга	220 В;
напруга джерел живлення	+12 В; - 12 В; +5 В;
робоча частота	50 Гц.

Виходячи з приведених характеристик, вочевидь, що для людини існує небезпека поразки електричним струмом, унаслідок недбалого поводження з комп'ютером і порушення правил експлуатації, залишення частин ПЕОМ, що знаходяться під напругою, відкритими або знятих для ремонту вузлів.

Відповідно до ГОСТ 12.1.005-88[20] до легкої фізичної роботи відносяться всі види діяльності, виконувані сидячи і ті, що не потребують фізичної напруги. Робота користувача ПК відноситься до категорії 1а.

При роботі на ПЕОМ користувач піддається ряду потенційних небезпек. Унаслідок недотримання правив техніки безпеки при роботі з машиною(невиконання огляду відкритих частин ПЕОМ, що знаходяться під напругою або знятих для ремонту вузлів) для користувача існує небезпека поразки електричним струмом.

Джерелами підвищеної небезпеки можуть служити наступні елементи:

- розподільний щит;
- джерела живлення;
- блоки ПЕОМ і друку, що знаходяться в ремонті.

Ще одна проблема полягає у тому, що спектр випромінювання комп'ютерного монітора включає рентгенівську, ультрафіолетову і інфрачервону області, а також широкий діапазон хвиль інших частот. Небезпека рентгенівського проміння мала, оскільки цей вид випромінювання поглинається речовиною екрану. Проте велику увагу

слід приділяти біологічним ефектам низькочастотних електромагнітних полів(аж до порушення ДНК).

Відповідно до ГОСТ 12.1.003-74, при обслуговуванні ПЕОМ мають місце фізичні і психофізичні небезпечні, а також шкідливі виробничі чинники:

– підвищене значення напруги в електричному ланцюзі, замикання якої може відбутися через тіло людини;
– підвищений рівень статичної електрики;
– підвищений рівень електромагнітних випромінювань;
– підвищена або знижена температура повітря робочої зони;
– підвищений або знижений рух повітря;
– підвищена або знижена вологість повітря;
– відсутність або недостатність природного світла;
– підвищена пульсація світлового потоку;
– недостатня освітленість робочого місця;
– підвищений рівень шуму на робочому місці;
– розумове перенапруження;
– емоційні навантаження;
– монотонність праці.

5.2 Заходи щодо техніки безпеки

Основним небезпечним чинником при роботі з ЕОМ є небезпека поразки людини електричним струмом, яка посилюється тим, що органи чуття людини не можуть на відстані знайти наявності електричної напруги на устаткуванні.

Проходячи через тіло людини, електричний струм чинить на нього складну дію, що є сукупністю термічної(нагрів тканин і біологічних середовищ), електролітичної(розкладання крові і плазми) і біологічної(роздратування і збудження нервових волокон і інших органів тканин організму) дій.

Тяжкість поразки людини електричним струмом залежить від цілого ряду чинників:

– значення сили струму;
– електричного опору тіла людини і тривалості протікання через нього струму;

– роду і частоти струму;

– індивідуальних властивостей людини і навколишнього середовища.

Розроблений дипломний проект передбачає наступні технічні способи і засоби, що застерігають людину від ураження електричним струмом:

– заземлення електроустановок;

– занулення;

– захисне відключення;

– електричне розділення ятерів;

– використовування малої напруги;

– ізоляція частин, що проводять струм;

– огорожа електроустановок.

Занулення зменшує напругу дотику і обмежує година, протягом якого людина, ткнувшись до корпусу, може потрапити під дію напруги.

Струм однофазного короткого замикання визначається по наближеній формулі:

$$I_K = \frac{\text{Uф}}{\text{Zп} + \frac{\text{Zт}}{3}}\ , \qquad (5.1)$$

де Uф - номінальна фазна напруга мережі, В;

Zп - повний опір петлі, створене фазними і нульовими дротами, Ом;

Zт - повний опір струму короткого замикання на корпус, Ом.

Згідно таблиці 4 ДСТУ 7237:2011 [23]: Zт /3 = 0,1 Ом.

Для провідників і жил кабелю для розрахунку повного опору петлі використовуємо формулу(4.2.) :

$$\text{Zп} = \sqrt{\text{Rп}^2 + \text{Xп}^2}\ , \qquad (5.2)$$

де Rп = Rф + Ro - сумарний активний опір фазного Rф і нульового Ro дротів, Ом;

Xп - індуктивний опір паяння дротів, Ом.

Перетин 1 км мідного дроту S = 2.5 мм, тоді згідно таблицям 5 і 6 ДСТУ 7237:2011 [23], має такий опір:

Хп = 0,11 Ом;

Rф = 7,55 Ом;

Rо = 7,55 Ом.

Отже, Rп = 7,55 + 7,55 = 15,1 Ом.

Тоді по формулі (5.2) знаходимо повний опір петлі :

$$Zп = \sqrt{15{,}1^2 + 0{,}11^2} \approx 15{,}1 \text{ (Ом).}$$

Струм однофазного короткого замикання рівний:

$$Iк = \frac{220}{15{,}1 + 0{,}1} = 14{,}47 \text{ (А).}$$

Дія плавкої вставки на ПЕОМ забезпечується, якщо виконується співвідношення:

$$Iк \geq k * Iн, \quad (5.3)$$

де Iн - номінальний струм спрацьовування плавкої вставки, А;

k - коефіцієнт кратності нелінійного струму Iн, А.

Коефіцієнт кратності нелінійного струму Iн розраховується по формулі (5.4.) :

$$Iн = P / U, \quad (5.4)$$

де P = 220 Вт - споживана потужність;

U = 220 В - робоча напруга;

k = 3 А - для плавких вставок.

Отже, Iн = 220 / 220 = 1 А.

Підставивши значення у вираз (4.3), одержимо:

14,47> 3*1.

Таким чином, доведено, що апарат забезпечить спрацьовування(і захист) при підвищенні номінального струму.

5.3 Заходи, що забезпечують виробничу санітарію і гігієну праці

Вимоги до виробничих приміщень встановлюються ДСП 173-96, СНіП, відповідними ГОСТами і ОСТами з урахуванням небезпечних і шкідливих чинників, що утворюються в процесі експлуатації електроустаткування.

Підвищення працездатності людини і збереження її здоров'я забезпечується стабільними метеорологічними умовами.

Мікроклімат виробничих приміщень визначається діючими на організм людини поєднаннями температури, вологості і швидкості руху повітря, а також температури навколишніх поверхонь. Значне коливання параметрів мікроклімату приводити до порушення систем кровообігу, нервової і потовидільної, що може викликати підвищення або пониження температури тіла, слабкість, запаморочення і навіть непритомність.

Відповідно до ГОСТ 12.1.005-88[20] встановлюють оптимальну і допустиму температуру, відносну вологість і швидкість руху повітря в робочій зоні . За відсутності надмірного тепла, вологи, шкідливих речовин в приміщенні досить природної вентиляції.

У приміщенні для виконання робіт операторського типу(категорія 1а), пов'язаних з нервово-емоційною напругою, проектом передбачається дотримання наступних нормованих величин параметрів мікроклімату (табл. 5.1).

Таблиця 5.1 - Санітарні норми мікроклімату робочої зони приміщень для робіт категорії 1а.

Пора року	Температура, С	Відносна вологість, %	Швидкість руху повітря, м/с
Холодна	22…24	40…60	0,1
Тепло	23…25	40…60	0,1

У приміщенні, де знаходиться ПЕОМ, повітрообмін реалізується за допомогою природної організованої вентиляції(з пристроєм вентиляційних каналів в перекриттях будівлі і вертикальних шахт) й установленого промислового кондиціонера фірми Samsung, який дозволяє вирішити переважну більшість завдань по створінню та підтримці необхідних параметрів повітряного середовища. Цей метод забезпечує приток потрібної кількості свіжого повітря, визначеного в СНіП (30 $м^3$ в годину на одного працівника).

Шум на виробництві має шкідливу дію на організм людини. Стомлення операторів через шум збільшує число помилок при роботі, призводить до виникнення травм. Для

оператора ПЕОМ джерелом шуму є робота принтера. Щоб усунути це джерело шуму, використовують наступні методи. При покупці принтера слід вибирати найбільш шумозахисні матричні принтери або з великою швидкістю роботи(струменеві, лазерні). Рекомендується принтер поміщати в найбільш віддалене місце від персоналу, або застосувати звукоізоляцію та звукопоглинання(під принтер підкладають демпфуючі підкладки з пористих звукопоглинальних матеріалів з листів тонкої повсті, поролону, пеноплену).

При роботі на ПЕОМ, проектом передбачені наступні методи захисту від електромагнітного випромінювання : обмеження часом, відстанню, властивостями екрану.

Обмеження годині роботи на ПЕОМ складає 3,5-4,5 години. Захист відстанню передбачає розміщення монітора на відстані 0,4-0,5 м від оператора. Передбачений монітор 20" TFT, LG 2043BW відповідає вимогам стандарту ТСО'03.

ТСО'03 пред’являє жорсткі вимоги в таких областях: ергономіка(фізична, візуальна і зручність користування), енергія, випромінювання(електричних і магнітних полів), навколишнє середовище і екологія, а також пожежна та електрична безпека, які відповідають всім вимогам ДСанПіН 3.3.2.007-98.

Для зниження стомлюваності та підвищення продуктивності праці обслуговуючого персоналу в колірній композиції інтер'єру приміщень для ПЕОМ дипломним проектом пропонується використовувати спокійні колірні поєднання і покриття, що не дають відблисків.

У проекті передбачається використання сумісного освітлення. У світлий час доби приміщення освітлюватиметься через віконні отвори, в решту часу використовуватиметься штучне освітлення.

Як штучне освітлення необхідно використовувати штучне робоче загальне освітлення. Для загального освітлення необхідно використовувати люмінесцентні лампи. Вони володіють наступними перевагами: високою світловою віддачею, тривалим терміном служби, хоча мають і недоліки: високу пульсацію світлового потоку.

При експлуатації ПЕОМ виробляється зорова робота. Відповідно до ДБН В.2.5-28-2006 ця робота відноситься до розряду 5а. При цьому нормоване освітлення на робочому місці(Ен) при загальному освітленні рівна 200 лк.

Приміщення завдовжки 12 м, шириною 10 м, заввишки 4 м обладнується світильниками типу ЛП02П, оснащеними лампами типу ЛБ зі світловим потоком 3120 лм кожна.

Виконаємо розрахунок кількості світильників в робочому приміщенні завдовжки a=12 м, шириною b=10 м, заввишки з=4 м, використовуючи формулу (5.5) розрахунку штучного освітлення при горизонтальній робочій поверхні методом світлового потоку:

$$n = (E \cdot S \cdot Z \cdot k)/(F \cdot U \cdot M), \quad (5.5)$$

де F - світловий потік = 3120 лм;

E - максимально допустима освітленість робочих поверхонь = 200 лк;

S - площа підлоги = 120 м2;

Z - поправочний коефіцієнт світильника = 1,2;

k - коефіцієнт запасу, що враховує зниження освітленості в процесі експлуатації світильників = 1,5;

n - кількість світильників;

U - коефіцієнт використовування освітлювальної установки = 0,6;

M - кількість ламп у світильнику =2.

З формули (5.5) виразимо n (5.6) і визначимо кількість світильників для даного приміщення:

$$n = (E \cdot S \cdot Z \cdot k)/(F \cdot U \cdot M), \quad (5.6)$$

Отже, $n = (200 \cdot 120 \cdot 1{,}2 \cdot 1{,}5)/(3120 \cdot 0{,}6 \cdot 2) = 12$.

Виходячи з цього, рекомендується використовувати 12 світильників. Світильники слід розміщувати рядами, бажано паралельно стіні з вікнами. Схема розташування світильників зображена на рис. 5.1.

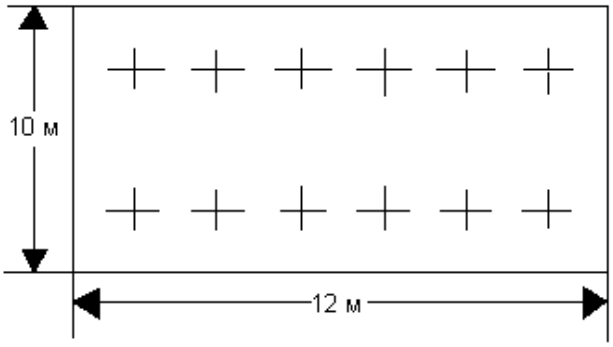


Рисунок 5.1 - Схема розташування світильників

5.4 Рекомендації по пожежній безпеці

Пожежі в приміщеннях, де встановлена обчислювальна техніка, представляють небезпеку для життя людини. Пожежі також пов’язані як з матеріальними втратами, так і з відмовою засобів обчислювальної техніки, що у свою чергу спричиняє за собою порушення ходу технологічного процесу.

Пожежа може виникнути при наявності горючої речовини та внесення джерела запалювання в горюче середовище. Пальними матеріалами в приміщеннях, де розташовані ПЕОМ, є:

– поліамід - матеріал корпусу мікросхеми, горюча речовина, температура самозаймання аерогелю 420 З ;

– полівінілхлорид - ізоляційний матеріал, горюча речовина, температура запалювання 335 З, температура самозаймання 530 З, кількість енергії, що виділяється при згоранні - 18000 - 20700 кДж/кг;

– стеклотекстоліт ДЦ - матеріал друкарських плат, важкозаймистий матеріал, показник горючості 1.74, не схильний до температурного самозаймання;

– пластика кабельний №489 - матеріал ізоляції кабелю, горючий матеріал, показник горючості більш 2.1;

– деревина - будівельний і обробний матеріал, матеріал з якого виготовлені меблі, горючий матеріал, показник горючості більше 2.1, теплота згорання 18731 - 20853 кДж/кг, температура запалювання 399 З, схильна до самозаймання [26].

Згідно НАПБ.Б. 03.002-2007 приміщення відносяться до категорії В(пожежовибухонебезпечним) і згідно правилам побудови електроустановок простір усередині приміщення відноситься до вогненебезпечної зони класу П - IIа (зони, розташовані в приміщеннях, в яких зберігаються тверді горючі речовини).

Потенційними джерелами запалення при роботі ПЕОМ є:

– іскри при замиканні і розмиканні ланцюгів;

– іскри і дуги коротких замикань;

– перегріви від тривалого перевантаження і наявності перехідного опору.

Продуктами згорання, що виділяються при пожежі, є : оксид вуглецю, сірчистий газ, оксид азоту, синильна кислота, акропеін, фосген, хлор та ін. При горінні пластмас, окрім звичайних продуктів згорання, виділяються різні продукти термічного розкладання:

хлорангідрідні кислоти, формальдегіди, хлористий водень, фосген, синильна кислота, аміак, фенол, ацетон, стирол та ін., що шкідливо впливають на організм людини.

Для захисту персоналу від дії небезпечних і шкідливих чинників пожежі проектом передбачається застосування промислового протигаза з коробкою марки В(жовта).

Пожежна безпека об'єктів народного господарства регламентується ГОСТ 12.1.004-91[22] і забезпечується системами запобігання пожежам і протипожежному захисту. Для успішного гасіння пожеж вирішальне значення має швидке виявлення пожежі і своєчасний виклик пожежних підрозділів до місця пожежі.

Зменшити горюче навантаження не представляється можливим, тому проектом передбачається застосувати наступні способи і їх комбінації для запобігання утворенню(внесення) джерел запалення :

– застосування устаткування, що задовольняє вимогам електростатичної безпеки;

– застосування в конструкції швидкодіючих засобів захисного відключення можливих джерел запалення;

– виключення можливості появи іскрового заряду статичної електрики в горючому середовищі з енергією, рівної і вище мінімальної енергії запалення;

– підтримка температури нагріву поверхні машин, механізмів, устаткування, пристроїв, речовин і матеріалів, які можуть увійти до контакту з пальним середовищем, нижче гранично допустимої, становить 80% якнайменшої температури самозаймання пального.

– заміна небезпечних технологічних операцій більш безпечними;

– ізольоване розташування небезпечних технологічних установок і устаткування;

– зменшення кількості пальних і вибухонебезпечних речовин, що знаходяться у виробничих приміщеннях;

– запобігання можливості утворення пальних сумішей на лінії, вентиляційних системах і ін.;

– механізація, автоматизація та справність(потокова) виробництва;

– суворе дотримання стандартів і точне виконання встановленого технологічного режиму;

– запобігання можливості появи в небезпечних місцях джерел запалення;

– запобігання розповсюдженню пожеж і вибухів;

– використовування устаткування і пристроїв, при роботі яких не виникає джерел запалення;

– виконання вимог сумісного зберігання речовин і матеріалів;

– наявність громовідводу;

– організація автоматичного контролю параметрів, що визначають джерела запалення;

– ліквідація можливості самозаймання речовин і матеріалів .

Для запобігання пожежі в обчислювальних центрах проектом пропонується виконання наступних вимог :

– електроживлення ЕОМ повинно мати автоматичне блокування відключення електроенергії на випадок зупинки системи охолоджування і кондиціонування;

– система вентиляції обчислювальних центрів повинна бути обладнана блокуючими пристроями, що забезпечують її відключення на випадок пожежі;

– робочі місця повинні бути оснащені пожежними щитами, сигналізацією, засобами для сповіщення про пожежну небезпеку (телефонами), медичними аптечками для надання першої медичної допомоги, розробленим планом евакуації.

Для зниження пожежної небезпеки в приміщеннях використовуються первинні засоби гасіння пожеж, а також система автоматичної пожежної сигналізації, яка дозволяє знайти початкову стадію загоряння, швидко і точно оповістити службу пожежної охорони про час і місце виникнення пожежі.

Відповідно до правил пожежної безпеки для промислових підприємств приміщення категорії В підлягають устаткуванню системами автоматичної пожежної сигналізації. Проектом передбачається застосування датчика типу ІДФ - 1(димовий фотоелектричний датчик), оскільки специфікою пожеж обчислювальної техніки і радіоапаратури є, в першу чергу, виділення диму, а потім - підвищення температури.

При виникненні пожежі в робочому приміщенні обслуговуючий персонал зобов'язаний негайно вжити заходи по ліквідації пожежі. Для ліквідації пожежі використовують вогнегасники (хімічно-пінні, пінні для повітря ОП-5, ОП-6, ОП-9, вуглекислотні ОУ-5), пісок, пожежний інвентар(сокири, ломи, багри, шерстяну або азбестову ковдри). Як засіб індивідуального захисту проектом передбачається використовування промислового протигаза з маскою, фільтруючої коробки В.

В якості організаційно-технічних заходів рекомендується проводити навчання робочого персоналу правилам пожежної безпеки.

5.5 Вплив інформаційних технологій на навколишнє середовище.

Бурхливий розвиток науки і техніки в кінці XX століття привело до широкого поширення інформаційних технологій в різних сферах діяльності людини. Сучасна людина схиблений на технологіях.

Нас оточують комп'ютери, мобільні телефони, автомобілі, навігатори і гори інших високотехнологічних речей. Ми звикли покладатися на технології:

– ми дзвонимо по стільникового зв'язку, щоб вирішити важливе для нас питання;

– ми живемо в інтернеті;

– ми мчимо на зустріч з діловим партнером на потужному швидкісному автомобілі або розмовляємо з друзями і знайомими по скайпу.

Технологічні процеси, що приносять людству все нові і нові блага і зручності, як правило, є прямими (або непрямими) джерелами забруднення навколишнього середовища.

Як ми бачимо, бурхлива діяльність людини по освоєнню природних ресурсів, перенаселення, забруднення атмосфери, знищення флори і фауни Землі обіцяє мало чого хорошого і вже зараз стає причиною великої кількості катастроф.

Але не одна тільки промисловість забруднює навколишнє середовище. Спеціалісти по боротьбі із забрудненням навколишнього середовища виміряли згубний вплив пошуку за допомогою Google на природу-матінку. В середньому один гуглопошук призводить до викиду в атмосферу 7 грамів вуглекислого газу, що відповідає половині емісії CO2 від кип'ятіння води в чайнику.

До такого висновку прийшов гарвардський фізик Алекс Гросс, який досліджує вплив комп'ютерної індустрії на екологію. Крім цього, як стверджує вчений: «Пошуковий алгоритм оптимізований на швидке отримання результату, а не на економію електроенергії».

Навіть інтернет здатний, хоча і побічно, впливати на екологію світу. Наприклад, на пересилку реклами в інтернеті в рік витрачається більше 30 мільярдів КВТ / г, що призводить до викиду більше 17 мільярдів тон вуглекислого газу в атмосферу. Таким чином, сумарно, по викиду вуглекислого газу сучасні інформаційні технології випереджають авіацію. І відповідно з розвитком технологій передачі інформації кількість CO2 в атмосфері буде збільшуватися.

ВИСНОВКИ

В ході виконання даної атестаційної роботи були проаналізовані різні хмарні технології, що надаються нам компаніями Google і Amazon. Серед них були виділені самі відповідні технології для організації самостійної роботи студента і контролю його навчальної діяльності:

- Google:
  1) Google Drive - як сховище напрацювань.
  2) Google Docs - як текстовий редактор.
  3) Google Calendar - для організації time-management-а студента.
  4) Gmail - для спрощення комунікації.
  5) Hangouts - для проведення Webinar-ів і онлайн консультацій.
- Amazon:
  1) EC2 - як хостингу.
  2) ELB - для балансування навантаження, якщо не справляється 1 машина.
  3) S3 - при необхідності зберігання великого обсягу даних.

Як фреймворк для серверної частини був обраний Python Tornado завдяки його асинхронності. Це позитивна властивість може допомогти заощадити величезну кількість апаратних засобів і при цьому домогтися максимальної продуктивності. Також варто зауважити, що розробка на Python досить проста і швидкість розробки дуже висока. Ідеальною архітектурою для досягнення поставленої мети можна вважати мікросервісний підхід, оскільки кожен з модулів системи може бути реалізований як окремий мікросервіс з метою подальшого перевикористання і спрощення зміни логіки роботи програми, в разі появи такої необхідності. Але оскільки програми не є достатньо великим, а багато функцій виконуються хмарними технологіями від Google, то монолітна архітектура також цілком підходить, хоча в подальшому може призвести до деяких проблем.

Для реалізації клієнтської частини системи цілком може підійти будь-який фреймворк або бібліотека, але був обраний React.js. У ньому використовується компонентний підхід, що дозволяє писати набагато менше програмного коду оскільки ми заново використовуємо вже написані компоненти (кожен з яких є автономним). Також React був одним з перших клієнтських фреймворків для якого вирішена проблема StateManagement. Ця проблема дуже актуальна на даний момент, оскільки з нею

стикаються практично всі розробники, які розробляють складний, з точки зору функціоналу (незалежно від фреймворка або бібліотеки), веб-клієнт.

У розділі «Охорона праці» виконано аналіз потенційних небезпек при роботі із засобами обчислювальної техніки і механізмами, розроблені заходи щодо техніки безпеки, заходи, які забезпечують виробничу санітарію і гігієну праці, розраховане штучне освітлення, виконані рекомендації по пожежній безпеці.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Что такое облачные вычисления [Электронный ресурс] . – Режим доступа: https://www.ibm.com/cloud-computing/ru/ru/what-is-cloud-computing.html . –Дата доступа: 22.01.2017. – Загл. с экр.

2. Облачные вычисления, краткий обзор или статья для начальника [Електронный ресурс] . – Режим доступа: https://habrahabr.ru/post/111274 . – 12.11.2016 г.

3. Google Cloud Platform [Электронный ресурс] . – Режим доступа: https://cloud.google.com/ . – Дата доступа: 22.01.2017. – Загл. с экр.

4. Amazon Web Services [Электронный ресурс] . – Режим доступа: https://aws.amazon.com/ru/types-of-cloud-computing/ . – Дата доступа: 22.01.2017. – Загл. с экр.

5. Amazon Web Services Products [Электронный ресурс] . – Режим доступа: https://aws.amazon.com/ru/products/ . – Дата доступа: 22.01.2017. – Загл. с экр.

6. OpenStack Software [Електронный ресурс] . – Режим доступа: https://www.openstack.org/software/ . – – Дата доступа: 22.01.2017. – Загл. с экр.

7. Анатольев А.Г. Компоненты сетевого приложения. Клиент-серверное взаимодействие и роли серверов [Електронный ресурс] / А.Г. Анатольев – Режим доступа: http://www.4stud.info/networking/lecture5.html . – Дата доступа: 22.01.2017. – Загл. с экр.

8. Кулаков В. Справочник по шаблонам проектирования [Электронный ресурс] / В. Кулаков. – Режим доступа: http://design-pattern.ru/patterns/mvc.html . – Дата доступа: 22.01.2017. – Загл. с экр.

9. Определение шаблона MVVM [Електронный ресурс] . – Режим доступа: http://metanit.com/sharp/wpf/22.1.php . – Дата доступа: 22.01.2017. – Загл. с экр.

10. What the Flux? An Overview of the React State Management Ecosystem [Електронный ресурс] . – Режим доступа: http://thenewstack.io/flux-overview-react-state-management-ecosystem/ . – Дата доступа: 22.01.2017. – Загл. с экр.

11. Определение шаблона MVVM [Електронный ресурс] . – Режим доступа: http://metanit.com/sharp/wpf/22.1.php . – Дата доступа: 22.01.2017. – Загл. с экр.

12. Introduction to Facebook's Flux architecture [Електронный ресурс] . – Режим доступа: https://ryanclark.me/getting-started-with-flux/ . – Дата доступа: 22.01.2017. – Загл. с экр.

13. Redux Official Documentation [Електронный ресурс]. – Режим доступа: https://docs.reduxframework.com . – Дата доступа: 22.01.2017. – Загл. с экр.

14. Dirr. How to out curate your competitors [Електронныйресурс] / Ragan – Режимдоступа: http://www.ragan.com/Main/Articles/How_to_out_curate_your_competitors _45613.aspx – 18.09.2016 г. – How to out curate your competitors.

15. Canali De Rossi, L. Real-Time News Curation, Newsmastering And Newsradars – The Complete Guide Part 1: Why We Need It [Електронныйресурс] / Master new media – Режимдоступа: http://www.masternewmedia.org/real-time-news-curation-newsmasteringand-newsradars-the-complete-guide-part-1/ – 11.11.2016 г. – Why We Need It.

16. Aders, T. Content Curation: Interview with Robert Scoble [Електронныйресурс] / Social Media Today – Режимдоступа: http://socialmediatoday.com/tatiana-aders/1783216/robert-scoble-contentcuration-interview– 12.11.2016 г. – Interview with Robert Scoble.

17. Охрана труда в вычислительных центрах. Ю.Г. Сибаров, Н,Н, Сколотнев, В.К. Васин и др. – Машиностроение, 1990 – 192 с.

18. ГОСТ 12.1.003-83. ССБТ. Шум. Общие требования безопасности.

19. ГОСТ 10.1.004-85. ССБТ. Пожарная безопасность. Общие требования.

20. ГОСТ 12.1.005-88. ССБТ. Общие санитарно-гигиенические правила к воздуху рабочей зоны.

21. ГОСТ 12.1.006-76. ССБТ. Электромагнитные поля радиочастот. Общие требования безопасности.

22. ГОСТ 12.1.004-91. ССБТ. Пожарная безопасность. Общие требования.

23. ДСТУ 7237:2011. Электробезопасность. Общие требования и номенклатура видов защиты.

24. ГОСТ 12.1.029-80. ССБТ. Средства защиты от шума. Классификация.

25. ГОСТ 12.1.030-81. ССБТ. Электробезопасность. Защитное заземление, зануление.

26. ДБН В 2.5-28:2015. Естественное и искусственное освещение.

27. ДБН В.2.5-67:2013. Отопление, вентиляция и кондиционирование.

28. ДСТУ ГОСТ 12.1..012:2008. Система стандартов безопасности труда.

ДОДАТОК А.

Організація Redux архітектури для панелі адміністрування викладача

```
// actions/ActionTypes.js
export const SAVE_TASK = "SAVE_TASK";
export const ADD_NEW_TASK = "ADD_NEW_TASK";
export const REMOVE_TASK = "REMOVE_TASK";
export const CHANGE_GROUP = "CHANGE_GROUP";

// actions/Actions.js
import * as actionTypes from './ActionTypes';
export function addnewTask () {
    return {
        type: actionTypes.ADD_NEW_TASK
    }
}
export function removeTask (id) {
    return {
        type: actionTypes.REMOVE_TASK,
        id: id
    }
}
export function saveTask (task) {
    return {
        type: actionTypes.SAVE_TASK,
        task: task
    }
}
export function changeGroup (id) {
    return {
        type: actionTypes.CHANGE_GROUP,
        id: id
    }
}

// redusers/GroupReducer.js
import { CHANGE_GROUP } from '../actions/ActionTypes';
import _ from 'lodash';

export default function(state = window.groupsInitialState, action) {
    switch (action.type) {
        case CHANGE_GROUP:
            return Object.assign({}, state, { currentGroup: action.id });

        default:
            return state;
    }
}

// redusers/TasksReducer.js
import * as actionTypes from '../actions/ActionTypes';
import _ from 'lodash';

export default function (state = window.tasksInitialState, action) {
    switch( action.type ) {
```

```
        case actionTypes.ADD_NEW_TASK:
            return {...state, editing: true, currentTask: -1 };
        case actionTypes.REMOVE_TASK:
            _.remove(state.tasks, task =>task.id == action.id);
            return { ...state };
        default:
            return state
    }
}

// store/mainStore.js
import { createStore, combineReducers } from 'redux';
import tasksReducer from '../reducers/TasksReducer';
import groupReducer from '../reducers/GroupReducer';

export default createStore(
    combineReducers({
        tasks: tasksReducer,
        group: groupReducer
    })
);

// index.js

import React from 'react'
import Head from 'next/head';
import { bindActionCreators } from 'redux';
import { Provider, connect } from 'react-redux';
import store from '../src/store/mainStore';
import * as actions from './../src/actions/Actions'

const mapStateToProps = ( state ) => {
    return {...state};
};

const mapDispatchToProps = ( dispatch ) => {
    return {
        actions: bindActionCreators(actions, dispatch)
    }
};
class PageContainer extends React.Component {
    constructor(props) {
        super(props);
    }
    render() {
        return (
            <div className="container">
                <div className="divider"></div>
                <div className="row">
                    <div className="col-xs-12">
                        <h5>Add students task</h5>
                    </div>
                    <div className="col-xs-6">
                        <div className="row">
                            <div className="input-field col s12">
                                <select value={this.props.group.currentGroup}>
                                    <option    value=""    disabled    selected>Choose
group</option>
```

```
                                    {this.props.group.groups.map((item)=>     <option
value={item.id}>{item.title}</option>)}
                                </select>
                                <label>Select a group</label>
                            </div>
                            <div className="input-field col s12">
                                <select>
                                    <option value="" disabled>Choose task</option>
                                    {this.props.tasks.tasks.map((item)=>      <option
value={item.id}>{item.title}</option>)}
                                </select>
                                <label>Select a task for group</label>
                            </div>
                        </div>
                    </div>
                    <div className="col-xs-6">
                        <div className="row">
                            <div className="input-field col s12">
                                <input    value=""    id="first_name2"    type="text"
className="validate" />
                                <label                                 className="active"
for="first_name2">Title</label>
                            </div>
                            <div className="input-field col s12">
                                <textarea    id="textarea1"    className="materialize-
textarea"/>
                                <label for="textarea1">Text</label>
                            </div>
                        </div>
                    </div>
                    <div className="col-xs-12">
                        <a       className="waves-effect       waves-light       btn"
style={{marginRight: "10px"}}>Add</a>
                        <a       className="waves-effect       waves-light       btn"
style={{marginRight: "10px"}}>Delete</a>
                        <a       className="waves-effect       waves-light       btn"
style={{marginRight: "10px"}}>Save</a>
                    </div>
                </div>

            </div>
        );

    }
}
const PageContainerConnected = connect(
    mapStateToProps,
    mapDispatchToProps
)( PageContainer );
class Container extends React.Component {
    constructor(props) {
        super(props);

        this.store = store;
    }

    static async getInitialProps({req}) {
        return {}
    }
```

```
    componentDidMount(){
        $(document).ready(function() {
            $('select').material_select();
        });
    }
    render() {
        return (
                <div className="main">
                    <Head>
                        <title>My page title</title>
                        <meta charSet="utf-8"/>
                            <title>Course project constructor</title>
                            <link               href="/static/css/bootstrap.grid.css"
rel="stylesheet" />
                            <link              href="/static/css/materialize.min.css"
rel="stylesheet" />
                            <link href="/static/css/styles.css" rel="stylesheet" />
                            <script src="/static/js/jq.js"></script>
                            <script src="/static/js/init.js"></script>
                            <script src="/static/js/materialize.min.js"/>
                    </Head>
                    <nav className="teal lighten-1" role="navigation">
                        <div   className="nav-wrapper   container-big"><a   id="logo-
container" href="/" className="brand-logo">Digital Education</a>
                            <ul className="right hide-on-med-and-down">
                                <li      className="waves-effect      waves-light"><a
href="#">About Us</a></li>
                                <li      className="waves-effect      waves-light"><a
href="/login">Login</a></li>
                            </ul>
                            <ul id="nav-mobile" className="side-nav">
                                <li><a href="#">Navbar Link</a></li>
                                <li><a href="/login">Login</a></li>
                            </ul>
                            <a          href="#"          data-activates="nav-mobile"
className="button-collapse"><i className="material-icons">menu</i></a>
                        </div>
                    </nav>
                    <Provider store={this.store}>
                        <PageContainerConnected />
                    </Provider>

                    <footer className="page-footer teal">
                        <div className="container-big">
                            <div className="row shades-text text-white">
                                SNU
                            </div>
                        </div>
                    </footer>
                </div>
        )
    }
}
export default Container;
```

ДОДАТОК Б.

Електронні плакати

Східноукраїнський національний університет ім.В.Даля

Магістерська робота

Інформаційні технології в системах управління проектними роботами

Магістрант ст.гр. ІТП-16зм: Кравченко О.В.

Керівник: Кривуля Г.Ф.

Сєвєродонецьк 2018

Хмарні технології в CyberUnivercity

2

Самостійна робота студента

Самостійна робота студентів за Законом України «Про вищу освіту», прийнятому в 2014 році, є формою організації освітнього процесу. Окремо в законі визначено, що система внутрішнього забезпечення якості проведення освітнього процесу у вузі передбачає забезпечення наявності необхідних ресурсів для організації самостійна роботі студентів по кожній освітній програмі.

Підвищення ролі самостійної роботи студентів в процесі навчання у вузі вимагає відповідної організації освітнього процесу:

. модернізації навчально-методичного забезпечення,

. розробки нових дидактичних засобів навчання, спрямованих на ефективну організацію самостійної роботи студентів,

. створення за допомогою інформаційних технологій зручних форм підтримки самостійної роботи студентів,

. створення нових форм і методів контролю і самоконтролю знань студентів.

3

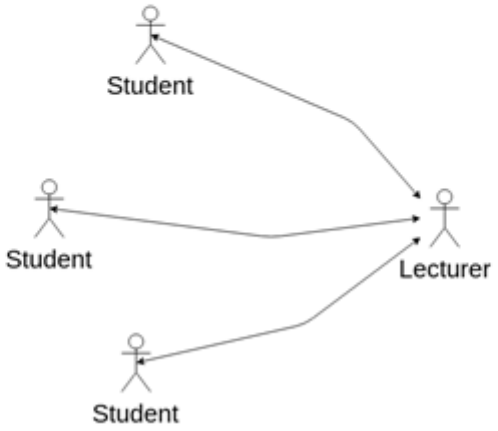
Самостійна робота студента

Індивідуальні завдання є однією з форм організації самостійної роботи студентів, яка має на меті поглиблення, узагальнення та закріплення знань, які студенти отримують в процесі навчання, а також застосування цих знань на практиці. До індивідуальних завдань відносяться реферати, РГЗ, курсові та дипломні проекти (роботи) і ін. Особливістю індивідуальних завдань проектного напрямку є наявність керівника проекту, з яким студент повинен мати можливість при необхідності організувати інформаційну зв'язок і обмін навчальними і проектними матеріалами.

4

Проблема

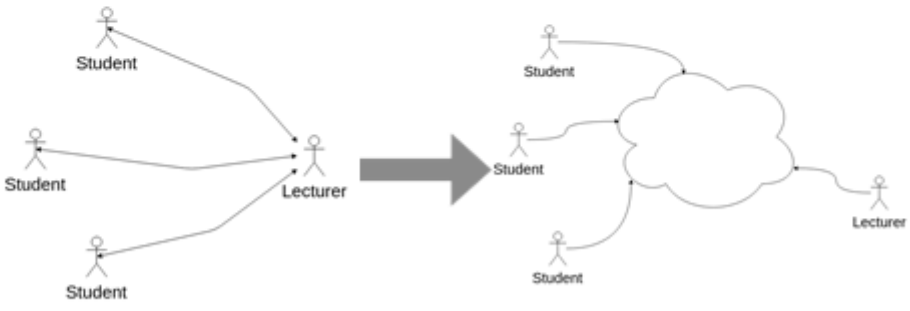
Не завжди студент може регулярно відвідувати університет і консультуватися з викладачем стосовно своєї курсової (дипломної) роботи. Це може значно позначитися на отриманих результатах.



5

Рішення

Надати студенту та преподавтель ресурс, що дозволяє студенту віддалено розробляти курсову (дипломну) роботу, а викладачеві контролювати процес розробки і консультувати студента.



6

Мета і завдання дослідження

Предмет дослідження - моделі, технології та архітектурні рішення, використовувані при розробці клієнт-серверних освітніх систем з використанням хмарних технологій.

Мета дослідження - розробити систему управління проектною діяльністю студентів з використанням хмарних технологій.

1) **Розробити загальну структуру системи, розбити на систему на ряд модулів, описати взаємодія між ними.**
2) **Розробити принципи взаємодії зі сторонніми сервісами, провести дослідження інших рішень.**
3) **Розробити систему аутентифікації пов'язану з доменом nure.ua.**
4) **Сформулювати вимоги до системи засновані на користувальницьких історіях.**
5) **Розробити архітектуру системи, а також архітектуру бази даних.**
6) **Реалізувати систему і провести інтеграцію зі сторонніми сервісами.**

7

Хмарні технології

- Amazon
  - EC2
  - ELB
  - S3
- Google
  - Google Docs
  - Google Sheets
  - Google Drive
- OpenStack

OpenStack — комплекс проектів вільного програмного забезпечення, який може бути використаний для створення інфраструктурних хмарних сервісів і хмарних сховищ, при тому як публічних, так і приватних.

8

Суб'єкти і їх вимоги до системи (Призначені для користувальницьких історій)

• Студент	• Як студент я хотів би мати можливість віддалено виконувати курсову (дипломну) роботу і отримувати консультацію від викладача.
• Викладач	• Як викладач я хотів би віддалено стежити за виконанням курсової (дипломної) роботи студента і консультувати його.
• Адміністратор	• Як адміністратор я хотів би мати можливість контролювати дані про суб'єктів в системі.

9

Суб'єкти і їх вимоги до системи (Призначені для користувальницьких історій)

• Студент	• Як студент я хотів би мати можливість віддалено виконувати курсову (дипломну) роботу і отримувати консультацію від викладача.
• Викладач	• Як викладач я хотів би віддалено стежити за виконанням курсової (дипломної) роботи студента і консультувати його.
• Адміністратор	• Як адміністратор я хотів би мати можливість контролювати дані про суб'єктів в системі.

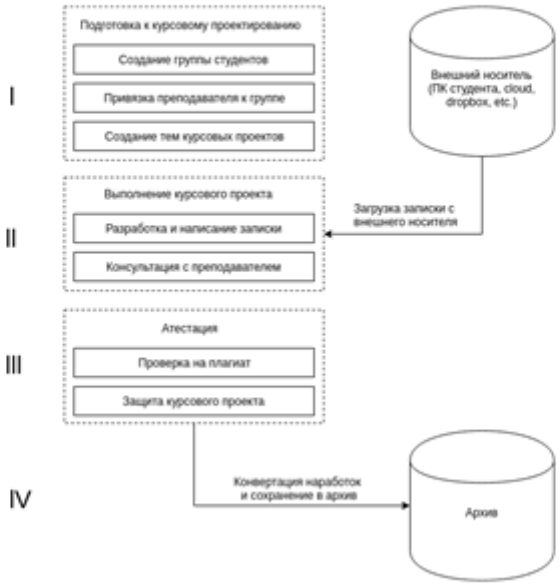
9

Суб'єкти і їх вимоги до системи (Призначені для користувальницьких історій)

• Студент	• Як студент я хотів би мати можливість віддалено виконувати курсову (дипломну) роботу і отримувати консультацію від викладача.
• Викладач	• Як викладач я хотів би віддалено стежити за виконанням курсової (дипломної) роботи студента і консультувати його.
• Адміністратор	• Як адміністратор я хотів би мати можливість контролювати дані про суб'єктів в системі.

9

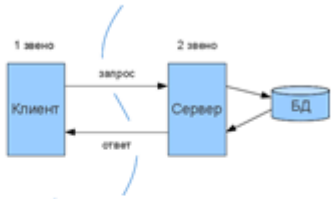
Бізнес процеси у системі ККП



12

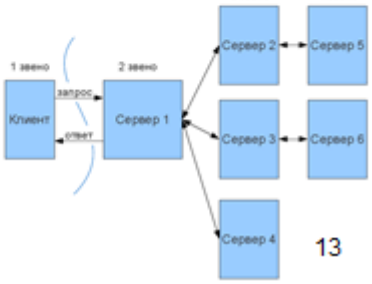
Організація клієнт-серверних систем

- Дволанкова архітектура



- Трохланкова архітектура

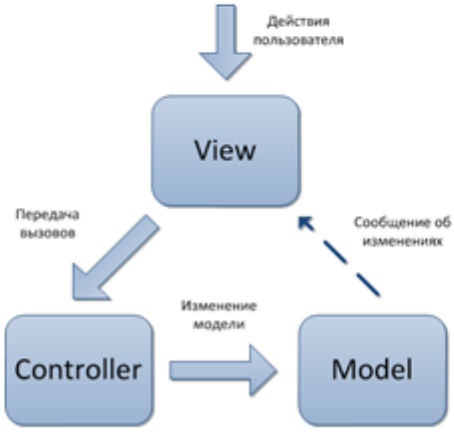
- N-ланкова (мікросервісна) архітектура



13

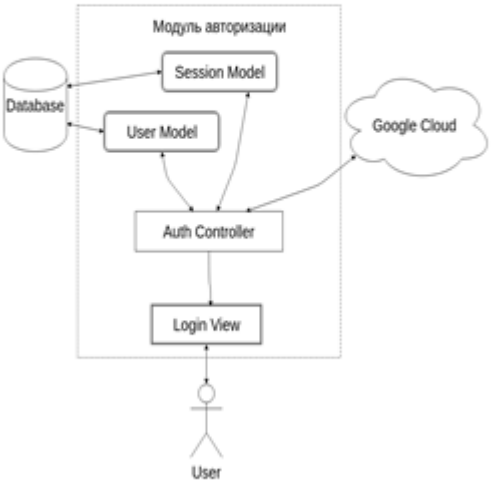
Организация клієнт-серверних додатків (модель MVC)

Система написана (Backend) на фреймворку Tornado, на основі моделі MVC, з метою розмежування логіки і спрощення розробки та внесення правок в систему.



14

Реалізація модуля аутентифікації



Програмний код даного контролера:

```
class OathApiHandler(BaseAPIHandler):
    @gen.coroutine
    def get(self):
        flow = flow_from_clientsecrets('credentials/client_secrets.json',
                                       scope='https://www.googleapis.com/auth/userinfo.email',
                                       redirect_uri='http://edu-space.com:8888/login')
        if 'code' not in self.request.arguments.keys():
            url = flow.step1_get_authorize_url()
            self.redirect(url)
        else:
            code = self.get_argument('code')
            credentials = flow.step2_exchange(code)
            http_auth = credentials.authorize(httplib2.Http())
            me = http_auth.request('https://www.googleapis.com/oauth2/v1/userinfo?alt=json')
            user_google_dict = json.loads(me[1].decode('utf-8'))

            user = yield User.get_one_by('email', user_google_dict['email'])
            if user:
                session = yield Session.create(user['_id'])
            elif '@nure' not in user_google_dict['email']:
                self.render('error-auth.html', error='Incorrect email address',
                            user=None)
                return
            else:
                user_def = defaultdict(lambda: '', user_google_dict)
                result_id = yield User.create_from_google_json(user_def)
                session = yield Session.create(result_id)

            self.set_secure_cookie('token', session['token'])

            self.redirect('/')
```

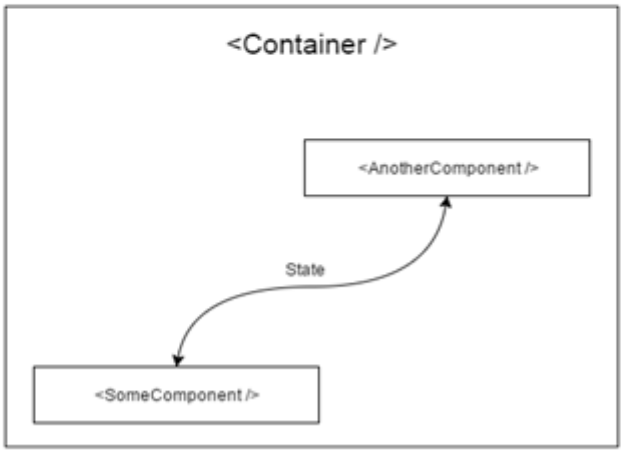
15

Приклад аутентифікації

16

React і проблема StateManagement-1

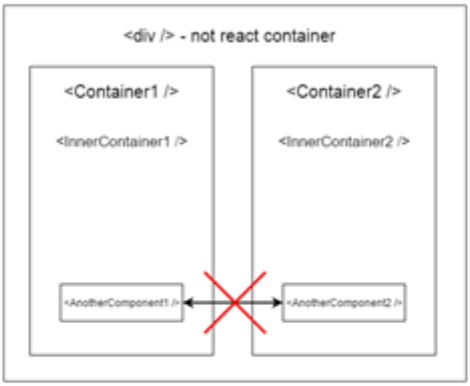
React.js - бібліотека (фреймворк) для створення користувацьких інтерфейсів, яка вирішує проблему часткового оновлення сторінки.




```
<Container>
        <SomeComponent
/>
        <AnotherCompone
nt />
</Container>
```

17

React і проблема StateManagement-2



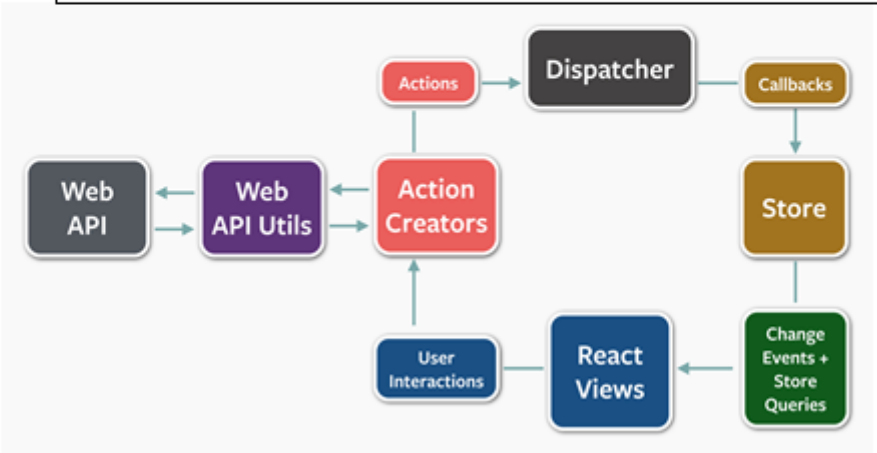
З ростом додатків, прямий обмін станами декількох компонентів може стати неможливий або сильно ускладнений через глибокої вкладеності компонентів.

Дана проблема вирішується за допомогою правильно організованого StateManagement-у. Кращим варіантом рішення є технологія (бібліотека) Redux.

18

Архітектура Redux

Redux - це бібліотека, яка реалізує контейнер для даних додатка, завдяки об'єднанню ідей від таких бібліотек, як FLUX і ELM. За допомогою Redux, є можливість керувати будь-яким станом даних, за умови дотримання таких принципів



19

Висновки

• В ході виконання даної атестаційної роботи були проаналізовані різні хмарні технології, що надаються нам компаніями Google і Amazon. Серед них були виділені самі відповідні технології для організації самостійної роботи студента і контролю його навчальної діяльності: Google Drive, Google Docs, Google Calendar, Gmail, Hangouts, EC2, ELB, S3 (при необхідності зберігання великого обсягу даних).

• Як фреймворк для серверної частини був обраний Python Tornado завдяки його асинхронності.

• Для реалізації клієнтської частини системи відмінно підходить React.js в комбінації з бібліотекою Redux.

20